# Approximation Algorithms for Action-Reward Query-Commit Matching

Mahsa Derakhshan * Andisheh Ghasemi † Calum MacRury ‡

**Abstract**

Matching problems under uncertainty arise in applications such as kidney exchange, hiring, and online marketplaces. A decision-maker must sequentially explore potential matches under local exploration constraints, while committing irrevocably to successful matches as they are revealed. The *query-commit matching problem* captures these challenges by modeling edges that succeed independently with known probabilities and must be accepted upon success, subject to vertex patience (time-out) constraints limiting the number of incident queries.

In this work, we introduce the *action-reward query-commit matching problem*, a strict generalization of query-commit matching in which each query selects an action from a known action space, determining both the success probability and the reward of the queried edge. If an edge is queried using a chosen action and succeeds, it is irrevocably added to the matching, and the corresponding reward is obtained; otherwise, the edge is permanently discarded. We study the design of approximation algorithms for this problem on bipartite graphs.

This model captures a broad class of stochastic matching problems, including the *sequential pricing problem* introduced by Pollner, Roghani, Saberi, and Wajc (EC 2022). For this problem, computing the optimal policy is NP-hard even when one side of the bipartite graph consists of a single vertex. On the positive side, Pollner et al. designed a polynomial-time approximation algorithm achieving a ratio of 0.426 in the one-sided patience setting, which degrades to 0.395 when both sides have bounded patience. These guarantees rely on a linear program that cannot be rounded beyond a factor of 0.544 in the worst case.

In this work, we design computationally efficient algorithms for the action-reward query-commit in one-sided and two-sided patience settings, achieving approximation ratios of $1-1/e \approx 0.63$ and $\frac{1}{27}\left(19-67/e^3\right) \approx 0.58$ respectively. These results improve the state of the art for the sequential pricing problem, surpassing the previous guarantees of 0.426 and 0.395. A key ingredient in our approach is a new configuration linear program that enables more effective rounding.

*Khoury College of Computer Sciences, Northeastern University, m.derakhshan@northeastern.edu

†Khoury College of Computer Sciences, Northeastern University, ghasemi.e@northeastern.edu

‡School of Industrial and Systems Engineering (ISyE), Georgia Tech, calum.macrury@isye.gatech.edu

# 1 Introduction

Matching problems under uncertainty appear in diverse applications, from kidney exchange to hiring and online marketplaces, where a set of feasible matches is known in advance, but the rewards for each match are uncertain. In such settings, a decision-maker may adaptively query information about potential matches while facing natural constraints on how many options can be explored per participant. Moreover, each query may require an immediate commitment to accept or reject a match based on its revealed outcome. These constraints give rise to a challenging algorithmic problem: selecting which matches to explore, subject to local query budgets, while making irrevocable decisions in order to achieve high overall reward.

Motivated by this challenge, [CIK+09] and [BGL+12] initiated the study of *query-commit matching problem with patience constraints*, where the input is a graph $G = (V, E)$ with edge rewards and probabilities $(r_e)_{e\in E}$ and $(q_e)_{e\in E}$, respectively. The policy must sequentially *query* the edges in an order of its choosing, where a query to $e$ reveals whether it succeeds (i.e., can be included in the matching). In this problem, each edge $e$ is guaranteed to *succeed* independently with probability $q_e$. Moreover, the policy must be *committal*: if it queries $e$ and learns that it succeeds, then it must add $e$ to its current matching, in which case it gains the reward of $r_e$. Finally, each vertex $v \in V$ has an integer $\ell_v \geq 1$ *patience* or *time-out* constraint, indicating the maximum number of queries which can be performed on its incidence edges. The goal of the policy is maximize the sum of the rewards of the edges matched in expectation. This problem can be formulated as an exponential-sized Markov Decision Process (MDP), and so the optimal policy can be described by a dynamic program (DP). Due to the impracticality of computing such a DP, much of the literature has focused on attaining approximation algorithms of the optimal policy (see [CIK+09, Ada11, BGL+12, AGM15, BCN+18, BGMS24], and Subsection 1.2 for an extended overview of the literature).[1]

In this work, we introduce a generalization of the query-commit matching problem with patience constraints in which the policy has a set of *actions* $\mathcal{A}$ available when querying an edge. The input is again a graph $G = (V, E)$; however, for each edge $e$ and each action $a \in \mathcal{A}$, we are given an action-dependent success probability $q_e(a)$ and reward $r_e(a)$. The policy sequentially queries edges in an order of its choosing. When querying an edge $e$, it additionally selects an action $a \in \mathcal{A}$. Querying $e$ with action $a$ *succeeds* independently with probability $q_e(a)$. If $e$ succeeds via $a$, the edge $e$ is irrevocably added to the current matching and the policy gains reward $r_e(a)$. If it fails, the edge $e$ is not added to the matching, and it cannot be queried again in future steps. As before, each vertex $v \in V$ has a patience constraint $\ell_v$, limiting the number of incident queries, and the objective is to maximize the expected reward of the resulting matching. We refer to this problem as the *action-reward query-commit matching problem*.

A special case of our model is the *sequential pricing problem*, introduced by [PRSW24] and motivated by challenges arising in labor markets. The input is a bipartite graph, with vertices on the two sides representing jobs and workers, respectively. Each job is associated with a known reward. The goal of the *platform* is to have workers complete jobs by offering take-it-or-leave-it prices. This interaction is modeled stochastically: if the platform offers a price $\tau \geq 0$ to worker $v$ for completing job $u$, then $v$ accepts with a known probability $p_{u,v}(\tau)$, in which case $u$ and $v$ are matched and leave the system. As in the query-commit model, workers have known patience (time-out) constraints, specifying the maximum number of job offers they are willing to consider before exiting. The platform's objective is to maximize either its expected *revenue* or its expected

[1] We note that an alternative benchmark is the expected reward of the optimal offline matching; however, due to the patience constraints, it is not possible to design a policy within a constant factor approximation of it.

*social welfare*. By interpreting prices as actions in our framework (i.e., $\mathcal{A} = [0, \infty)$) and defining the edge rewards and success probabilities appropriately, our model captures both objectives.

We are also motivated by a natural extension of the query-commit matching problem where the rewards of the edges are described by arbitrary real-valued distributions, as opposed to weighted Bernoulli random variables. This is closely related to the *free-order prophet matching* problem [FTW$^+$21, BGMS24, PRSW24, MMG23, MM24], with the distinctions being that there are patience constraints, and that we benchmark against the optimal policy, as opposed to the prophet/hindsight optimum. Once again, our action-reward query-commit matching problem subsumes this problem, whether the objective is social welfare or revenue. We provide the details of both reductions in Section 6.

For the sequential pricing problem, [PRSW24] design a polynomial-time algorithm achieving an approximation ratio of 0.426 in the *one-sided patience* setting, where only workers have patience constraints. They also consider the *two-sided patience* setting, in which jobs have patience constraints as well; in this case, the approximation ratio drops to 0.395. Their algorithm and analysis rely on rounding a linear program (LP) inspired by the work of [BGL$^+$12]. However, due to a classic result of [KS81], it is not possible to design a policy with an expected reward greater than 0.544-fraction of the LP's value, leaving open the question of whether improved approximation ratios are achievable via alternative LP formulations and rounding techniques.

In this work, we answer this question affirmatively by establishing improved approximation guarantees for the action-reward query-commit matching problem on bipartite graphs. Since our model subsumes the sequential pricing problem of [PRSW24], our results improve the best known approximation ratios to $1 - 1/e$ and $\frac{1}{27}\left(19 - \frac{67}{e^3}\right) \approx 0.5801$ for the one-sided and two-sided patience settings, respectively.

Here, the approximation ratio of a policy is defined as the ratio between its expected reward and the expected reward of an optimal policy, i.e., the best policy without computational constraints, whose value on $G$ we denote by $\mathrm{OPT}(G)$. Our main result is formally stated as follows.

**Theorem 1.1** (Main Theorem)**.** *Given a bipartite graph $G = (U, V, E)$ with action set $\mathcal{A}$, let $\beta = 1 - 1/e$ if $\ell_u \in \{1, \infty\}$ for all $u \in U$, and $\beta = \frac{1}{27}(19 - \frac{67}{e^3})$, otherwise. For any arbitrarily small $\epsilon \in (0, 1)$, there exists a policy with expected reward of*

$$\beta \cdot (1 - \epsilon) OPT(G).$$

*Moreover, the policy can be computed in time $f(1/\epsilon) \cdot poly(|G|, |\mathcal{A}|)$ where $f$ is some function and poly is a polynomial in $|G|$ and $|\mathcal{A}|$.*

Most closely related to our work are [BM25] and [BGMS24], who gave polynomial-time $(1-1/e)$-approximations for the (standard) query-commit matching problem on bipartite graphs with one-sided patience, and *unit patience values* (e.g., $\ell_u = 1$ for all $u \in U$), respectively. Our $(1 - 1/e)$-approximation ratio can thus be seen as a generalization of both of these of results to our action-reward framework.

Moving to two-sided patience constraints, even in the (standard) query-commit matching problem, the previous best known approximation ratio follows from the aforementioned 0.426-approximation due to [PRSW24]. Thus, our $\frac{1}{27}(19-\frac{67}{e^3})$-approximation where $\frac{1}{27}(19-\frac{67}{e^3}) \approx 0.5801$, simultaneously improves on the approximation ratios of past works, while also applying to a more general setting.

### 1.1 Technical Overview

Our approach to proving Theorem 1.1 involves three high level steps. First, we design a new configuration LP to relax/upper bound the optimal policy's expected reward. Second, we design a

variant of an online contention resolution scheme for patience constraints, and show how to use it to round our LP into a policy. Finally, we compute an approximately optimal solution to our LP via a reduction to approximately solving the "star graph" case of our problem, where one partite set has a single vertex. Each of these steps is self-contained, and taken together quickly imply Theorem 1.1, as we explain in Section 5. Below, we provide an overview of the technical challenges faced in each step, and how these relate to the past approaches in the literature.

**Our Configuration LP (Section 2).** The majority of the past works in the literature use an LP which accounts solely for the marginal edge probabilities for which the optimal policy queries the various edges. While this LP was originally introduced by [BGL+12] for the (standard) query-commit matching problem, [PRSW24] used an analogous LP for their sequential pricing problem. It extends to our action-reward problem, and so we present it in this setting. For each $e \in E$ and $a \in \mathcal{A}$, the LP has a decision variable $z_e(a)$ which is the probability the benchmark queries $e$ via $a$. This leads to the following LP, where we recall that $q_e(a)$ denotes the probability that $e$ succeeds via $a$, $r_e(a)$ denotes the reward attained when this occurs, and $\partial(s)$ denotes the edges incident to vertex $s \in U \cup V$:

$$\text{maximize} \quad \sum_{e \in E, a \in \mathcal{A}} r_e(a) q_e(a) z_e(a) \tag{LP-M}$$

$$\text{subject to} \quad \sum_{e \in \partial(s), a \in \mathcal{A}} q_e(a) z_e(a) \leq 1 \qquad \forall s \in U \cup V \tag{1.1}$$

$$\sum_{e \in \partial(s), a \in \mathcal{A}} z_e(a) \leq \ell_s \qquad \forall s \in U \cup V \tag{1.2}$$

$$\sum_{a \in \mathcal{A}} z_e(a) \leq 1 \qquad \forall e \in E, \tag{1.3}$$

$$z_e(a) \geq 0 \qquad \forall e \in E, a \in \mathcal{A} \tag{1.4}$$

After computing an optimal solution $(z_e(a))_{e \in E, a \in \mathcal{A}}$ to LP-M, one still needs to efficiently round it into a policy. While earlier works [BGL+12, AGM15, BSSX20] used the GKSP algorithm of [GKPS06], the last number of works [BCN+18, BGMS24, PRSW24] – including the previous best state-of-the-art [PRSW24] – have all used independent sampling combined with a *uniform at random* (u.a.r) processing order of the edges. The simplest form of this algorithmic template is due to [BCN+18], where each time $e \in E$ is processed, if $u$ and $v$ are unmatched and have remaining patience, then $e$ is queried via action $a$ independently with probability (w.p.) $z_e(a)$ (this is well-defined, due to (1.3)). [BCN+18] showed that any edge $e$ and action $a$ is queried w.p. $0.310 z_e(a)$, which implies their approximation ratio of 0.310. The works of [BGMS24, PRSW24] further tuned this algorithm by introducing additional independent random bits $(B_e)_{e \in E}$, with the requirement that $e$ is queried only if $B_e = 1$. These bits help to increase the *minimum* probability that any edge $e$ is queried via any action $a$, and so they are able to get an improved guarantee of $\alpha z_e(a)$ for $\alpha = 0.382$ in [BGMS24] and $\alpha = 0.395$ in [PRSW24] [2] . In addition to the aforementioned 0.544 hardness result against comparisons to (LP-M), there is also a hardness result against algorithms which process the edges in a u.a.r order. The 1/2-impossibility result of [MMG23] implies that no such algorithm can attain an approximation ratio better than 1/2 against LP-M. Thus, there is limited room for attaining improved approximation ratios in this way.

[2] The policies of [BGMS24, PRSW24] additionally require the input to have minimum patience 2 for their approximation ratios to hold. For instance, the policy of [BGMS24] attains at most 1/3 for an input with vertices with patience 1.

While a tightening of LP-M was introduced by [CTT12, GKS19] that is more easily rounded, its definition is restricted to inputs with unlimited patience, and the (standard) query-commit matching problem (i.e., $|\mathcal{A}| = 1$). In this restricted setting, [GKS19] added additional LP constraints to LP-M, and derived a $(1-1/e)$-approximation. Unfortunately, it's unclear how to extend this LP to arbitrary patience constraints, let alone an action space with $|\mathcal{A}| \geq 2$. An alternative approach to handling the patience constraints was taken by [BM25, BGMS25], who presented a *configuration* LP for the case when $|\mathcal{A}| = 1$, where if $\boldsymbol{e} = (e_1, \ldots, e_k)$ is a sequence of distinct edges incident to $v \in V$ with $k \leq \ell_v$, then the LP has a decision variable $z_v(\boldsymbol{e})$ for the probability the benchmark queries the edges of $\boldsymbol{e}$ in order.

Our approach is to first generalize the configuration LP of [BM25, BGMS25] (formally stated in LP-C of Section 2) to our action-reward problem. We introduce a decision variable for each policy associated with each vertex $v \in V$. That is, if $\boldsymbol{e} = (e_1, \ldots, e_k)$ is a sequence of distinct edges incident to $v \in V$ with $k \leq \ell_v$, and $\boldsymbol{a} = (a_1, \ldots, a_k) \in \mathcal{A}^k$ is a sequence of actions, then $z_v(\boldsymbol{e}, \boldsymbol{a})$ is the probability the benchmark queries $e_i$ via $a_i$ for $i = 1, \ldots, k$ in order, outputting the first $e_i$ which succeeds via $a_i$ (if any). Setting $\text{val}(\boldsymbol{e}, \boldsymbol{a})$ as the expected reward for $v$ in this case, our LP's objective is

$$\sum_{v \in V} \sum_{(\boldsymbol{e}, \boldsymbol{a}) \in \mathcal{C}_v} \text{val}(\boldsymbol{e}, \boldsymbol{a}) \cdot z_v(\boldsymbol{e}, \boldsymbol{a}), \tag{1.5}$$

where the second sum is over $\mathcal{C}_v$, the set of all policies of length at most $\ell_v$ for $v$. Next, for each $v \in V$, we add the constraint

$$\sum_{(\boldsymbol{e}, \boldsymbol{a})} z_v(\boldsymbol{e}, \boldsymbol{a}) \leq 1, \tag{1.6}$$

which ensures each $v$ selects at most one policy $(\boldsymbol{e}, \boldsymbol{a})$. In order to motivate the remaining constraints, imagine drawing $(\boldsymbol{e}, \boldsymbol{a}) \in \mathcal{C}_v$ w.p. $z_v(\boldsymbol{e}, \boldsymbol{a})$, and then querying $e_i$ via $a_i$ in order, stopping if $e_i$ succeeds via $a_i$. In this case, for any $e \in E$ incident to $v$ and $a \in \mathcal{A}$, we can write out the probability that $e$ is queried via $a$ as follows:

$$\widetilde{z}_e(a) := \sum_{\substack{i \in [\ell_v], (\boldsymbol{e}, \boldsymbol{a}) \in \mathcal{C}_v : j < i \\ e_i = e, a_i = a}} \prod (1 - q_{e_j}(a_j)) \cdot z_v(\boldsymbol{e}, \boldsymbol{a}). \tag{1.7}$$

By applying (1.7) over all the vertices of $V$, we add two constraints for each vertex $u \in U$, which ensure that *in expectation*, $u$ is matched at most once, and has at most $\ell_u$ incident edges queried:

$$\sum_{e \in \partial(u)} \sum_{a \in \mathcal{A}} q_e(a) \widetilde{z}_e(a) \leq 1, \text{ and } \sum_{e \in \partial(u)} \sum_{a \in \mathcal{A}} \widetilde{z}_e(a) \leq \ell_u. \tag{1.8}$$

We will expand upon the exact use of (1.8) shortly, but we mention for now that they serve an analogous purpose as (1.1) and (1.2) do for LP-M, and the algorithms which round it.

In Theorem 2.1 of Section 2, we prove that LP-C relaxes our problem's benchmark. While LP-C generalizes the configuration LP's of [BM25, BGMS25], it has two key differences. First, it incorporates an arbitrary action space into its objective (1.5). Second, it allows for arbitrary patience constraints on the vertices of $U$. In contrast, the LP's of [BM25, BGMS25] are defined only for the standard query-commit problem (i.e., $|\mathcal{A}| = 1$), and assume that the vertices of $U$ have unlimited patience (i.e., $\ell_u = \infty$ for all $u \in U$).

**Rounding our LP (Section 3).** We next explain how to round a solution to LP-C into a policy. Assume for now that we have an optimal solution to LP-C, and first consider a natural *greedy* policy, which is well-defined due to (1.6):

1. Draw a u.a.r. arrival order of $V$.
2. For each arriving $v \in V$, draw a policy $(\boldsymbol{e}, \boldsymbol{a})$ for $v$ independently w.p. $z_v(\boldsymbol{e}, \boldsymbol{a})$
    - Then, when processing edge $e_i = (u_i, v_i)$, suppose $u_i$ has remaining patience and is currently unmatched. In this case, query $e_i$ via action $a_i$ and match $e_i$ if $e_i$ via $a_i$ succeeds.

This policy queries each edge $(u, v) \in E$ and $a \in \mathcal{A}$ w.p. $\beta_0 \cdot \widetilde{z}_{u,v}(a)$, where $\beta_0 = 1/2$ if $\ell_u \in \{1, \infty\}$, and $\beta_0 = (4-e)/e \approx 0.4715$ otherwise [3]. Thus, since we can equivalently write our LP's objective using (1.7) as

$$\sum_{v \in V} \sum_{(\boldsymbol{e},\boldsymbol{a}) \in \mathcal{C}_v} \text{val}(\boldsymbol{e}, \boldsymbol{a}) \cdot z_v(\boldsymbol{e}, \boldsymbol{a}) = \sum_{e \in E} \sum_{a \in \mathcal{A}} r_e(a) q_e(a) \widetilde{z}_e(a),$$

the greedy policy gets an approximation of $\beta_0$, and so it improves upon [PRSW24]. However, to get the guarantee of Theorem 1.1, we must further modify our policy. Our idea is to recognize that as the greedy policy executes, from the perspective of a fixed vertex $u$, the constraints (1.8) ensure that *in expectation*, at most $\ell_u$ edges of $u$ are queried, and at most one of the queried edges will succeed via some action. When $\ell_u \in \{1, \infty\}$, one of these constraints is subsumed, and we can execute a *(single-item) random order contention resolution (RCRS)* to decide which edge/action pair of $u$ to query (opposed to just being greedy). Due to a known result of [LS18, BGMS24], this quickly implies the $1 - 1/e$ guarantee of Theorem 1.1. On the other hand, if $2 \leq \ell_u < \infty$ for some $u \in U$, then we must design our own randomized rounding tool to determine which edge/action pair of $u$ to query. This is formalized in the *patience random-order contention resolution (P-RCRS) problem* of Section 3, and in Theorem 3.4, we state the rounding guarantee which we use to derive the $\frac{1}{27}(19 - \frac{67}{e^3}) \approx 0.5801$ result of Theorem 1.1.

We note that our P-RCRS problem is effectively the same rounding problem considered in [BGMS24, PRSW24], except that we only have to consider it for the special case of a fixed vertex $u \in U$ and its incident edges (i.e., a star graph). In contrast, [BGMS24, PRSW24] study this problem for an arbitrary graph whose edges are processed in random order. This explains why we improve on these past works. That being said, while our P-RCRS for this problem is based on the ideas in [BGMS24, PRSW24], there is a key technical step that was overlooked in [BGMS24] involving the worst-case input of their rounding algorithm [4]. We discuss this is more detail in Subsection 3.1, and mention that the updated arXiv version of [BGMS24] now cites the "exchange argument" of Lemma 3.10 from Subsection 3.2 to correct their algorithm's analysis.

**Solving our LP (Section 4).** Finally, in order to solve our LP efficiently, we first take its dual. While designing a separation oracle for an LP's dual is the standard way to solve an LP with exponentially many variables, and was in fact the approach taken in [BM25, BGMS25], this is not possible for our LP's dual. This is because the separation problem reduces to designing an optimal policy for the special case of our action-reward problem when $V$ contains a single vertex $v$. We call this the *star graph action-reward problem*, and note that it is NP-hard if $|\mathcal{A}| \geq 2$, even when

[3] To get the guarantee of $\beta_0$, technically the greedy policy must use simulated bits drawn from $(q_e(a))_{a \in \mathcal{A}, e \in E}$ to make "fake queries" when necessary, however we defer this discussion to Section 3.

[4] Since [PRSW24] builds upon the algorithm and analysis of [BGMS24], they were also not aware of this missing technical step.

$\ell_v = \infty$. This is because we can set the edge rewards/probabilities so as to encode the (single-item) free-order prophet problem, of which computing an optimal policy is known to be NP-hard for random variables with support at least 3 [ASZ20]. Fortunately, there exists an efficient polynomial time approximation scheme (EPTAS) for this problem [SS21]. In Subsection 4.1, we leverage the techniques of [SS21] to design an EPTAS for our star graph problem. That is, for any $\varepsilon > 0$, we design a policy with runtime $f(\varepsilon)\cdot\text{poly}(|U|, |\mathcal{A}|)$ whose expected reward is at least $(1-\varepsilon)$-fraction of the optimal policy's expected reward (here $f(\varepsilon)$ is some arbitrary function of $\varepsilon$, and $\text{poly}(|U|, |\mathcal{A}|)$ is a polynomial in $|U|$ and $|\mathcal{A}|$). As we explain in Section 4, this allows us to get a $(1-\varepsilon)$-approximate solution to LP-C, which together with the aforementioned rounding techniques of Section 3, implies Theorem 1.1. More, since our setting encodes the free-order prophet problem, as a corollary we get a generalization of the EPTAS of [SS21]. This generalization works for either the social welfare or revenue objective, and also handles the extension of the problem when the policy can only make a limited number of queries.

### 1.2 Further Related Work

For the query-commit matching problem with patience constraints and uniform edge rewards (i.e., unweighted graphs), [CIK$^+$09] showed that the greedy algorithm attains an approximation ratio of 1/4 and its analysis was later improved to 1/2 by [Ada11]. Afterwards, [BGL$^+$12] introduced the version of this problem with edge rewards, and attained an approximation ratio of 1/4 via a linear programming (LP) based policy. A sequence of papers later improved upon this result, [AGM15, BCN$^+$18, BGMS24], with the state of the art of 0.395 being due to [PRSW24].

We also mention some related results for the (standard) query-commit matching problem *without* patience values, in which case the benchmark taken is the optimal offline matching. For bipartite graphs, [GKS19] gave an algorithm with a guarantee of $1 - 1/e$. This was slightly improved by [DF23], further refined by [CHLT25] to 0.641, and most recently improved to 0.659 by [HSWZ25]. For general graphs, [FTW$^+$21] derived a guarantee of $8/15 \approx 0.533$, and this was slightly improved by [MM24]. We note that [CHLT25, FTW$^+$21, MM24] allow the edges to have rewards drawn independently from arbitrary distributions (i.e., the prophet matching problem), and [HSWZ25] applies when the edge probabilities are correlated (i.e., the oblivious bipartite matching problem).

### 1.3 Preliminaries

We formalize notation for our problem, recapping some concepts from the introduction.

Let $G = (U, V, E)$ be a bipartite graph. An edge $e = (u, v)$ is said to be *incident* to vertices $u$ and $v$, and $v$ is said to be a *neighbour* of $u$ (and vice versa). For any vertex $s \in U \cup V$, let $N(s) \subseteq V$ denote its neighbours, and let $\partial(s) \subseteq E$ denote the set of edges incident to $s$. A *matching* $M$ of $G$ is a subset of edges no two of which are incident to the same vertex, i.e. satisfying $|M \cap \partial(s)| \leq 1$ for all $s \in U \cup V$. We say that $s \in U \cup V$ is *matched* by $M$, provided $|M \cap \partial(s)| = 1$.

Suppose that associated with $G$ we have an action *action space* $\mathcal{A}$, *edge rewards* $\boldsymbol{r} = (r_e(a))_{e\in E, a\in\mathcal{A}}$ and *edge probabilities* $\boldsymbol{q} = (q_e(a))_{e\in E, a\in\mathcal{A}}$. More, assume that each $s \in U \cup V$ has *patience* $\ell_s \in \mathbb{N}_0$. A *policy* for the *action/reward matching problem* is given $G$, $\boldsymbol{r}, \boldsymbol{q}$ and $(\ell_s)_{s\in U\cup V}$ as its input, and its output is a matching $\mathcal{M}$ of $G$. The policy must be *committal*, which means its matching is constructed in the following way: In each step $t \geq 1$, it *adaptively* chooses an edge $e_t \in E$, as well as action $a_t \in \mathcal{A}$. Then, it *queries* $e_t$ via action $a_t \in \mathcal{A}$, at which point an independent Bernoulli $Q_{e_t}(a_t) \sim \text{Ber}(q_{e_t}(a_t))$ is drawn. If $Q_{e_t}(a_t) = 1$, then $(e_t, a_t)$ is said to *succeed*, and the policy must add $e_t$ to its current matching, in which case $e_t$ contributes a reward of $r_{e_t}(a_t)$. Otherwise, if $Q_{e_t}(a_t) = 0$, then $(e_t, a_t)$ is said to *fail*, and $e_t$ cannot be added to the matching. In addition, the

policy must follow the following additional rules:

1. For each $e \in E$, $e$ is queried by at most one action.
2. For each $s \in U \cup V$, there are at most $\ell_s$ edges in $\partial(s)$ which were queried via actions.

If at step $t$, there have been less than $\ell_s$ queries to $\partial(s)$, then we say that $s$ has *remaining patience* (at step $t$).

After at most $|E|$ steps, the matching $\mathcal{M}$ output by the policy is finalized, and the *reward* of the policy is $\sum_{e \in E} \sum_{a \in \mathcal{A}} r_e(a) \cdot Q_e(a) \cdot \mathbf{1}_{\{e \text{ queried via } a\}}$.

The goal of the problem is to design a policy whose *expected* reward is as large as possible, where the expectation is over $(Q_e(a))_{e \in E, a \in \mathcal{A}}$, as well as any randomized decisions made by the policy. We write as this

$$\sum_{e \in E} \sum_{a \in \mathcal{A}} r_e(a) \mathbb{P}[\{Q_e(a) = 1\} \cap \{e \text{ queried via } a\}] = \sum_{e \in E} \sum_{a \in \mathcal{A}} r_e(a) q_e(a) \mathbb{P}[e \text{ queried via } a], \tag{1.9}$$

where the equality follows since the policy must decide whether to query $e$ via $a$, prior to observing $Q_e(a)$. We denote the expected reward of the (computationally unconstrained) *optimal policy* on $G$ by $\mathrm{OPT}(G)$. By extending $r_e(a) := 0$ and $q_e(a) := 0$ for each $e \in U \times V \setminus E$ and $a \in \mathcal{A}$, we hereby assume without loss that $E = U \times V$.

Finally, we review some standard notation. For an arbitrary set $S$ and integer $k \geq 1$, let $S^k$ be the set of all vectors of length $k$ with entries in $S$, and $|\boldsymbol{s}| = k$ denote the length of $\boldsymbol{s} \in S^k$. We also set $[k] := \{1, \ldots, k\}$ for convenience.

## 2 Our Configuration LP

In this section, we introduce our configuration LP for $G = (U, V, E)$ with actions $\mathcal{A}$, edge rewards $\boldsymbol{r} = (r_e(a))_{e \in E, a \in \mathcal{A}}$ and edge probabilities $\boldsymbol{q} = (q_e(a))_{e \in E, a \in \mathcal{A}}$. Our LP introduces a decision variable for each policy associated with each vertex $v \in V$. Specifically, suppose that $\boldsymbol{e} = (e_1, \ldots, e_k)$ is permutation of a subset of $\partial(v)$ of size $k \leq \ell_v$, and $\boldsymbol{a} = (a_1, \ldots, a_k) \in \mathcal{A}^k$. Let $\mathcal{C}_v$ be denote the set of all such pairs $(\boldsymbol{e}, \boldsymbol{a})$. Observe that $(\boldsymbol{e}, \boldsymbol{a}) \in \mathcal{C}_v$ corresponds to the policy which queries the edges $e_1, \ldots, e_k$ incident to $v$ via actions $a_1, \ldots, a_k$, outputting the first edge/action pair $(e_i, a_i)$ which succeeds (if any). We define $\mathrm{val}_{\boldsymbol{r},\boldsymbol{q}}(\boldsymbol{e}, \boldsymbol{a})$ to be the expected reward of this policy, and observe that

$$\mathrm{val}_{\boldsymbol{r},\boldsymbol{q}}(\boldsymbol{e}, \boldsymbol{a}) = \sum_{i=1}^{k} r_{e_i}(a_i) \cdot q_{e_i}(a_i) \prod_{j<i} (1 - q_{e_j}(a_j)). \tag{2.1}$$

We will sometimes drop the dependence on $\boldsymbol{r}$ and $\boldsymbol{q}$ in (2.1) as the edge rewards and probabilities will be implicitly clear. With this notation, we now state our LP, where we introduce a decision variable $z_v(\boldsymbol{e}, \boldsymbol{a})$ for each $v \in V$ and $(\boldsymbol{e}, \boldsymbol{a}) \in \mathcal{C}_v$:

$$\text{maximize} \quad \sum_{v \in V} \sum_{(\boldsymbol{e},\boldsymbol{a}) \in \mathcal{C}_v} \text{val}(\boldsymbol{e},\boldsymbol{a}) \cdot z_v(\boldsymbol{e},\boldsymbol{a}) \tag{LP-C}$$

$$\text{subject to} \quad \sum_{v \in V} \sum_{\substack{i \in [\ell_v], (\boldsymbol{e},\boldsymbol{a}) \in \mathcal{C}_v: \\ e_i = (u,v)}} q_{(u,v)}(a_i) \cdot \prod_{j<i} (1 - q_{e_j}(a_j)) \cdot z_v(\boldsymbol{e},\boldsymbol{a}) \leq 1 \qquad \forall u \in U \tag{2.2}$$

$$\sum_{v \in V} \sum_{\substack{i \in [\ell_v], (\boldsymbol{e},\boldsymbol{a}) \in \mathcal{C}_v: \\ e_i = (u,v)}} \prod_{j<i} (1 - q_{e_j}(a_j)) \cdot z_v(\boldsymbol{e},\boldsymbol{a}) \leq \ell_u \qquad \forall u \in U \tag{2.3}$$

$$\sum_{(\boldsymbol{e},\boldsymbol{a}) \in \mathcal{C}_v} z_v(\boldsymbol{e},\boldsymbol{a}) \leq 1 \qquad \forall v \in V, \tag{2.4}$$

$$z_v(\boldsymbol{e},\boldsymbol{a}) \geq 0 \quad \forall v \in V, (\boldsymbol{e},\boldsymbol{a}) \in \mathcal{C}_v \tag{2.5}$$

Consider an optimal policy for $G$ (i.e, its expected reward is $\text{OPT}(G)$). If we interpret $z_v(\boldsymbol{e},\boldsymbol{a})$ as the probability this policy queries $\boldsymbol{e} \in \mathcal{C}_v$ via actions $\boldsymbol{a}$, the objective provides an upper bound on the policy's expected reward, and (2.4) ensures that each $v \in V$ chooses at most one policy for $\partial(v)$. Next, (2.2) corresponds to each $u \in U$ being matched at most once in expectation, and (2.3) corresponds to each $u \in U$ receiving at most $\ell_u$ queries in expectation.

**Theorem 2.1** (Proof in Appendix A)**.** *Let $LP(G)$ denote the optimal value of* (LP-C)*. Then, $OPT(G) \leq LP(G)$.*

While the above decision variable interpretation is roughly correct, to prove Theorem 2.1, we in fact consider a *relaxed* action-reward problem where the constraints on each $u \in U$ need only hold on average. That is, *in expectation*, (1) $u$ is matched at most once, and (2) $u$ has at most $\ell_u$ queries to its incident edges. We prove that for this relaxed problem, there exists an optimal *vertex-iterative* policy, which deals completely with all edges incident to a vertex before moving to the next. Policies of this form are easily seen to be upper bounded by LP-C, and so since the optimal reward of the relaxed problem is no smaller than our original problem (by definition), we establish Theorem 2.1.

# 3 Rounding a Solution to LP-C

Suppose that we are presented an arbitrary solution $(z_v(\boldsymbol{e},\boldsymbol{a}))_{v \in V, (\boldsymbol{e},\boldsymbol{a}) \in \mathcal{C}_v}$ to LP-C for $G$. Our goal is to use $(z_v(\boldsymbol{e},\boldsymbol{a}))_{v \in V, (\boldsymbol{e},\boldsymbol{a}) \in \mathcal{C}_v}$ to design a policy whose expected reward is at least

$$\beta \cdot \sum_{v \in V} \sum_{(\boldsymbol{e},\boldsymbol{a}) \in \mathcal{C}_v} \text{val}(\boldsymbol{e},\boldsymbol{a}) z_v(\boldsymbol{e},\boldsymbol{a}), \tag{3.1}$$

where $\beta := 1 - 1/e$ if $\ell_u \in \{1, \infty\}$ for all $u \in U$, and $\beta := 27(19 - \frac{67}{e^3})$ otherwise. Afterwards, in Section 4, we shall explain how to efficiently compute a $(1-\varepsilon)$-approximation of an optimal solution to LP-C for any $\varepsilon > 0$, which combined with Theorem 2.1 will imply Theorem 1.1.

In order to understand our approach to proving (3.1), it will be useful to again consider a *relaxed policy*, as was done in the proof of Theorem 2.1. A relaxed policy respects the patience constraints of the vertices of $V$, and outputs a *one-sided matching* $\mathcal{N}$ of $V$, i.e., $\mathcal{N} \subseteq E$ such that each $v \in V$ satisfies $|\partial(v) \cap \mathcal{N}| \leq 1$, yet it can query $\partial(u)$ more than $\ell_u$ times or have $|\mathcal{N} \cap \partial(u)| \geq 2$, for some $u \in U$. It still must follow the remaining rules of a (proper) policy (i.e., it is committal, and each

edge is queried via at most one action). Consider now the following relaxed policy which is defined using the solution $(z_v(\boldsymbol{e},\boldsymbol{a}))_{v\in V,(\boldsymbol{e},\boldsymbol{a})\in\mathcal{C}_v}$:

**Algorithm 1** Relaxed Rounding of LP-C

**Input:** a solution $(z_v(\boldsymbol{e},\boldsymbol{a}))_{v\in V,(\boldsymbol{e},\boldsymbol{a})\in\mathcal{C}_v}$ to LP-C for $G$.
**Output:** a one-sided matching $\mathcal{N}$ of $G$.

1: For each $e\in E$ and $a\in\mathcal{A}$, set $Z_e(a)=0$.
2: Independently draw a u.a.r. order $\pi$ on $V$.
3: **for** $v\in V$ in increasing order of $\pi$ **do**
4: Draw $(\boldsymbol{e},\boldsymbol{a})\in\mathcal{C}_v$ independently w.p. $z_v(\boldsymbol{e},\boldsymbol{a})$, where $\boldsymbol{e}=(e_1,\dots,e_k)$ and $\boldsymbol{a}=(a_1,\dots,a_k)$.
5: **for** $j=1,\dots,k$ **do**
6: Set $Z_{e_j}(a_j)=1$. ▷ $e_j$ with $a_j$ is suggested.
7: Query $e_j$ via $a_j$.
8: **if** $Q_{e_j}(a_j)=1$ **then**
9: $\mathcal{N}\leftarrow\mathcal{N}\cup\{e_j\}$.
10: Exit the internal for loop.
11: **return** $\mathcal{N}$.

Clearly, Algorithm 1 respects the patience constraints of $V$, and each $v\in V$ is assigned at most one edge, so $\mathcal{N}$ is a one-sided matching. Now, when line 6. is reached, we say that edge $e_j$ with action $a_j$ is *suggested*, and denote the indicator random variable for this event by $Z_{e_j}(a_j)$. Of course $Z_{e_j}(a_j)=1$ if and only if $e_j$ is queried via $a_j$, however this definition will be useful when we design a (proper) policy. Observe for now that for each $e=(u,v)\in E$ and $a\in\mathcal{A}$,

$$\widetilde{z}_e(a):=\mathbb{P}[Z_e(a)=1]=\mathbb{P}[e\text{ is queried via }a]=\sum_{\substack{i\in[\ell_v],(\boldsymbol{e},\boldsymbol{a})\in\mathcal{C}_v:\,j<i\\ e_i=e,a_i=a}}\prod(1-q_{e_j}(a_j))\cdot z_v(\boldsymbol{e},\boldsymbol{a}). \tag{3.2}$$

Using the definition of $\widetilde{z}_e(a)$ and (3.2), notice that we can write the expected reward of Algorithm 1 in two equivalent ways:

$$\sum_{e\in E}\sum_{a\in\mathcal{A}}r_e(a)q_e(a)\widetilde{z}_e(a)=\sum_{v\in V}\sum_{(\boldsymbol{e},\boldsymbol{a})\in\mathcal{C}_v}\mathrm{val}(\boldsymbol{e},\boldsymbol{a})z_v(\boldsymbol{e},\boldsymbol{a}). \tag{3.3}$$

On the other hand, due to the constraints of (LP-C), $(\widetilde{z}_e(a))_{e\in E,a\in\mathcal{A}}$ must also satisfy some linear constraints. The first says that the relaxed policy queries any edge via at most one action. The second and third say that for any $u\in U$, *in expectation*, the relaxed policy queries at most $\ell_u$ edges of $\partial(u)$, and assigns at most one edge to $u$.

**Lemma 3.1** (Proof in Appendix B.1)**.** *For any solution $(z_v(\boldsymbol{e},\boldsymbol{a}))_{v\in V,(\boldsymbol{e},\boldsymbol{a})\in\mathcal{C}_v}$ to LP-C for $G$, it holds that*

- *For each $(u,v)\in E$, $\sum_{a\in\mathcal{A}}\widetilde{z}_{u,v}(a)\le 1$.*
- *For each $u\in U$, $\sum_{v\in V}\sum_{a\in\mathcal{A}}\widetilde{z}_{u,v}(a)\le\ell_u$, and $\sum_{v\in V}\sum_{a\in\mathcal{A}}q_{u,v}(a)\widetilde{z}_{u,v}(a)\le 1$.*

We shall now modify the decisions of the relaxed policy so that it respects patience constraints of $U$, and outputs a (proper) matching of $G$, while following the decisions of the relaxed policy as best as possible. Roughly speaking, we will query an edge $e$ via $a$ *only if* $Z_e(a)=1$, but not all the time, due to the patience and matching constraints on $U$. To formalize this, let us first consider

an intermediate problem, where for a fixed vertex $u_0 \in U$, we respect the patience constraints on $\partial(u_0)$, and ensure $u_0$ is matched to at most one edge (for the vertices of $U \setminus \{u_0\}$, there are no hard restrictions).

Consider the execution of our relaxed policy from the perspective of $u_0$. In each step $t = 1, \ldots, |V|$, the vertex $v \in V$ with $\pi(v) = t$ is processed. At this point, suppose that $(u_0, v)$ with $a$ is suggested (i.e., $Z_{u_0,v}(a) = 1$) – otherwise, we will never match $u_0$ to $v$. In our intermediate problem, we can only query $(u_0, v)$ via $a$ if $u_0$ is currently unmatched and has remaining patience. If we make this decision *greedily*, i.e., query $(u_0, v)$ via $a$ whenever possible, one can prove that

$$\Pr[(u_0, v) \text{ is queried via } a \mid Z_{u_0,v} = 1] \geq \beta_0, \tag{3.4}$$

for each $v \in V$ and $a \in \mathcal{A}$, where $\beta_0 := 1/2$ if $\ell_{u_0} \in \{1, \infty\}$, and $\beta_0 := (4-e)/e \approx 0.4715$ otherwise. Thus, since $\mathbb{P}[Z_{u_0,v}(a) = 1] = \widetilde{z}_{u_0,v}(a)$ by (3.2), the expected reward of the match for $u_0$ is

$$\beta_0 \sum_{v \in V} \sum_{a \in \mathcal{A}} r_e(a) q_{u_0,v}(a) \widetilde{z}_{u_0,v}(a). \tag{3.5}$$

If we could achieve the same bound for the remaining vertices $u \in U \setminus \{u_0\}$, the equivalence in (3.3) implies that we'd get (3.1) for the weaker bound of $\beta_0$. To achieve our goal of $\beta$, we must beat the greedy strategy, and also extend this approach to work for all $u \in U$. Since the former problem is the challenging step, we focus on this. The latter problem will be handled via simulation in context of our formally defined policy (see Algorithm 2).

In order to get the bound of $\beta$ claimed in (3.1), we must more carefully decide when we should query $(u_0, v)$ when $Z_{u_0,v}(a) = 1$, assuming $u_0$ has remaining patience and is currently unmatched. We make this decision for $u_0$ using a *patience random-order contention resolution scheme* (P-RCRS). The objective of the P-RCRS is to ensure that $\Pr[(u_0, v) \text{ is queried via } a \mid Z_{u_0,v}(a) = 1] \geq \alpha$ for all $v \in V$ and $a \in \mathcal{A}$, where $\alpha \in [0, 1]$ is as large as possible. It operates from the aforementioned perspective of the fixed vertex $u_0$, where each time a vertex $v$ is processed with respect to $\pi$, it learns if $Z_{u_0,v}(a) = 1$ for some $a \in \mathcal{A}$, and if so, decides whether to query $(u_0, v)$ via $a$. We now formalize an abstract version of this problem for an arbitrary collection of elements and random variables. The elements $[n] = \{1, \ldots, n\}$ correspond to $\partial(u_0)$, and $(P_i(a))_{i \in [n], a \in \mathcal{A}}$ (respectively, $(X_i(a))_{i \in [n], a \in \mathcal{A}}$) correspond to $(Q_e(a))_{e \in \partial(u_0), a \in \mathcal{A}}$ (respectively, $(Z_e(a))_{e \in \partial(u_0), a \in \mathcal{A}}$). The technical assumptions on the random variables are motivated by Lemma 3.1, and the independent draws used in the execution of Algorithm 1.

**Definition 1.** An input to the *patience random-order contention resolution (P-RCRS) problem*, denoted $(\mathcal{A}, n, \ell, (p_i(a))_{i \in [n], a \in \mathcal{A}}, (x_i(a))_{i \in [n], a \in \mathcal{A}})$ is specified by an *action set* $\mathcal{A}$, an integer $n \geq 1$, *patience* $1 \leq \ell \leq \infty$, and probability vectors $(p_i(a))_{i \in [n], a \in \mathcal{A}}$ and $(x_i(a))_{i \in [n], a \in \mathcal{A}}$, where

$$\sum_{i \in [n]} \sum_{a \in \mathcal{A}} x_i(a) \leq \ell, \sum_{i \in [n]} \sum_{a \in \mathcal{A}} p_i(a) x_i(a) \leq 1 \text{ and } \sum_{a \in \mathcal{A}} x_i(a) \leq 1 \text{ for each } i \in [n]. \tag{3.6}$$

We assume that each element $i \in [n]$ independently draws $P_i(a) \sim \text{Ber}(p_i(a))$ for each $a \in \mathcal{A}$, as well as $(X_i(a))_{a \in \mathcal{A}}$ where $X_i(a) \sim \text{Ber}(x_i(a))$ and $\sum_{a \in \mathcal{A}} X_i(a) \leq 1$ (which is well-defined due to the last assumption in (3.6)). We refer to $(P_i(a))_{i \in [n], a \in \mathcal{A}}$ as the *states*, and $(X_i(a))_{i \in [n], a \in \mathcal{A}}$ as the *suggestions*.

A *patience random-order contention resolution* outputs at most one pair $(i, a)$ with $P_i(a) X_i(a) = 1$. Suppose that $\pi$ is an independently drawn u.a.r. permutation of $[n] := \{1, \ldots, n\}$. Initially, the instantiations of $(P_i(a), X_i(a))_{i \in [n], a \in \mathcal{A}}$ are unknown, and the elements of $[n]$ are presented in the increasing order specified by $\pi$. That is, in step $t \in [n]$, if $\pi(i) = t$ for $i \in [n]$, then it is revealed

$(X_i(a))_{a\in\mathcal{A}}$. If $\sum_{a\in\mathcal{A}} X_i(a) = 0$, then it *passes* on $i$ and moves to the next element. Otherwise, if $X_i(a) = 1$ for some $a \in \mathcal{A}$, then it either passes on $i$, or *queries* $i$ via $a$, thereby revealing $P_i(a)$. If $P_i(a) = 1$, then it must output $(i, a)$. Otherwise, it passes on $i$. It is also constrained in that it may query at most $\ell$ elements of $[n]$ (if $\ell = \infty$, then it may query all the elements of $[n]$). Given $\alpha \in [0, 1]$, we say that a P-RCRS is $\alpha$-*selectable* on the input, provided for all $i \in [n]$ and $a \in \mathcal{A}$,

$$\Pr[i \text{ is queried via } a \mid X_i(a) = 1] \geq \alpha. \tag{3.7}$$

Here (3.7) is taken over the randomness in $(P_i(a), X_i(a))_{i\in[n],a\in\mathcal{A}}$, $\pi$, as well as any randomized decisions made by the P-RCRS. When (3.7) holds, we also say that the P-RCRS has a *selection guarantee* of $\alpha$ on the input. If for a fixed $\alpha \in [0, 1]$, a P-RCRS satisfies (3.7) for *all* inputs with patience $\ell$, then we say it has a selection guarantee of $\alpha$ on inputs with patience $\ell$.

Given this definition, we are now ready to describe our policy. We assume that our policy is given access to a P-RCRS $\psi_u$ for each $u \in U$. In order to use the selection guarantee of each $\psi_u$ (opposed to just a single vertex $u_0$ as in the previously discussed intermediate problem), our policy defines its own suggestions $(\widetilde{Z}_e(a))_{e\in E,a\in\mathcal{A}}$. We ensure that these are distributed identically as $(Z_e(a))_{e\in E,a\in\mathcal{A}}$ from Algorithm 1, by using simulated bits $(\widetilde{Q}_e(a))_{e\in E,a\in\mathcal{A}}$ to make "fake" queries when necessary.

**Algorithm 2** Rounding LP-C via P-RCRS

**Input:** a solution $(z_v(\boldsymbol{e}, \boldsymbol{a}))_{v\in V,(\boldsymbol{e},\boldsymbol{a})\in\mathcal{C}_v}$ to LP-C for $G$.
**Output:** a matching $\mathcal{M}$ of $G$.

1: $\mathcal{M} \leftarrow \emptyset$.
2: For each $e \in E$ and $a \in \mathcal{A}$, set $\widetilde{Z}_e(a) = 0$, and draw $\widetilde{Q}_e(a) \sim \mathrm{Ber}(q_e(a))$ independently.
3: Independently draw a u.a.r. order $\pi$ on $V$.
4: **for** $v \in V$ in increasing order of $\pi$ **do**
5: Draw $(\boldsymbol{e}, \boldsymbol{a}) \in \mathcal{C}_v$ independently w.p. $z_v(\boldsymbol{e}, \boldsymbol{a})$, where $\boldsymbol{e} = (e_1, \ldots, e_k)$ and $\boldsymbol{a} = (a_1, \ldots, a_k)$.
6: **for** $j = 1, \ldots, k$ **do**
7: Set $e_j = (u_j, v)$ and $\widetilde{Z}_{u_j,v}(a_j) = 1$.
8: Execute $\psi_{u_j}$ on the P-RCRS input $(\mathcal{A}, \partial(u_j), (q_{u_j,v'}(a'))_{v'\in V,a'\in\mathcal{A}}, (\widetilde{z}_{u_j,v'}(a'))_{v'\in V,a'\in\mathcal{A}})$ using the arrival order on $\partial(u_j)$ induced by $\pi$, with random variables $(Q_{u_j,v'}(a))_{v'\in V,a'\in\mathcal{A}}$ as the states and $(\widetilde{Z}_{u_j,v'}(a'))_{v'\in V,a'\in\mathcal{A}}$ as the suggestions.
9: **if** $\psi_{u_j}$ queries $(u_j, v)$ via $a_j$ **then**
10: $\mathcal{M} \leftarrow \mathcal{M} \cup \{(u_j, v)\}$ provided $Q_{u_j,v}(a_j) = 1$.
11: Exit the internal for loop.
12: **else if** $\widetilde{Q}_{u_j,v}(a_j) = 1$ **then** ▷ simulate to maintain correct marginals
13: Exit the internal for loop.
14: **return** $\mathcal{M}$.

Now, for any $u \in U$, $\psi_u$ is passed the input $(\mathcal{A}, \partial(u), (q_{u,v}(a))_{v\in V,a\in\mathcal{A}}, (\widetilde{z}_{u,v}(a))_{v\in V,a\in\mathcal{A}})$, which satisfies the required inequalities in Definition 1, due to Lemma 3.1. However, we still must ensure that the random variables used in line 8. are drawn via $(q_{u,v}(a))_{v\in V,a\in\mathcal{A}}$ and $(\widetilde{z}_{u,v}(a))_{v\in V,a\in\mathcal{A}}$ in the required way as specified in Definition 1. We verify this in Lemma 3.2.

**Lemma 3.2** (Proof in Appendix B.2)**.** *For each $u \in U$, $(\mathcal{A}, \partial(u), (q_{u,v}(a))_{v\in V,a\in\mathcal{A}}, (\widetilde{z}_{u,v}(a))_{v\in V,a\in\mathcal{A}})$ satisfies the required inequalities of a P-RCRS input. Moreover, the random variables $(Q_{u,v}(a))_{a\in\mathcal{A}}$ and $(\widetilde{Z}_{u,v}(a))_{a\in\mathcal{A},v\in V}$ used in line 8. are drawn via $(q_{u,v}(a), \widetilde{z}_{u,v}(a))_{v\in V,a\in\mathcal{A}}$ in the way specified in Definition 1.*

Assuming Lemma 3.2, it is now easy to analyze the performance of Algorithm 2:

**Theorem 3.3.** *Fix $\alpha \geq 0$. Suppose that each P-RCRS $\psi_u$ used by Algorithm 2 is $\alpha$-selectable. Then, expected reward of Algorithm 2 is at least $\alpha \cdot \sum_{v \in V} \sum_{(\boldsymbol{e},\boldsymbol{a}) \in \mathcal{C}_v} val(\boldsymbol{e},\boldsymbol{a}) z_v(\boldsymbol{e},\boldsymbol{a})$.*

*Proof of Theorem 3.3.* Using Lemma 3.2, the expected reward of the policy is

$$\sum_{u \in U} \sum_{v \in V, a \in \mathcal{A}} r_{(u,v)}(a) \cdot \mathbb{P}[\widetilde{Z}_{u,v}(a) = 1] \cdot \mathbb{P}[Q_{u,v}(a) = 1 \cap \{(u,v) \text{ is queried via } a\} \mid \widetilde{Z}_{u,v}(a) = 1] \cdot$$
$$= \sum_{u \in U} \sum_{v \in V, a \in \mathcal{A}} r_{u,v}(a) \cdot q_{u,v}(a) \cdot \widetilde{z}_{u,v}(a) \cdot \mathbb{P}[(u,v) \text{ is queried via } a \mid \widetilde{Z}_{u,v}(a) = 1],$$

where the equality follows since the policy decides to query $(u,v)$ via $a$ prior to revealing $Q_{u,v}(a)$, and that $\mathbb{P}[\widetilde{Z}_{u,v}(a)] = \widetilde{z}_{u,v}(a)$ by Lemma 3.2. On the other hand, since each $u \in U$ executes an $\alpha$-selectable P-RCRS $\psi_u$ on a well-defined input due to Lemma 3.2, the definition of $\alpha$-selectable implies that $\mathbb{P}[(u,v) \text{ is queried via } a \mid \widetilde{Z}_{u,v}(a) = 1] \geq \alpha$ for each $a \in \mathcal{A}$ and $v \in V$. The theorem thus follows by (3.3). □

It remains to argue that there exists a $\beta$-selectable P-RCRS for the dependence on $\beta$ as claimed in Theorem 1.1.

**Theorem 3.4.** *For inputs with patience $2 \leq \ell < \infty$, there exists a P-RCRS with a selection guarantee of $\frac{1}{27}(19 - \frac{67}{e^3}) \approx 0.5801$. For inputs with patience $\ell \in \{1, \infty\}$, this selection guarantee improves to $1 - 1/e \approx 0.6321$.*

### 3.1 Proving Theorem 3.4

In order to design a P-RCRS with the guarantee claimed in Theorem 3.4, we first prove a reduction to the case when the input has a *trivial action set* (i.e., $|\mathcal{A}| = 1$). When $|\mathcal{A}| = 1$, we note that this is a special case of the definition in [AGM15] for single-item selection and uniform matroid (i.e., patience) query constraints. More, as we expand upon below, for $\ell \in \{1, \infty\}$ and $|\mathcal{A}| = 1$, the P-RCRS problem is equivalent to the *(single-item) random order contention resolution (RCRS) problem*, and so we will be able to use existing RCRS results to establish these cases of Theorem 3.4.

We now state our reduction from the case of an arbitrary action space to a trivial one. The main idea used to prove Lemma 3.5 is as follows: Suppose we are given an arbitrary input to the P-RCRS problem with $|\mathcal{A}| \geq 2$, whose states and suggestions we denote by $(P_i(a))_{a \in \mathcal{A}, i \in [n]}$ and $(X_i(a))_{a \in \mathcal{A}, i \in [n]}$, respectively. We construct an input to the P-RCRS problem with a single action, and couple it so that its states $(X_i)_{i \in [n]}$ and suggestions $(P_i)_{i \in [n]}$ satisfy $X_i = \sum_{a \in \mathcal{A}} X_i(a)$ and $P_i X_i = \sum_{a \in \mathcal{A}} P_i(a) X_i(a)$ for all $i \in [n]$. By executing an $\alpha$-selectable P-RCRS for a single action on $(P_i, X_i)_{i=1}^n$, we can then transfer its selection guarantee to apply to the input with $|\mathcal{A}| \geq 2$.

**Lemma 3.5** (Proof in Appendix B.3)**.** *Given an $\alpha$-selectable P-RCRS for inputs with patience $\ell$ and the trivial action space, there exists an $\alpha$-selectable P-RCRS for inputs with patience $\ell$ and an arbitrary action space.*

Due to Lemma 3.5, we now focus exclusively on the P-RCRS problem with the trivial action set for the remainder of the section. As previously suggested, we drop the dependence on $a \in \mathcal{A}$, and write our input as $\mathcal{I} = (n, \ell, (p_i)_{i=1}^n, (x_i)_{i=1}^n)$, where now $\sum_{i=1}^n x_i \leq \ell$ and $\sum_{i=1}^n p_i x_i \leq 1$. Then, each $i \in [n]$ draws state $P_i \sim \text{Ber}(p_i)$ and suggestion $X_i \sim \text{Ber}(x_i)$ independently. More, since there is now a single action, we simply say that the P-RCRS queries the elements of $[n]$ (opposed

to specifying which action this is done by). Our goal is then to design a P-RCRS such that for all $i \in [n]$,

$$\Pr[i \text{ is queried} \mid X_i = 1] \geq \beta, \tag{3.8}$$

where $\beta = 1 - 1/e$ if $\ell \in \{2, \infty\}$, and $\beta = \frac{1}{27}(19 - \frac{67}{e^3})$ otherwise. Now, if $\ell = \infty$, then the P-RCRS can query all the elements of $[n]$, and so one can use an $\alpha$-selectable RCRS on the input $(n, (p_i x_i)_{i=1}^n)$ to get an $\alpha$-selectable P-RCRS. Similarly, if $\ell = 1$, then $\sum_{i=1}^n x_i \leq 1$, and so one can use an $\alpha$-selectable RCRS on $(n, (x_i)_{i=1}^n)$ to get an $\alpha$-selectable P-RCRS. Thus, the cases of $\ell \in \{1, \infty\}$ in Theorem 3.4 are immediate due to the $(1-1/e)$-selectable RCRS of [LS18, BGMS24].

We thus design an P-RCRS for $2 \leq \ell < \infty$, in which case $\beta := \frac{1}{27}(19 - \frac{67}{e^3})$. It will be convenient to define $x(S) := \sum_{i \in S} x_i$ for each $S \subseteq [n]$, and $\text{Pois}(\mu)$ to be a Poisson random variable of mean $\mu \geq 0$. We define the *attenuation function* $b : [0,1] \to [0,1]$, where for $s \in [0,1]$,

$$b(s) = \frac{\beta}{\int_0^1 e^{-y(1-s)} \mathbb{P}[\text{Pois}(2y) < 3] dy} \tag{3.9}$$

It is easy to check that $\beta = \int_0^1 e^{-y} \mathbb{P}[\text{Pois}(2y) < 3] dy$, and so $b(0) = 1$. More, $b$ is decreasing function of $s$ on $[0,1]$.

Our P-RCRS draws an independent random bit $B_i \sim \text{Ber}(b(p_i x_i))$ for each $i \in [n]$. Then, when $i$ arrives, it queries $i$ if $B_i X_i = 1$, less than $\ell$ queries were previously made, and no element was previously output. We provide a formal description below:

**Algorithm 3** P-RCRS

**Input:** $\mathcal{I} = (n, \ell, (x_i)_{i=1}^n, (p_i)_{i=1}^n)$
**Output:** at most one queried element $i \in [n]$ with $P_i X_i = 1$.

Set $L \leftarrow 0$.
**for** arriving $i \in [n]$ **do**
    Draw $B_i \sim \text{Ber}(b(p_i x_i))$ independently.
    **if** $B_i X_i = 1$, $L < \ell$, and no element was previously output **then**
        Query $i$ and set $L \leftarrow L + 1$.
        **if** $P_i = 1$ **then**
            **return** $i$.

In order to establish (3.8), we reformulate the arrival model, and define each $i \in [n]$ to have an independent and u.a.r. *arrival time* $Y_i \in [0,1]$. We assume that the elements of $[n]$ have distinct arrival times, and that they are presented to Algorithm 3 in increasing order of these values.

For notational convenience, let us hereby focus on proving (3.8) for $i = 1$. We say that 1 is *safe* when executing Algorithm 3 on $\mathcal{I}$, provided that before 1 arrives:

1. At most $\ell - 1$ queries were made.
2. No element of $[n] \setminus \{1\}$ was output.

Observe then that

$$\mathbb{P}[1 \text{ is queried} \mid Y_1 = y, X_1 = 1] = b(p_1 x_1) \mathbb{P}[1 \text{ is safe} \mid Y_1 = y]. \tag{3.10}$$

We first argue with respect to lower bounding (3.10), we can assume our input $\mathcal{I}$ is of a specific form. That is, $p_j \in \{0,1\}$ for all $j \in [n] \setminus \{1\}$, $p_1 = 1$, and $\sum_{i \in [n]} x_i = \ell$. Note that the first part of this simplification is proven in Lemma 2 of [BGMS24], and our argument proceeds almost identically. We construct a new input $\widetilde{\mathcal{I}}$ which satisfies the required conditions, and argue that for all $y \in [0,1]$, (3.10) can only decrease when Algorithm 3 is executed on $\widetilde{\mathcal{I}}$ instead of $\mathcal{I}$.

**Lemma 3.6** (Proof in Appendix B.4)**.** *When lower bounding* (3.10) *over all* $y \in [0,1]$*, we may assume that the input* $\mathcal{I}$ *satisfies* $p_1 = 1$*,* $p_j \in \{0,1\}$ *for all* $j \in [n] \setminus \{1\}$*, and* $\sum_{i\in[n]} x_i = \ell$*.*

Let us now define $N_q := \{j \in [n] \setminus \{1\} : p_j = q\}$ for $q \in \{0,1\}$. Due to Lemma 3.6, we hereby assume that $N_0 \cup N_1$ partitions $[n] \setminus \{1\}$, and that $p_1 = 1$. Let us now define $L_0$ as the number of queries Algorithm 3 makes to $N_0$ before 1 arrives. Observe then that

$$\begin{aligned}\mathbb{P}[1 \text{ is safe} \mid Y_1 = y] &= \mathbb{P}[L_0 \le \ell - 1 \mid Y_1 = y] \cdot \mathbb{P}[\cap_{i\in N_1}\{B_i X_i \mathbf{1}_{Y_i<y} = 0\} \mid Y_1 = y] \\ &= \mathbb{P}[L_0 \le \ell - 1 \mid Y_1 = y] \prod_{j\in N_1} (1 - yb(x_j)x_j),\end{aligned}$$

where the second line uses the independence of $(B_i, X_i, \mathbf{1}_{Y_i<y})_{i\in N_1}$, conditional on $Y_1 = y$. Thus, combined with (3.10),

$$\mathbb{P}[1 \text{ is queried} \mid X_1 = 1] = b(x_1) \int_0^1 \mathbb{P}[L_0 \le \ell - 1 \mid Y_1 = y] \prod_{j\in N_1} (1 - yb(x_j)x_j)dy. \tag{3.11}$$

We next argue that with respect to lower bounding (3.11), we may assume that $\max_{j\in N_1} x_j \to 0$. I.e., the worst-case configuration of $(x_j)_{j\in N_1}$ occurs in the *Poisson regime*. Since $b(0) = 1$, this allows us to replace $\prod_{j\in N_1}(1 - yb(x_j)x_j)$ by $e^{-yx(N_1)}$ in (3.11). Unlike Lemma 3.6, this worst-case configuration only applies if we integrate over $y \in [0,1]$.

**Lemma 3.7** (Proof in Appendix B.5)**.** *The probability* $\mathbb{P}[1$ *is queried* $\mid X_1 = 1]$ *is lower bounded by*

$$b(x_1) \int_0^1 e^{-yx(N_1)} \mathbb{P}[L_0 \le \ell - 1 \mid Y_1 = y]dy. \tag{3.12}$$

We prove Lemma 3.7 using a *splitting argument*. Specifically, given $i \in N_1$, we construct a new input $\widetilde{\mathcal{I}}$ with an additional element $i'$ added to $N_1$. Then, we set the new fractional value of $i$ and $i'$ to each be $x_i/2$, and argue that (3.11) is lower bounded by the analogous term for $\widetilde{\mathcal{I}}$. By iteratively applying this argument, and using that $b(0) = 1$, we conclude that the worst-case configuration of $(x_i)_{i\in N_1}$ occurs in the Poisson regime. For this argument to work, we need the following analytic properties of the attenuation function $b$.

**Proposition 3.8** (Proof in Appendix B.6)**.** *The attenuation function* $b : [0,1] \to [0,1]$ *satisfies the following properties:*

1. $b(0) = 1$ *and* $b$ *is non-increasing on* $[0,1]$*.*
2. $\int_0^1 (1 - yzb(z))dy \ge \int_0^1 (1 - yzb(z/2)/2)^2 dy$ *for each* $z \in [0,1]$*.*
3. *For each* $z \in [0,1]$*, there exists* $y_z \in [0,1]$*, such that* $(1 - yzb(z)) - (1 - yzb(z/2)/2)^2 \ge 0$ *for all* $y \in [0, y_z]$*, and* $(1 - yzb(z)) - (1 - yzb(z/2)/2)^2 \le 0$ *for all* $y \in (y_z, 1]$*.*

We next consider the term $\mathbb{P}[L_0 \le \ell - 1 \mid Y_1 = y]$ of (3.12). Observe that conditional on $Y_1 = y$, $L_0$ is a sum of $|N_0|$ independent Bernoulli random variables. Moreover, $\mathbb{E}[L_0 \mid Y_i = y] = \sum_{j\in N_0} b(p_j x_j)x_j y = \sum_{j\in N_0} x_j y = yx(N_1)$, as $b(0) = 1$, and $p_j = 0$ for all $j \in N_0$. A partial characterization of the worst-case value of $\mathbb{P}[L_0 \le \ell - 1 \mid Y_1 = y]$ is implied by Lemma 7 of [BCN+18], which we state below for convenience:

**Lemma 3.9** (Lemma 7 in [BCN+18]). *Fix an integer $t \geq 1$. Suppose $Z_1, \ldots, Z_k$ are independent Bernouli random variables, and $\mu := \sum_{i=1}^{k} \mathbb{E}[Z_i] < t - 1$. Then,*

$$\mathbb{P}\left[\sum_{i=1}^{k} Z_i < t\right] \geq \mathbb{P}[\text{Pois}(\mu) < t] = \sum_{i=0}^{t-1} \frac{e^{-y\mu}\mu^i}{i!}.$$

Applied to our setting, if $\mathbb{E}[L_0 \mid Y_i = y] = yx(N_1) < \ell - 1$, then

$$\mathbb{P}[L_0 \leq \ell - 1 \mid Y_1 = y] \geq \mathbb{P}[\text{Pois}(x(N_0)y) < \ell] = \sum_{k=0}^{\ell-1} \frac{e^{-yx(N_0)}(yx(N_0))^k}{k!}. \tag{3.13}$$

On the other hand, if we recall the feasibility constraints on $\mathcal{I}$ after the simplification of Lemma 3.6,

$$x(N_1) + x_1 \leq 1, \text{ and } x(N_1) = \ell - x(N_0) - x_1. \tag{3.14}$$

Thus, if $x(N_1) + x_1 < 1$, then $yx(N_0) \geq \ell - 1$ for $y \geq (\ell - 1)/x(N_0)$, and so (3.13) does not apply on the interval $[(\ell-1)/x(N_0), 1]$ [5]. Moreover, this is not simply a limitation of the characterization of [BCN+18]. For $y$ close to 1, the worst-case of behaviour of $\mathbb{P}[L_0 \leq \ell - 1 \mid Y_1 = y]$ changes to an input when $|N_0| = \ell$, and $x_i = x(N_0)/\ell$ for all $i \in N_0$. For instance, in the extreme case of $y = 1$ and $x(N_0) = \ell$, one can make $\mathbb{P}[L_0 \leq \ell - 1 \mid Y_1 = y] = 0$. Determining the worst-case configuration of $(x_i)_{i \in N_0}$ as a function of $y$ seems difficult, as it is closely related proving a longstanding conjecture of Samuels from probability theory (see [Sam66, Sam69] and [Fei06, Pau17]). More, it is also challenging to follow the approach of Lemma 3.7, and determine the worst-case behaviour of $\int_0^1 \mathbb{P}[L_0 \leq \ell - 1 \mid Y_1 = y]dy$. For instance, if $x(N_0) = \ell$, then the worst-case configuration $(x_i)_{i \in N_0}$ has $|N_0| = \ell$ and $x_i = 1$ for all $i \in N_0$, which is the opposite of the Poisson regime. Instead, we side-step this problem, where our approach differs depending on if $\ell = 2$, $3 \leq \ell < 120$ or $\ell \geq 120$.

For $\ell = 2$, we apply an *exchange argument* (formally stated in Lemma 3.10 of Subsection 3.2). Specifically, we construct an input $\widetilde{\mathcal{I}}$ which adds a new element $i_1$ to $N_1$ of fractional value at most $1 - x(N_1) - x_1$, and reduces the fractional value of an element of $i_0$ of $N_0$ by the same amount, thus ensuring that $\widetilde{\mathcal{I}}$ is feasible. We then argue that (3.12) for $\mathcal{I}$ is lower bounded by the analogously defined term for $\widetilde{\mathcal{I}}$. This implies that for $\ell = 2$, when lower bounding (3.12), we may assume that $x(N_1) = 1 - x_1$.

For $3 \leq \ell < 120$, while it is conceivable that an exchange argument could also apply, the comparisons become more complicated, and so we abandon this approach. Instead, we apply (3.13) to $\mathbb{P}[L_0 \leq \ell - 1 \mid Y_1 = y]$ when $y \in [0, (\ell-1)/x(N_0))$, and the trivial lower bound of 0 when $y \in [(\ell - 1)/x(N_0)), 1]$. We then integrate over $y \in [0, 1]$ to get a lower bound on (3.12) of $b(x_1) \cdot F_\ell(x_1, x(N_1))$. The function $F_\ell$ has a closed-form expression, and we prove that $F_\ell(x_1, x(N_1)) \geq F_\ell(x_1, 1 - x_1)$ (see Proposition 3.11 of Subsection 3.3). Thus, we are once again able to assume that $x(N_1) = 1 - x_1$ when lower bounding (3.12).

In either case (i.e., $\ell = 2$ or $3 \leq \ell < 120$), we may assume that $x(N_1) = 1 - x_1$, and so $x(N_0) = \ell - 1$. We can thus apply (3.13) to (3.12) for all $y \in [0, 1)$. Combining Lemma 3.7 with

[5] This detail was overlooked in [BGMS24, PRSW24], as they apply Lemma 7 of [BCN+18] for all $y \in [0, 1]$ (even when their analogously defined quantity has $x(N_1) + x_1 < 1$). This is not an issue in [BCN+18], as they do not use attenuation in their policy, and so they're able to easily argue that they can assume $x(N_1) + x_1 = 1$.

the definition of $b$, this implies that

$$\begin{aligned}\mathbb{P}[1 \text{ is queried} \mid X_1 = 1] &\geq b(x_1) \int_0^1 \mathbb{P}[\text{Pois}(y(\ell-1)) < \ell] e^{-y(1-x_1)} dy \\ &= \frac{\beta}{\int_0^1 \mathbb{P}[\text{Pois}(2y) < 3] e^{-y(1-x_1)} dy} \int_0^1 \mathbb{P}[\text{Pois}(y(\ell-1)) \leq \ell] e^{-y(1-x_1)} dy. \end{aligned} \tag{3.15}$$

For each $3 \leq \ell < 120$, (3.15) is a function of $0 \leq x_1 \leq 1$ with a closed form expression, and we verify that it is attains its minimum when $x_1 = 0$ and $\ell = 3$, in which case it is exactly $\beta$. This proves the guarantee of Theorem 3.4 assuming $2 \leq \ell < 120$.

Finally, for the case when $\ell \geq 120$, since $\ell$ is sufficiently large, it suffices to apply Bennett's inequality to $\mathbb{P}[L_0 \leq \ell - 1 \mid Y_1 = y]$ for each $y \in [0,1]$. In Subsection 3.4, we show that this leads to a bound of

$$\mathbb{P}[1 \text{ is queried} \mid X_1 = 1] \geq b(x_1) \int_0^1 \left(1 - e^{-120(y + \log\left(\frac{1}{y}\right) - 1)}\right) e^{-y(1-x_1)} dy. \tag{3.16}$$

Now, (3.16) is a function of $0 \leq x_1 \leq 1$ with a closed-form expression. We verify that it attains its minimum occurs when $x_1 = 1$, in which case we get

$$\frac{\beta}{\int_0^1 \mathbb{P}[\text{Pois}(2y) < 3] dy} \cdot \int_0^1 \left(1 - e^{-120(y + \log\left(\frac{1}{y}\right) - 1)}\right) dy \geq 0.5803,$$

which is strictly large than $\beta$. This proves the guarantee of Theorem 3.4 for $\ell \geq 120$.

We now provide the details of the $\ell = 2$, $3 \leq \ell < 120$ and $\ell \geq 120$ regimes, in Subsections 3.2, 3.3, and 3.4, respectively.

### 3.2 Details of Patience $\ell = 2$

Given the input $\mathcal{I} = (n, 2, (x_i)_{i=}^n, (p_i)_{i=1}^n)$, we construct a new input $\widetilde{\mathcal{I}} = (n+1, 2, (\widetilde{x}_i)_{i=1}^{n+1}, (\widetilde{p}_i)_{i=1}^{n+1})$, with an additional element $i_1 := n + 1$, and which modifies the fractional value of an arbitrary $i_0 \in N_0$. Specifically,

1. $\widetilde{p}_{i_1} = 1$ and $\widetilde{x}_{i_1} := \min\{1 - x(N_1) - x_1, x_{i_0}\}$,
2. $\widetilde{p}_{i_0} = 0$ and $\widetilde{x}_{i_0} = x_{i_0} - \min\{1 - x(N_1) - x_1, x_{i_0}\}$,
3. $(\widetilde{x}_i, \widetilde{p}_i) = (x_i, p_i)$ for all $i \in [n] \setminus \{i_0\}$.

Clearly, $\widetilde{\mathcal{I}}$ is a feasible input, which satisfies the conditions of Lemma 3.6. I.e., $\widetilde{p}_j \in \{0, 1\}$ for all $j \in [n+1]$, $\widetilde{p_1} = 1$ and $\sum_{j \in [n+1]} \widetilde{x}_j = \ell$. Define $\widetilde{N}_0 = N_0 \setminus \{i_0\}$ and $\widetilde{N}_1 = N_1 \cup \{i_1\}$, and $\widetilde{L}_0$ as the number of queries Algorithm 3 makes to $\widetilde{N}_0 \cup \{i_0\} = N_0$ when executing on $\widetilde{\mathcal{I}}$. Then, using $X_1$ and $Y_1$ for 1 during the execution of Algorithm 3 on $\widetilde{\mathcal{I}}$, Lemma 3.7 implies that

$$\mathbb{P}[1 \text{ from } \widetilde{\mathcal{I}} \text{ is queried} \mid X_1 = 1] \geq b(x_1) \int_0^1 \mathbb{P}[\widetilde{L}_0 \leq 1 \mid Y_1 = y] e^{-y(x(N_1) + \widetilde{x}_{i_1})} dy,$$

where we've used that $\sum_{i \in \widetilde{N}_1} \widetilde{x}_i = x(N_1) + \widetilde{x}_{i_1}$, and $\widetilde{x}_1 = x_1$. We are now ready to formalize the statement of our exchange argument:

**Lemma 3.10.** *The following inequality holds:*

$$\int_0^1 \mathbb{P}[L_0 \le 1 \mid Y_1 = y] e^{-yx(N_1)} dy \ge \int_0^1 \mathbb{P}[\widetilde{L}_0 \le 1 \mid Y_1 = y] e^{-y(x(N_1)+\widetilde{x}_{i_1})} dy.$$

*Proof of Lemma 3.10.* Our goal is to prove that

$$\int_0^1 e^{-yx(N_1)} \left( \mathbb{P}[L_0 \le 1 \mid Y_1 = y] - e^{-y\widetilde{x}_{i_1}} \mathbb{P}[\widetilde{L}_0 \le 1 \mid Y_1 = y] dy \right) \ge 0. \tag{3.17}$$

First observe that $\mathbb{P}[L_0 \le 1 \mid Y_1 = y]$ is equal to

$$\begin{aligned}(1 - yx_{i_0}) \prod_{j \in \widetilde{N}_0} (1 - yx_j) + yx_{i_0} \prod_{j \in \widetilde{N}_0} (1 - yx_j) + \sum_{i \in \widetilde{N}_0} yx_i (1 - yx_{i_0}) \prod_{j \in \widetilde{N}_0 \setminus \{i\}} (1 - yx_j). \\ = \prod_{j \in \widetilde{N}_0} (1 - yx_j) + \sum_{i \in \widetilde{N}_0} yx_i (1 - yx_{i_0}) \prod_{j \in \widetilde{N}_0 \setminus \{i\}} (1 - yx_j). \end{aligned} \tag{3.18}$$

and similarly, $\mathbb{P}[\widetilde{L}_0 \le 1 \mid Y_1 = y]$ is equal to

$$\prod_{j \in \widetilde{N}_0} (1 - yx_j) + \sum_{i \in \widetilde{N}_0} yx_i (1 - y\widetilde{x}_{i_0}) \prod_{j \in \widetilde{N}_0 \setminus \{i\}} (1 - yx_j). \tag{3.19}$$

Thus, applying (3.18) and (3.19), we can rewrite the integrand of (3.17) as

$$\begin{aligned} & e^{-yx(N_1)} \left( (1 - e^{-y\widetilde{x}_{i_1}}) \prod_{j \in \widetilde{N}_0} (1 - yx_j) + \sum_{i \in \widetilde{N}_0} (1 - yx_{i_0} - e^{-y\widetilde{x}_{i_1}} (1 - y\widetilde{x}_{i_0})) yx_i \prod_{j \in \widetilde{N}_0 \setminus \{i\}} (1 - yx_j) \right). \\ = & e^{-yx(N_1)} \left( (1 - e^{-y\widetilde{x}_{i_1}}) \prod_{j \in \widetilde{N}_0} (1 - yx_j) - \sum_{i \in \widetilde{N}_0} (e^{-y\widetilde{x}_{i_1}} (1 - y\widetilde{x}_{i_0}) - (1 - yx_{i_0})) yx_i \prod_{j \in \widetilde{N}_0 \setminus \{i\}} (1 - yx_j) \right). \end{aligned} \tag{3.20}$$

Now, since $\widetilde{x}_{i_1} \ge 0$, and $\widetilde{x}_{i_1} + \widetilde{x}_{i_0} = x_{i_0}$, we know that for all $y \in [0,1]$,

$$1 - e^{-y\widetilde{x}_{i_1}} \ge 0 \text{ and } e^{-y\widetilde{x}_{i_1}} (1 - y\widetilde{x}_{i_0}) - (1 - yx_{i_0}) \ge (1 - y\widetilde{x}_{i_1})(1 - y\widetilde{x}_{i_0}) - (1 - yx_{i_0}) \ge 0, \tag{3.21}$$

where we have applied the elementary inequality $1 - z \le e^{-z}$. Thus, we focus on lower and upper bounding $\prod_{j \in \widetilde{N}_0} (1 - yx_j)$ and $\sum_{i \in \widetilde{N}_0} yx_i \prod_{j \in \widetilde{N}_0 \setminus \{i\}} (1 - yx_j)$, respectively. Starting with the first term, observe that for any fixed $y \in [0,1]$ and $k := |\widetilde{N}_0|$, the function $\psi : [0,1]^k \to [0,1]$, $\psi(z_1, \ldots, z_k) := \prod_{j=1}^k (1 - yz_j)$ is *Schur-concave* (see [PT92]). This implies that its minimum over all $(z_1, \ldots, z_k) \in [0,1]^k$ with $\sum_{i=1}^k z_i = x(\widetilde{N}_0)$ is attained when $z_i = 1$ for $i = 1, \ldots, \lfloor x(\widetilde{N}_0) \rfloor$, and $z_{\lfloor x(\widetilde{N}_0) \rfloor + 1} = x(\widetilde{N}_0) - \lfloor x(\widetilde{N}_0) \rfloor$. We thus get that

$$\prod_{j \in \widetilde{N}_0} (1 - yx_j) \ge (1 - y)^{\lfloor x(\widetilde{N}_0) \rfloor} (1 - y(x(\widetilde{N}_0) - \lfloor x(\widetilde{N}_0) \rfloor)) \tag{3.22}$$

On the other hand, using the elementary inequality $1 - z \le e^{-z}$, and that $x_i \le 1$ for all $i \in \widetilde{N}_0$,

$$\sum_{i \in \widetilde{N}_0} yx_i \prod_{j \in \widetilde{N}_0 \setminus \{i\}} (1 - yx_j) \le \sum_{i \in \widetilde{N}_0} yx_i e^{-y(x(\widetilde{N}_0) - x_i)} \le yx(\widetilde{N}_0) e^{-y(x(\widetilde{N}_0) - 1)}. \tag{3.23}$$

Thus, applying (3.21), (3.22) and (3.23) to (3.20), we can lower bound the integrand of (3.17) by

$$\begin{aligned} &e^{-yx(N_1)}(1-e^{-y\widetilde{x}_{i_1}})(1-y)^{\lfloor x(\widetilde{N}_0)\rfloor}(1-y(x(\widetilde{N}_0)-\lfloor x(\widetilde{N}_0)\rfloor)) \\ &-(e^{-y\widetilde{x}_{i_1}}(1-y\widetilde{x}_{i_0})-(1-yx_{i_0}))yx(\widetilde{N}_0)e^{-y(x(\widetilde{N}_0)+x(N_1)-1)}. \end{aligned} \tag{3.24}$$

Observe next that $x(\widetilde{N}_0) = x(N_0) - x_{i_0} = 2 - x(N_1) - x_1 - x_{i_0}$, since $x(N_0) = 2 - x(N_1) - x_1$ by (3.14). We simplify (3.24), depending on whether $x_{i_0} > 1 - x(N_1) - x_1$, or $x_{i_0} \leq 1 - x(N_1) - x_1$.

If $x_{i_0} > 1 - x(N_1) - x_1$, then $x(\widetilde{N}_0) = 2 - x(N_1) - x_1 - x_{i_0} \leq 1$, so $\lfloor x(\widetilde{N}_0)\rfloor = 0$. Moreover, $\widetilde{x}_{i_1} = 1 - x(N_1) - x_1$, and $\widetilde{x}_{i_0} = x_{i_0} - (1 - x(N_1) - x_1)$. Thus, (3.24) simplifies to

$$\begin{aligned} &(e^{-yx(N_1)} - e^{-y(1-x_1)})(1 - y(2 - x(N_1) - x_1 - x_{i_0})) \\ &-(e^{-y(1-x(N_1)-x_1)}(1 - y(x_{i_0} - (1 - x(N_1) - x_1)) - (1 - yx_{i_0}))y(2 - x(N_1) - x_1 - x_{i_0})e^{-y(1-x_1-x_{i_0})}. \end{aligned} \tag{3.25}$$

Otherwise, if $x_{i_0} \leq 1 - x(N_1) - x_1$, then $x(\widetilde{N}_0) \geq 1$, so $\lfloor x(\widetilde{N}_0)\rfloor = 1$. More, $\widetilde{x}_{i_0} = 0$ and $\widetilde{x}_{i_1} = x_{i_0}$. Thus, (3.24) simplifies to

$$\begin{aligned} &e^{-yx(N_1)}(1-e^{-yx_{i_0}})(1-y)(1-y(1-x(N_1)-x_1-x_{i_0})) \\ &-(e^{-yx_{i_0}}-(1-yx_{i_0}))y(2-x(N_1)-x_1-x_{i_0})e^{-y(1-x_1-x_{i_0})}. \end{aligned} \tag{3.26}$$

If we integrate (3.25) (respectively, (3.26)) over $y \in [0, 1]$, then we are left with a function of non-negative variables $x(N_1), x_{i_0}$ and $x_1$, where $x(N_1) + x_1 \leq 1$ and $x_{i_0} > 1 - x(N_1) - x_1$ (respectively, $x_{i_0} \leq 1 - x(N_1) - x_1$). We numerically verify that restricted to the appropriate domain of values, either function is non-negative, and so (3.17) holds. The lemma is thus proven. □

### 3.3 Details of Patience $3 \leq \ell < 120$

For $\ell \geq 3$, let us define $y_c := (\ell - 1)/x(N_0)$ for convenience, and recall that $x(N_0) = \ell - x_1 - x(N_1)$. Then, for $y \in [0, y_c)$, we can apply (3.13) to ensure that

$$\mathbb{P}[L_0 \leq \ell - 1 \mid Y_1 = y] \geq \sum_{k=0}^{\ell-1} \frac{e^{-y(\ell-x_1-x(N_1))}y^k(\ell - x(N_1) - x_1)^k}{k!} \tag{3.27}$$

Thus, after simplification, (3.12) is lower bounded by

$$b(x_1)\int_0^{y_c}\sum_{k=0}^{\ell-1}\frac{e^{-y(\ell-x_1)}y^k(\ell-x(N_1)-x_1)^k}{k!}dy \tag{3.28}$$

If $\Gamma(s, z) := \int_z^\infty \zeta^{s-1}e^{-\zeta}d\zeta$ for $s > 0$ and $z \in \mathbb{R}$ is the *upper incomplete gamma function* (here $\Gamma(s, 0)$ is the usual *gamma function*), then the integral in (3.28) has the closed-form expression:

$$F_\ell(x_1, x(N_1)) := \frac{\Gamma(\ell) - \Gamma(\ell, \ell-1)e^{\frac{-(l-1)x(N_1)}{\ell-x(N_1)-x_1}} - \left(\frac{\ell-x_1-x(N_1)}{\ell-x(N_1)}\right)^\ell\left(\Gamma(\ell) - \Gamma(\ell, \ell-1+\frac{(\ell-1)x(N_1)}{\ell-x(N_1)-x_1})\right)}{x(N_1)\Gamma(\ell)}.$$

If $\ell = 3$, then this simplifies to

$$F_3(x_1, x(N_1)) = \frac{2 - 10e^{\frac{-2(3-x(N_1))}{3-x(N_1)-x_1}} - \left(\frac{3-x_1-x(N_1)}{3-x(N_1)}\right)^3\left(2 - \Gamma(3, 2+\frac{2x(N_1)}{3-x(N_1)-x_1})\right)}{2x(N_1)},$$

which we numerically verified is minimized at $x(N_1) = 1 - x_1$ for any $0 \le x_1 \le 1$. Otherwise, if $4 \le \ell < 120$, $D_2 F_\ell(x_1, x(N_1))$ is non-positive for all $0 \le x_1 \le 1$ and $0 \le x(N_1) \le 1$. Thus, we conclude that no matter the choice of $x_1$, $F_\ell$ is minimized when $x(N_1) = 1 - x_1$ as desired.

**Proposition 3.11.** *For each $3 \le \ell < 120$, $x_1, x(N_1) \in [0,1]$ with $0 \le x_1 + x(N_1) \le 1$, it holds that $F_\ell(x_1, x(N_1)) \ge F_\ell(x_1, 1 - x_1)$.*

### 3.4 Details of Patience $\ell \ge 120$

For this regime of $\ell$, we make use of Bennett's inequality:

**Theorem 3.12** (Bennett's Inequality – [Ben62])**.** *Suppose that $Z_1, \dots, Z_k$ are random variables with finite variance, and for which $|Z_i - \mathbb{E}[Z_i]| \le c$ for all $i \in [k]$. If $Z = \sum_{i=1}^k Z_i$, $\mu = \sum_{i=1}^k \mathbb{E}[Z_i]$ and $\sigma^2 = \sum_{i=1}^k \mathbb{E}[(Z_i - \mathbb{E}[Z_i])^2]$, then for each $t \ge 0$,*

$$\mathbb{P}[Z \ge \mu + t] \le \exp\left(-\frac{\sigma^2}{c^2} h\left(\frac{bt}{\sigma^2}\right)\right) \tag{3.29}$$

*where $h(s) = (1+s)\log(1+s) - s$ for $s \ge 0$. In particular, if $c = 1$ and $\sigma^2 \le \mu$, then*

$$\mathbb{P}[Z \ge \mu + t] \le \exp\left(-(\mu + t)\log\left(\frac{\mu + t}{t}\right) + t\right). \tag{3.30}$$

Recall that conditional on $Y_1 = y$ for $y \in [0,1]$, $L_0$ is a sum of $|N_0|$ independent Bernoulli random variables with means $(yx_i)_{i \in N_0}$

We can therefore bound its (conditional) variance by $\sum_{i \in N_0} x_i y x_i \le yx(N_0) = \mathbb{E}[L_0 \mid Y_1 = y]$. Thus, we can apply Bennett's inequality (stated as (3.30) in Theorem 3.12) with parameters $c = 1$, $\mu = yx(N_0)$ and $t = \ell - yx(N_1)$, to conclude that,

$$\mathbb{P}[L_0 < \ell \mid Y_1 = y] \ge 1 - e^{-(\mu+t)\log\left(\frac{\mu+t}{\mu}\right)+t} = 1 - e^{-\ell \log\left(\frac{\ell}{yx(N_0)}\right) - yx(N_0) + \ell}. \tag{3.31}$$

Recall that $x(N_0) \le \ell$ by (3.14) due to Lemma 3.6. For any fixed $y \in [0,1]$ and $\ell$, the right-hand side of (3.31) is minimized when $x(N_0) = \ell$, so we may conclude that

$$\mathbb{P}[L_0 < \ell \mid Y_1 = y] \ge 1 - e^{-\ell(y + \log\left(\frac{1}{y}\right) - 1)} \tag{3.32}$$

Finally, for any fixed $y \in [0,1]$, the right-hand side of (3.32) is a decreasing function of $\ell$. Thus, since $\ell \ge 120$,

$$\mathbb{P}[L_0 < \ell \mid Y_1 = y] \ge 1 - e^{-120(y + \log\left(\frac{1}{y}\right) - 1)},$$

as claimed in (3.16).

## 4 Approximately Solving LP-C

Given $\varepsilon > 0$, we must efficiently compute a solution to LP-C with value at least $(1 - \varepsilon)\mathrm{LP}(G)$. In order to do so, we first take the dual of LP-C.

$$\text{minimize} \qquad \sum_{u\in U}\alpha_u + \sum_{u\in U}\ell_u\gamma_u + \sum_{v\in V}\beta_v \tag{D-LP-c}$$

$$\text{s.t.} \qquad \beta_v \geq \mathrm{val}_{\boldsymbol{r},\boldsymbol{q}}(\boldsymbol{e},\boldsymbol{a}) - \sum_{i=1}^{|\boldsymbol{e}|}(q_{e_i}(a_i)\alpha_{u_i} + \gamma_{u_i})\cdot\prod_{j<i}(1-q_{e_j}(a_j)) \qquad \forall v\in V, (\boldsymbol{e},\boldsymbol{a})\in\mathcal{C}_v :$$

$$e_i = (u_i, v) \quad \forall i \in [|\boldsymbol{e}|]$$

$$\alpha_u, \gamma_u \geq 0 \qquad \forall u \in U$$

$$\beta_v \geq 0 \qquad \forall v \in V$$

Let us suppose that we are presented $((\alpha_u)_{u\in U}, (\gamma_u)_{u\in U}, (\beta_v)_{v\in V})$, an arbitrary (potentially infeasible) dual solution to D-LP-c . Observe that for $(\boldsymbol{e},\boldsymbol{a})\in\mathcal{C}_v$ with $e_i = (u_i, v)$ for $i = 1,\ldots,|\boldsymbol{e}|$,

$$\begin{aligned}&\mathrm{val}_{\boldsymbol{r},\boldsymbol{q}}(\boldsymbol{e},\boldsymbol{a}) - \sum_{i=1}^{|\boldsymbol{e}|}(q_{e_i}(a_i)\cdot\alpha_{u_i} + \gamma_{u_i})\prod_{j<i}(1-q_{e_j}(a_j))\\ &= \sum_{i=1}^{|\boldsymbol{e}|}(r_{e_i}(a_i) - \alpha_{u_i} - \frac{\gamma_{u_i}}{q_{e_i}(a_i)})q_{e_i}(a_i)\prod_{j<i}(1-q_{e_j}(a_j)).\end{aligned} \tag{4.1}$$

This motivates defining new edge rewards, where for each $(u,v)\in E$ and $a\in\mathcal{A}$,

$$\hat{r}_{u,v}(a) := \max\{r_{u,v}(a) - \alpha_u - \gamma_u/q_{u,v}(a), 0\} \tag{4.2}$$

For each $v\in V$, consider the *star graph* input to the action-reward problem on $\partial(v)$ with patience $\ell_v$, action space $\mathcal{A}$, edge rewards $\hat{\boldsymbol{r}}^{(v)} = (\hat{r}_{u,v}(a))_{u\in U, a\in\mathcal{A}}$ and edge probabilities $\boldsymbol{q}^{(v)} = (q_{u,v}(a))_{u\in U,a\in\mathcal{A}}$. On such a restricted type of input, the goal of this problem is to solve the following maximization problem:

$$\begin{aligned}&\text{maximize} \quad \mathrm{val}_{\hat{\boldsymbol{r}}^{(v)},\boldsymbol{q}^{(v)}}(\boldsymbol{e},\boldsymbol{a})\\ &\text{subject to} \quad (\boldsymbol{e},\boldsymbol{a})\in\mathcal{C}_v\end{aligned} \tag{4.3}$$

Notice that if we could efficiently solve (4.3) for each $v\in V$, then due to (4.1), this would immediately yield a *separation oracle* for D-LP-c.

It is well known that one can use ellipsoid method with a separation oracle for D-LP-c to then solve LP-C efficiently (see Chapter 4 of [KV12]). Unfortunately, unless the action space satisfies $|\mathcal{A}| = 1$ (see [PGR19, BGMS25, BM25]), we cannot expect to solve (4.3) *exactly*, due to the NP-hardness result of [ASZ20].

Fortunately, there exists a generalization of this approach for covering and packing LP's provided in Theorem 5.1 of [Jan03]. In particular, to get a multiplicative approximation of LP-C, it suffices to design a *(strong) approximate separation oracle* for D-LP-c. We argue in Appendix B that if we can compute a $(1-\varepsilon)$-approximation of (4.3) for each $v\in V$, then we can design an approximate separation oracle, and thus get a $(1-\varepsilon)$-approximate solution to LP-C.

**Proposition 4.1** (Proof in Appendix C.1)**.** *Suppose that for every $v\in V$, we can compute a $(1-\varepsilon)$-approximate solution to the maximization problem in* (4.3) *in time $O(f(1/\varepsilon)\cdot poly(|U|,|\mathcal{A}|))$, where $f$ is an arbitrary function of $\varepsilon$. Then, a solution to LP-C with value at least $(1-\varepsilon)LP(G)$ can be computed in time $O(f(1/\varepsilon)\cdot poly(|G|,|\mathcal{A}|))$.*

In Subsection 4.1, we approximately solve the star graph problem for an arbitrary input in the proposed runtime of Proposition 4.1. This suffices to get the required approximate solution to LP-C in the runtime claimed in Theorem 1.1.

## 4.1 Approximately Solving the Star Graph Problem

In this section, we provide an approximation scheme for the star graph problem. We do this by providing a reduction to the Santa Claus problem (see Definition 2). The techniques in this section are mostly adapted from Section 3.5 of [SS25], however we include all of the proofs for completeness.

We first restate the definition of the star graph problem in a slightly more convenient notation and terminology than that which was used previously. An instance of this problem consists of:

- A star graph with $n$ edges, where each edge $e \in [n]$ has a *reward* $r_e(a)$ for each action $a \in \mathcal{A}$ independently with probability $q_e(a)$ and otherwise 0.
- A *budget/patience* constraint $\ell \leq n$ limiting the number of edges that can be *queried*.

We define a *policy* to be a pair $(\boldsymbol{e}, \boldsymbol{a})$ where:

- $\boldsymbol{e} = (e_1, \ldots, e_\ell)$ specifies the edges queried, and the order this is done.
- $\boldsymbol{a} = (a_1, \ldots, a_\ell)$ specifies the actions set for each edge of $\boldsymbol{e}$.

The *expected reward* of policy $(\boldsymbol{e}, \boldsymbol{a})$ is then

$$\mathrm{val}_{\boldsymbol{r},\boldsymbol{q}}(\boldsymbol{e}, \boldsymbol{a}) = \sum_{i=1}^{\ell} r_{e_i}(a_{e_i}) \cdot q_{e_i}(a_{e_i}) \cdot \prod_{j=1}^{i-1} \left(1 - q_{e_j}(a_{e_j})\right). \tag{4.4}$$

The objective is to find a policy $(\boldsymbol{e}^*, \boldsymbol{a}^*)$ that maximizes (4.4). We denote OPT as the value of an optimal policy, and approximately solve this problem:

**Theorem 4.2.** *For the star graph problem, there exists an efficient polynomial-time approximation scheme (EPTAS): For any $\varepsilon > 0$, there exists an algorithm that runs in time $O\left(\left(\frac{1}{\varepsilon^2}\right)^{O(1/\varepsilon)} \cdot \mathrm{poly}(n, |\mathcal{A}|)\right)$ and outputs a policy $(\boldsymbol{e}, \boldsymbol{a})$ with $\mathrm{val}_{\boldsymbol{r},\boldsymbol{q}}(\boldsymbol{e}, \boldsymbol{a}) \geq (1 - 7\varepsilon) \cdot \mathrm{OPT}$.*

**Outline.** We prove Theorem 4.2 via three main steps:

1. *Parameter Estimation and Bucketing:* Given an estimate $\mathcal{E}$ of the optimal value, partition the optimal policy into a constant number of buckets based on reward contribution patterns.
2. *Santa Claus Reduction:* Define load contributions that capture how much each edge/action pair can contribute to each bucket's value requirement. Formulate an integer program that assigns edge/action pairs to buckets while satisfying capacity and load constraints.
3. *Policy Reconstruction:* Convert the bucket assignment back to a policy and optimize actions via dynamic programming.

### 4.1.1 Step 1: Parameter Estimation and Bucketing

We first assume that $1/\epsilon$ is an integer, and that we have an estimate $\mathcal{E}$ for which $(1 - \epsilon) \cdot \mathrm{OPT} \leq \mathcal{E} \leq \mathrm{OPT}$. While the first assumption is clearly without loss, in order to see the second, first write

LP-C with $|U| = 1$. In our current notation, we get the following LP:

$$\begin{aligned}
\text{maximize} \quad & \sum_{e\in[n],a\in\mathcal{A}} r_e(a)q_e(a)z_e(a) && && \text{(LP-C-s)}\\
\text{subject to} \quad & \sum_{e\in[n],a\in\mathcal{A}} q_e(a)z_e(a) \le 1 && && (4.5)\\
& \sum_{e\in[n],a\in\mathcal{A}} z_e(a) \le \ell && && (4.6)\\
& \sum_{a\in\mathcal{A}} z_e(a) \le 1 && \forall e \in [n] && (4.7)\\
& z_e(a) \ge 0 && \forall e \in [n], a \in \mathcal{A} && (4.8)
\end{aligned}$$

Suppose that LPOPT denotes the optimal value of LP-C-s. Clearly, OPT $\le$ LPOPT. On the other hand, after solving for an optimal solution $(z_e(a))_{e\in[n],a\in\mathcal{A}}$ to LP-C-s, Theorem 3.3 implies that we can use an $\alpha$-selectable P-RCRS (see Definition 1) to get policy with expected reward at least $\alpha \sum_{e\in[n],a\in\mathcal{A}} r_e(a)q_e(a)z_e(a)$. Thus, since Theorem 3.4 guarantees the existence of a $\frac{1}{27}(19 - \frac{67}{e^3})$-selectable P-RCRS where $\frac{1}{27}(19 - \frac{67}{e^3}) > 1/2$, we know that

$$\text{LPOPT}/2 \le \text{OPT} \le \text{LPOPT}. \tag{4.9}$$

We can therefore try all powers of $(1+\epsilon)$ in the interval $[\text{LPOPT}/2, \text{LPOPT}]$ and at least one of them must be an $(1+\epsilon)$-approximation of OPT. This justifies our second assumption involving $\mathcal{E}$.

Now, for any subset of ordered edges $\boldsymbol{e} = (e_1, \ldots, e_\ell)$, let $R(\boldsymbol{e})$ denote the expected reward obtained by an algorithm that uses the *optimal* strategy on $\boldsymbol{e}$. This quantity can be computed in polynomial time via dynamic programming by considering the optimal actions for each position. Specifically, for the optimal policy $(\boldsymbol{e}^*, \boldsymbol{a}^*)$, we can recursively define the *future value function* as follows. Let $R_i^*$ be the maximum expected value achievable from positions $i, i+1, \ldots, \ell$ when following the optimal policy:

$$R_{\ell+1}^* = 0 \tag{4.10}$$

$$R_i^* = \max_{a\in\mathcal{A}} \left\{ r_{e_i^*}(a) \cdot q_{e_i^*}(a) + R_{i+1}^* \cdot (1 - q_{e_i^*}(a)) \right\} \tag{4.11}$$

where the maximum is taken over all possible actions $a$ for position $i$. The optimal expected value is $\text{OPT} = R_1^*$.

Intuitively, $R_i^*$ represents the expected value we can achieve starting from position $i$ in the optimal ordering, assuming we use optimal actions for all remaining positions. The sequence $R_1^* \ge R_2^* \ge \ldots \ge R_\ell^* \ge R_{\ell+1}^* = 0$ is non-increasing, as each position can only decrease the future value by potentially accepting a suboptimal offer.

Let position $i \in [\ell]$ be a *jump* if $R_i^* - R_{i+1}^* \ge \epsilon \cdot R_1^*$. Since $R_1^* = \text{OPT} \ge \mathcal{E}$ and each jump contributes at least $\epsilon \cdot \mathcal{E}$ to the total decrease, we can have at most $\mathcal{E}/(\epsilon \cdot \mathcal{E}) = 1/\epsilon$ jumps.

**Bucket Construction.** Let $\tilde{t}_1 < \tilde{t}_2 < \ldots < \tilde{t}_K$ be the positions of reward jumps, where $K \le 1/\epsilon$. We partition the positions $[\ell]$ into $M = 2K + 1$ buckets:

- Stable Bucket $B_1$: Positions $\{\tilde{t}_K + 1, \tilde{t}_K + 2, \ldots, \ell\}$ (after the last jump)
- Jump Bucket $B_2$: Position $\{\tilde{t}_K\}$ (the last jump position)

- Stable Bucket $B_3$: Positions $\{\tilde{t}_{K-1}+1, \tilde{t}_{K-1}+2, \ldots, \tilde{t}_K - 1\}$
- Jump Bucket $B_4$: Position $\{\tilde{t}_{K-1}\}$
- And so on...

For each bucket $B_i$ we define $\text{BaseVal}(B_i)$ as the minimum future value $R^*_j$ that corresponds to any position appearing immediately after this bucket in the optimal policy $(\boldsymbol{e}^*, \boldsymbol{a}^*)$. Formally, $\text{BaseVal}(B_i) = R^*_{\max\{j : j \text{ is a position in bucket } B_i\}+1}$ where we set $R^*_{\ell+1} = 0$ by convention. This means that $\text{BaseVal}(B_1) = 0$, $\text{BaseVal}(B_2) = R^*_{\tilde{t}_K+1}$, $\text{BaseVal}(B_3) = R^*_{\tilde{t}_K}$, and so forth.

Similarly, we define $\text{DeltaVal}(B_i)$ as the total value contribution of bucket $B_i$, which equals the difference between the future value before and after processing this bucket: $\text{DeltaVal}(B_i) = \text{BaseVal}(B_{i+1}) - \text{BaseVal}(B_i)$.

**Parameter Guessing.** Since we do not know the optimal policy, we guess approximations to these parameters:

For each bucket $B_i$, we guess:

- $\text{BaseGuess}(B_i) \in \{0, \epsilon^2\mathcal{E}, 2\epsilon^2\mathcal{E}, \ldots, \mathcal{E}\}$
- $\text{DeltaGuess}(B_i) \in \{0, \epsilon^2\mathcal{E}, 2\epsilon^2\mathcal{E}, \ldots, \mathcal{E}\}$

such that:

$$\text{BaseVal}(B_i) - \epsilon^2\mathcal{E} \le \text{BaseGuess}(B_i) \le \text{BaseVal}(B_i)$$
$$\text{DeltaVal}(B_i) - \epsilon^2\mathcal{E} \le \text{DeltaGuess}(B_i) \le \text{DeltaVal}(B_i)$$

The total number of parameter combinations to try is $(1/\epsilon^2)^{O(1/\epsilon)}$.

#### 4.1.2 Step 2: Santa Claus Reduction

At this step, we reduce our problem to an allocation problem, where we allocate the variables to buckets such that each bucket has a load of at least DeltaGuess, with a small error. To solve this, we use another problem called the Santa Claus problem, which was studied in [SS25]. Here is the definition:

**Definition 2.** An instance of the *Santa Claus* problem consists of:

- A set of $m$ unrelated machines, where each machine $i \in [m]$ has:
  - An upper bound $\ell_i \in \mathbb{N}$ on the number of jobs it can process
  - A lower bound $L_i \in \mathbb{R}_+$ on its required load
- A collection of $n$ jobs, where assigning job $j \in [n]$ to machine $i \in [m]$ incurs a scalar load $c_{ij} \in \mathbb{R}_+$
- A global budget $\ell \in \mathbb{N}$ on the total number of assigned jobs

A feasible assignment is a function $\phi : [n] \to [m] \cup \{\emptyset\}$ such that:

1. $|\{j \in [n] : \phi(j) = i\}| \le \ell_i$ for all $i \in [m]$ (capacity constraints)
2. $\sum_{j:\phi(j)=i} c_{ij} \ge L_i$ for all $i \in [m]$ (load constraints)

3. $\sum_{j\in[n]} \mathbf{1}_{\phi(j)\neq\emptyset} \leq \ell$ (budget constraint)

Assume without loss of generality that $L_i = 1$ (by scaling) and $c_{ij} \in [0,1]$ for all $i, j$.

$$\text{(SC-IP)} \tag{4.12}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in [m], j \in [n] \tag{SC-1}$$

$$\sum_{i\in[m]} x_{ij} \leq 1 \qquad \forall j \in [n] \tag{SC-2}$$

$$\sum_{j\in[n]} x_{ij} \leq \ell_i \qquad \forall i \in [m] \tag{SC-3}$$

$$\sum_{j\in[n]} c_{ij} x_{ij} \geq 1 \qquad \forall i \in [m] \tag{SC-4}$$

$$\sum_{i\in[m]} \sum_{j\in[n]} x_{ij} \leq \ell \tag{SC-5}$$

**Remark 4.3.** The EPTAS of [SS25] was designed for the more general *Multi-Dimensional Santa Claus* problem, where loads and lower bounds are $D$-dimensional vectors. In our application, we only require the single-dimensional case ($D = 1$), for which the same techniques apply and simplify considerably. It also does not include the global budget constraint (SC-5). However, their techniques generalize to account for this, allowing us to ensure that this constraint can be enforced with probability 1. This is because [SS25] introduce a "strengthened" LP which partitions the jobs $j$ based on their load $(c_{i,j})_{i\in[m]}$ – these are called the load classes. They then guess the number of jobs from any load class assigned to any machine. By restricting to guesses in which these load class guesses satisfy the global budget constraint, their algorithm outputs a solution for which (SC-5) holds.

**Theorem 4.4** (Adapted from Theorem 2.1 of [SS25])**.** *For any $\epsilon > 0$, when (SC-IP) is feasible, there exists an algorithm running in time $O(2^{m\log(1/\epsilon)} \cdot poly(n))$ that computes $\{0,1\}$-valued random variables $(X_{i,j})_{i\in[m],j\in[n]}$ such that with probability at least $1/2$:*

1. *Constraints (SC-1), (SC-2), (SC-3), and (SC-5) are satisfied exactly.*
2. *Constraint (SC-4) is $\epsilon$-violated: $\sum_{j\in[n]} c_{ij} X_{ij} \geq 1 - \epsilon$ for all $i \in [m]$.*

To reduce to the Santa Claus problem with Global Budget, we need:

- $n$ *jobs* that correspond to the edges. We also have $2K + 1$ *machines* corresponding to the buckets.
- For variable $j \in [n]$ assigned to bucket $B_i \in [2K+1]$ with baseline future value $\text{BaseGuess}(B_i)$, we set the *assignment load* to be

$$c_{ij} = \max_a \left\{ [(r_j(a) - \text{BaseGuess}(B_i)) \cdot q_j(a)]^+ \right\}$$

- Jump machines can be assigned at most one job, and in total we can assign $\ell$ jobs to all the machines.
- Each machine $i \in [2K + 1]$ has a lower bound of $\text{DeltaGuess}(B_i)$ on its total load.

We formulate the variable-to-bucket assignment as:

$$\text{(IP-TH)} \tag{4.13}$$

$$\xi_{ij} \in \{0,1\} \qquad \forall i \in [M], j \in [n] \tag{4.14}$$

$$\sum_{i=1}^{M} \xi_{ij} \leq 1 \qquad \forall j \in [n] \tag{4.15}$$

$$\sum_{j=1}^{n} \xi_{ij} \leq 1 \qquad \forall i \in [M] : B_i \text{ is a jump bucket} \tag{4.16}$$

$$\sum_{j=1}^{n} \max_{a} \left\{ [(r_j(a) - \text{BaseGuess}(B_i)) \cdot q_j(a)]^+ \right\} \xi_{ij} \qquad \forall i \in [M] \tag{4.17}$$

$$\geq \text{DeltaGuess}(B_i)$$

$$\sum_{i=1}^{M} \sum_{j=1}^{n} \xi_{ij} \leq \ell \tag{4.18}$$

In order to use Theorem 4.4, we need to prove that the IP above admits a feasible solution.

**Lemma 4.5** (Proof in Appendix C.2)**.** *IP-TH admits a feasible solution.*

**Corollary 4.6** (of Theorem 4.4)**.** *There is an EPTAS for computing a cardinality-feasible variable-to-machine assignment $\boldsymbol{\xi} \in \{0,1\}^{(2K+1)\times n}$ such that every machine $i \in [2K+1]$ receives a total load of $\sum_{j\in[n]} c_{ij}\xi_{ij} \geq (1-\epsilon)\mathit{DeltaGuess}(B_i)$.*

### 4.1.3 Step 3: Policy Reconstruction

Given the assignment vector $\xi$ from Corollary 4.6, we construct a policy $(\boldsymbol{e}, \boldsymbol{a})$ as follows: We define the ordering by processing buckets in reverse order. Specifically: The last edges to be queried are those assigned to bucket $B_1$ (corresponding to the stable bucket after the last jump). Just before them, we query those assigned to bucket $B_2$ (the last jump bucket). Continuing this pattern through all buckets $B_3, B_4, \ldots, B_{2K+1}$. Within each bucket, the internal ordering of edges is arbitrary.

Let $e$ be the resulting ordering of the at most $\ell$ selected edges. We compute the optimal actions and expected future values for our ordering $e$ using dynamic programming. Let $\tilde{R}_i$ denote the maximum expected value achievable from positions $i, i+1, \ldots, \ell$ when using optimal actions. We compute:

$$\tilde{R}_{\ell+1} = 0$$
$$\tilde{R}_i = \max_{a\in\mathcal{A}} \left\{ r_{e_i}(a) \cdot q_{e_i}(a) + \tilde{R}_{i+1} \cdot (1 - q_{e_i}(a)) \right\}$$

for $i = \ell, \ell-1, \ldots, 1$. The optimal action for position $i$ is:

$$a_i = \arg\max_{a\in\mathcal{A}} \left\{ r_{e_i}(a) \cdot q_{e_i}(a) + \tilde{R}_{i+1} \cdot (1 - q_{e_i}(a)) \right\}$$

The expected value of our reconstructed policy is $\text{val}_r(\boldsymbol{e}, \boldsymbol{a}) = \tilde{R}_1$. We say our policy is *behind schedule* at position $i$ if $\tilde{R}_{i+1} < \text{BaseGuess}(B_{e_i}^{-1})$, where $B_{e_i}^{-1}$ denotes the bucket to which edge $e_i$ is assigned. Otherwise, the policy is *ahead of schedule*.

Let $i_{\min}$ be the first position where we are ahead of schedule (this is well-defined since we are always ahead of schedule at the last position, as $\tilde{R}_{\ell+1} = 0 = \text{BaseGuess}(B_1)$). By the recursive definition, we can decompose the total expected value as:

$$\text{val}_{r,q}(e,a) = \tilde{R}_1 = \tilde{R}_{i_{\min}} + \sum_{i=1}^{i_{\min}-1} (\tilde{R}_i - \tilde{R}_{i+1})$$

**Lemma 4.7** (Proof in Appendix C.3)**.** *The following inequality holds:*

$$\tilde{R}_{i_{\min}} \geq \mathit{BaseVal}(B^{-1}_{e_{i_{\min}}}) + (1-\epsilon) \cdot \mathit{DeltaGuess}(B^{-1}_{e_{i_{\min}}}) - 2\epsilon \cdot \mathit{OPT}.$$

**Lemma 4.8** (Proof in Appendix C.4)**.** *The following inequality holds:*

$$\sum_{i=1}^{i_{\min}-1} (\tilde{R}_i - \tilde{R}_{i+1}) \geq (1-\epsilon) \cdot \sum_{j > B^{-1}_{e_{i_{\min}}}} (\mathit{BaseVal}(B_{j+1}) - \mathit{BaseVal}(B_j)) - 3\epsilon \cdot \mathit{OPT}.$$

We can now prove that the reconstructed policy $(\boldsymbol{e}, \boldsymbol{a})$ achieves an expected reward of at least $(1-7\epsilon) \cdot \text{OPT}$.

*Proof of Theorem 4.2.* Combining Lemmas 4.7 and 4.8,

$$\begin{aligned}
\text{val}_{\boldsymbol{r},\boldsymbol{q}}(\boldsymbol{e},\boldsymbol{a}) &\geq \text{BaseVal}(B^{-1}_{e_{i_{\min}}}) + (1-\epsilon) \cdot \text{DeltaGuess}(B^{-1}_{e_{i_{\min}}}) - 2\epsilon \cdot \text{OPT} \\
&\quad + (1-\epsilon) \cdot \sum_{j > B^{-1}_{e_{i_{\min}}}} (\text{BaseVal}(B_{j+1}) - \text{BaseVal}(B_j)) - 3\epsilon \cdot \text{OPT} \\
&\geq (1-7\epsilon) \cdot \text{OPT},
\end{aligned}$$

where the final inequality holds for $\varepsilon$ sufficiently small. □

# 5 Putting it all together: Proving Theorem 1.1

Suppose we are given a graph $G = (U, V, E)$ with actions $\mathcal{A}$, edge rewards $\boldsymbol{r} = (r_e(a))_{e \in E, a \in \mathcal{A}}$ and edge probabilities $\boldsymbol{q} = (q_e(a))_{e \in E, a \in \mathcal{A}}$, and recall that $\beta := 1 - 1/e$ if $\ell_u \in \{1, \infty\}$ for all $u \in U$, and $\beta := 27(19 - \frac{67}{e^3})$ otherwise. For each $\varepsilon > 0$, we design a policy whose expected reward is at least $(1-\varepsilon) \cdot \beta \cdot \text{OPT}(G)$, where $\text{OPT}(G)$ is the expected reward of the optimal policy.

For any $v \in V$, we can apply Theorem 4.2, to the maximization problem (4.3), and solve it up to a multiplicative factor of $(1-\varepsilon)$. Thus, by Proposition 4.1, we can compute a solution to LP-C, say $(z_v(\boldsymbol{e},\boldsymbol{a}))_{v \in V, (\boldsymbol{e},\boldsymbol{a}) \in \mathcal{C}_v}$, for which

$$\sum_{v \in V} \sum_{(\boldsymbol{e},\boldsymbol{a}) \in \mathcal{C}_v} \text{val}(\boldsymbol{e},\boldsymbol{a}) z_v(\boldsymbol{e},\boldsymbol{a}) \geq (1-\varepsilon)\text{LP}(G), \tag{5.1}$$

where $\text{LP}(G)$ is the optimal value of LP-C. Now, by Theorem 3.4, there is a $\beta$-selectable P-RCRS $\psi_u$ for each $u \in U$. By passing $(z_v(\boldsymbol{e},\boldsymbol{a}))_{v \in V, (\boldsymbol{e},\boldsymbol{a}) \in \mathcal{C}_v}$ and these P-RCRS's $(\psi_u)_{u \in U}$ to Algorithm 2, Theorem 3.3 implies that we get a policy whose expected reward is at least

$$\beta \cdot \sum_{v \in V} \sum_{(\boldsymbol{e},\boldsymbol{a}) \in \mathcal{C}_v} \text{val}(\boldsymbol{e},\boldsymbol{a}) z_v(\boldsymbol{e},\boldsymbol{a}) \geq \beta(1-\varepsilon)\text{LP}(G) \geq (1-\varepsilon)\text{OPT}(G), \tag{5.2}$$

where first inequality applies (5.1), and the second inequality applies Theorem 2.1. Thus, (5.2) implies an approximation ratio of $(1-\varepsilon) \cdot \beta$ as claimed in Theorem 1.1. Finally, the runtime follows due to the runtime of Theorem 4.2, Proposition 4.1, and the fact that the remaining steps (i.e., the P-RCRS's of Theorem 3.4 and Algorithm 2) are polynomial time, and in fact do not depend on $\varepsilon$.

## 6 Details of Reductions

In this section, we show how both the sequential pricing problem and the free-order prophet matching problem can be reduced to our *action-reward query-commit matching problem*.

**Sequential pricing.** In the sequential pricing problem introduced by [PRSW24], the input is a bipartite graph, say $G = (U, V, E)$, in which $U$ are the *jobs*, each of which has a known value $b_u \geq 0$ to the platform for being completed, and $V$ are the *workers*. The platform may query an edge $e = (u, v)$ by offering a worker $v \in V$ a payment $\tau \geq 0$ to perform job $u$. The offer is then accepted independently with a known probability $p_{u,v}(\tau)$. As in our setting, each vertex is associated with a known patience constraint limiting the number of incident payment offers, and each $e = (u, v) \in E$ may be queried via at most one payment. The platform's objective is to maximize either its expected *revenue* or its expected *social welfare*.

This problem naturally reduces to our *action-reward query-commit matching problem* by appropriately defining the action space. Specifically, for each edge $e = (u, v)$, we introduce an action $a \in \mathcal{A}$ for every payment $\tau \geq 0$. Querying edge $e$ using the action corresponding to $\tau$ succeeds with probability $q_e(a) = p_{u,v}(\tau)$. The reward associated with an action depends on the optimization objective. For the revenue objective, the reward of the action $a$ corresponding to payment $\tau$ is the platform's revenue $r_e(a) = b_u - \tau$, where $b_u$ denotes the value of job $u$. For the social welfare objective, the reward is given by $r_e(a) = b_u - \mathbb{E}[C_{u,v} \mid C_{u,v} \leq \tau]$, where $C_{u,v}$ is a random variable representing the cost incurred by worker $v$ for performing job $u$, drawn from a known distribution.

**Free-order prophet matching.** In the *free-order prophet matching problem*, each edge $e$ of a graph $G$ has a weight $W_e$ drawn independently from a known distribution $\mathcal{D}_e$. An online algorithm processes the edges sequentially in an order of its choosing; upon processing an edge $e$, it observes the realization of $W_e$ and makes an irrevocable decision on whether to add $e$ to the current matching. The objective is to maximize the total weight of the resulting matching.

It is known that this problem admits an optimal threshold policy: before processing any edge $e$, the policy selects a threshold $\tau_e$ and includes $e$ in the matching if and only if $W_e \geq \tau_e$. Moreover, it suffices to consider thresholds $\tau_e$ drawn from the support of $\mathcal{D}_e$. To reduce this problem to our action-reward query-commit matching framework, we introduce, for each edge $e$, an action $a \in \mathcal{A}$ corresponding to each feasible threshold $\tau_e$. Querying edge $e$ using the action associated with threshold $\tau_e$ succeeds with probability

$$q_e(a) = \mathbb{P}[W_e \geq \tau_e],$$

and yields a reward

$$r_e(a) = \mathbb{E}[W_e \mid W_e \geq \tau_e].$$

We can also capture the revenue objective for this problem in a similar way.

This concludes our reduction of both the sequential pricing problem and the free-order prophet matching problem to the action-reward query-commit matching problem.

## References

[Ada11] Marek Adamczyk, *Improved analysis of the greedy algorithm for stochastic matching*, Inf. Process. Lett. **111** (2011), no. 15, 731–737.

[AGM15] Marek Adamczyk, Fabrizio Grandoni, and Joydeep Mukherjee, *Improved approximation algorithms for stochastic matching*, Algorithms - ESA 2015 - 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings (Nikhil Bansal and Irene Finocchi, eds.), Lecture Notes in Computer Science, vol. 9294, Springer, 2015, pp. 1–12.

[ASZ20] Shipra Agrawal, Jay Sethuraman, and Xingyu Zhang, *On optimal ordering in the optimal stopping problem*, Proceedings of the 21st ACM Conference on Economics and Computation (New York, NY, USA), EC '20, Association for Computing Machinery, 2020, p. 187–188.

[BCN+18] Alok Baveja, Amit Chavan, Andrei Nikiforov, Aravind Srinivasan, and Pan Xu, *Improved bounds in stochastic matching and optimization*, Algorithmica **80** (2018), no. 11, 3225–3252.

[Ben62] George Bennett, *Probability inequalities for the sum of independent random variables*, Journal of the American Statistical Association **57** (1962), no. 297, 33–45.

[BGL+12] Nikhil Bansal, Anupam Gupta, Jian Li, Julián Mestre, Viswanath Nagarajan, and Atri Rudra, *When LP is the cure for your matching woes: Improved bounds for stochastic matchings*, Algorithmica **63** (2012), no. 4, 733–762.

[BGMS24] Brian Brubach, Nathaniel Grammel, Will Ma, and Aravind Srinivasan, *Improved guarantees for offline stochastic matching via new ordered contention resolution schemes*, Mathematics of Operations Research 0(0) (2024).

[BGMS25] Brian Brubach, Nathaniel Grammel, Will Ma, and Aravind Srinivasan, *Online matching frameworks under stochastic rewards, product ranking, and unknown patience*, Oper. Res. **73** (2025), no. 2, 995–1010.

[BM25] Allan Borodin and Calum MacRury, *Online bipartite matching in the probe-commit model*, Mathematical Programming (2025), 1–54.

[BSSX20] Brian Brubach, Karthik Abinav Sankararaman, Aravind Srinivasan, and Pan Xu, *Attenuate locally, win globally: Attenuation-based frameworks for online stochastic matching with timeouts*, Algorithmica **82** (2020), no. 1, 64–87.

[CHLT25] Ziyun Chen, Zhiyi Huang, Dongchen Li, and Zhihao Gavin Tang, *Prophet secretary and matching: the significance of the largest item*, pp. 1371–1401, 2025.

[CIK+09] Ning Chen, Nicole Immorlica, Anna R. Karlin, Mohammad Mahdian, and Atri Rudra, *Approximating matches made in heaven*, Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part I, ICALP '09, 2009, pp. 266–278.

[CTT12] Kevin P. Costello, Prasad Tetali, and Pushkar Tripathi, *Stochastic matching with commitment*, Automata, Languages, and Programming (Berlin, Heidelberg) (Artur Czumaj, Kurt Mehlhorn, Andrew Pitts, and Roger Wattenhofer, eds.), Springer Berlin Heidelberg, 2012, pp. 822–833.

[DF23] Mahsa Derakhshan and Alireza Farhadi, *Beating (1 - 1/e)-approximation for weighted stochastic matching*, Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023 (Nikhil Bansal and Viswanath Nagarajan, eds.), SIAM, 2023, pp. 1931–1961.

[Fei06] Uriel Feige, *On sums of independent random variables with unbounded variance and estimating the average degree in a graph*, SIAM Journal on Computing **35** (2006), no. 4, 964–984.

[FTW+21] Hu Fu, Zhihao Gavin Tang, Hongxun Wu, Jinzhao Wu, and Qianfan Zhang, *Random Order Vertex Arrival Contention Resolution Schemes for Matching, with Applications*, 48th International Colloquium on Automata, Languages, and Programming (ICALP 2021) (Dagstuhl, Germany) (Nikhil Bansal, Emanuela Merelli, and James Worrell, eds.), Leibniz International Proceedings in Informatics (LIPIcs), vol. 198, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, pp. 68:1–68:20.

[GKPS06] Rajiv Gandhi, Samir Khuller, Srinivasan Parthasarathy, and Aravind Srinivasan, *Dependent rounding and its applications to approximation algorithms*, J. ACM **53** (2006), no. 3, 324–360.

[GKS19] Buddhima Gamlath, Sagar Kale, and Ola Svensson, *Beating greedy for stochastic bipartite matching*, Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms (USA), SODA '19, Society for Industrial and Applied Mathematics, 2019, p. 2841–2854.

[HSWZ25] Zhiyi Huang, Enze Sun, Xiaowei Wu, and Jiahao Zhao, *Edge-weighted matching in the dark*, IEEE 66th Annual Symposium on Foundations of Computer Science (FOCS 2025), 2025.

[Jan03] Klaus Jansen, *Approximate strong separation with application in fractional graph coloring and preemptive scheduling*, Theor. Comput. Sci. **302** (2003), no. 1-3, 239–256.

[KS81] Richard M Karp and Michael Sipser, *Maximum matching in sparse random graphs*, 22nd Annual Symposium on Foundations of Computer Science (sfcs 1981), IEEE, 1981, pp. 364–375.

[KV12] Bernhard Korte and Jens Vygen, *Combinatorial optimization: Theory and algorithms*, 5th ed., Springer Publishing Company, Incorporated, 2012.

[LS18] Euiwoong Lee and Sahil Singla, *Optimal Online Contention Resolution Schemes via Ex-Ante Prophet Inequalities*, 26th Annual European Symposium on Algorithms (ESA 2018) (Dagstuhl, Germany) (Yossi Azar, Hannah Bast, and Grzegorz Herman, eds.), Leibniz International Proceedings in Informatics (LIPIcs), vol. 112, Schloss Dagstuhl– Leibniz-Zentrum fuer Informatik, 2018, pp. 57:1–57:14.

[MM24] Calum MacRury and Will Ma, *Random-order contention resolution via continuous induction: Tightness for bipartite matching under vertex arrivals*, Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024 (Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, eds.), ACM, 2024, pp. 1629–1640.

[MMG23] Calum MacRury, Will Ma, and Nathaniel Grammel, *On (random-order) online contention resolution schemes for the matching polytope of (bipartite) graphs*, Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023 (Nikhil Bansal and Viswanath Nagarajan, eds.), SIAM, 2023, pp. 1995–2014.

[Pau17] Roland Paulin, *On some conjectures of samuels and feige*, 2017.

[PGR19] Manish Purohit, Sreenivas Gollapudi, and Manish Raghavan, *Hiring under uncertainty*, Proceedings of the 36th International Conference on Machine Learning (Kamalika Chaudhuri and Ruslan Salakhutdinov, eds.), Proceedings of Machine Learning Research, vol. 97, PMLR, 09–15 Jun 2019, pp. 5181–5189.

[PRSW24] Tristan Pollner, Mohammad Roghani, Amin Saberi, and David Wajc, *Improved online contention resolution for matchings and applications to the gig economy*, Math. Oper. Res. **49** (2024), no. 3, 1582–1606.

[PT92] J.E. Peajcariaac and Y.L. Tong, *Convex functions, partial orderings, and statistical applications*, Mathematics in Science and Engineering, Elsevier Science, 1992.

[Sam66] S. M. Samuels, *On a chebyshev-type inequality for sums of independent random variables*, The Annals of Mathematical Statistics **37** (1966), no. 1, 248–259.

[Sam69] S. M. Samuels, *The markov inequality for sums of independent random variables*, The Annals of Mathematical Statistics **40** (1969), no. 6, 1980–1984.

[SS21] Danny Segev and Sahil Singla, *Efficient approximation schemes for stochastic probing and prophet problems*, Proceedings of the 22nd ACM Conference on Economics and Computation (New York, NY, USA), EC '21, Association for Computing Machinery, 2021, p. 793–794.

[SS25] Danny Segev and Sahil Singla, *Efficient approximation schemes for stochastic probing and selection-stopping problems*, 2025.

# A Proof of Theorem 2.1 from Section 2

We introduce a relaxed action-reward problem where: (1) vertices in $U$ must be matched at most once in expectation, rather than at most once with probability 1, and (2) each vertex $u \in U$ has an expected patience constraint of at most $\ell_u$ queries, rather than a hard limit of $\ell_u$ queries.

Let $\overline{\mathrm{OPT}}(G)$ be the optimal value of this relaxed problem. Since this is a relaxation of the original constraints, we have $\mathrm{OPT}(G) \leq \overline{\mathrm{OPT}}(G)$. Therefore, to complete the proof, we will show that $\overline{\mathrm{OPT}}(G) \leq \mathrm{LP}(G)$.

Let OptAlg be an optimal policy for the relaxed problem. We construct a *vertex-iterative* policy Alg for the relaxed problem that achieves the same expected reward as OptAlg. In other words, Alg will process vertices in $V$ one by one in an arbitrary order, dealing completely with all edges incident to a vertex before moving to the next.

For each vertex $v \in V$, Alg runs a simulation of OptAlg where for any edge $f \notin \partial(v)$ and action $a \in \mathcal{A}$, it draws a simulated Bernoulli bit $\widetilde{Q}_f(a) \sim \mathrm{Ber}(q_f(a))$ and gives it to OptAlg as the real query outcome. Based on this simulation, Alg determines the adaptive ordering of edges and specific actions in $\partial(v)$ that OptAlg would use, without actually querying edges incident to

other vertices. It then follows this induced adaptive policy when processing $v$. After processing $v$, Alg discards all simulated information and moves to the next vertex.

Let $p_v(\boldsymbol{e}, \boldsymbol{a})$ be the probability that OptAlg uses the sequence-action pair $(\boldsymbol{e}, \boldsymbol{a}) \in \mathcal{C}_v$ when processing vertex $v$. Since Alg faithfully simulates the behavior of OptAlg with respect to the marginal distributions of the outcomes, it induces the same distribution over sequence-action pairs for each vertex $v \in V$. Therefore, the distributions over matched and queried edges are identical between Alg and OptAlg.

We now verify that Alg satisfies the relaxed constraints, and is an optimal policy for the relaxed problem:

- *Matching constraint:* For each $u \in U$, the probability that $u$ is matched is the sum of the probabilities that each incident edge $(u, v)$ is selected. Since Alg mirrors OptAlg, this expected total is at most 1.
- *Query budget constraint:* For each vertex $u \in U$, the expected number of queries to edges in $\partial(u)$ equals that of OptAlg and is at most $\ell_u$ by assumption.
- *Objective value:* For any edge $e = (u, v)$ and action $a$, the probability it is queried with action $a$ depends only on the sequence-action pair chosen for vertex $v$. Since these distributions are identical between the algorithms, the expected reward of Alg equals that of OptAlg. By definition, OptAlg is an optimal policy for the relaxed problem, so its expected objective value is $\overline{\mathrm{OPT}}(G)$. Therefore, the expected objective value of Alg is also $\overline{\mathrm{OPT}}(G)$.

Finally, we create a solution to LP-C. Let $z_v(\boldsymbol{e}, \boldsymbol{a}) := p_v(\boldsymbol{e}, \boldsymbol{a})$ for all $v \in V$ and $(\boldsymbol{e}, \boldsymbol{a}) \in \mathcal{C}_v$. We verify that this solution satisfies all LP constraints:

- Constraint (2.4) requires $\sum_{(\boldsymbol{e},\boldsymbol{a})\in\mathcal{C}_v} z_v(\boldsymbol{e}, \boldsymbol{a}) \le 1$ for each $v \in V$. Since $\sum_{(\boldsymbol{e},\boldsymbol{a})\in\mathcal{C}_v} z_v(\boldsymbol{e}, \boldsymbol{a}) = \sum_{(\boldsymbol{e},\boldsymbol{a})\in\mathcal{C}_v} p_v(\boldsymbol{e}, \boldsymbol{a}) \le 1$ (as Alg must choose some policy for each vertex, or query none of its edges), this constraint is satisfied.
- Constraint (2.2) requires $\sum_{v\in V} \sum_{(\boldsymbol{e},\boldsymbol{a})\in\mathcal{C}_v : e_i=(u,v)} q_{u,v}(a_i) \cdot \prod_{j<i}(1 - q_{e_j}(a_j)) \cdot z_v(\boldsymbol{e}, \boldsymbol{a}) \le 1$ for all $u \in U$. The left side equals the expected number of edges assigned to $u$ by Alg, which is at most 1 in the relaxed problem.
- Constraint (2.3) requires $\sum_{v\in V} \sum_{(\boldsymbol{e},\boldsymbol{a})\in\mathcal{C}_v : e_i=(u,v)} \prod_{j<i}(1-q_{e_j}(a_j)) \cdot z_v(\boldsymbol{e}, \boldsymbol{a}) \le \ell_u$ for all $u \in U$. The left side equals the expected number of queries to edges incident to $u$ by Alg, which is at most $\ell_u$ by the relaxed patience constraint.

Given that our solution satisfies all constraints of LP-C and its objective value equals $\overline{\mathrm{OPT}}(G)$, we have $\overline{\mathrm{OPT}}(G) \le \mathrm{LP}(G)$. Combining with our earlier inequality, we conclude that $\mathrm{OPT}(G) \le \mathrm{LP}(G)$. □

# B Details of Section 3

## B.1 Proof of Lemma 3.1

First observe that for any edge $e \in E$, $\sum_{a\in\mathcal{A}} Z_e(a) \le 1$, as Algorithm 1 queries $e$ via at most one action. Thus, since $\mathbb{E}[Z_e(a)] = \widetilde{z}_e(a)$ by definition (see (3.2)),

$$\sum_{a\in\mathcal{A}} \widetilde{z}_e(a) = \sum_{a\in\mathcal{A}} \mathbb{E}[Z_e(a)] \le 1,$$

which verifies the first property.

Now, if we fix $u \in U$, then we write the expected number of $v \in V$ with $\sum_{a \in \mathcal{A}} Z_{u,v}(a) = 1$ as:

$$\sum_{v \in V} \sum_{a \in \mathcal{A}} \widetilde{z}_{u,v}(a) = \sum_{v \in V} \sum_{a \in \mathcal{A}} \mathbb{P}[Z_{u,v}(a) = 1] = \sum_{v \in V} \sum_{\substack{i \in [\ell_v], (\boldsymbol{e}, \boldsymbol{a}) \in \mathcal{C}_v : \\ e_i = (u,v)}} \sum_{j < i} \prod (1 - q_{e_j}(a_j)) \cdot z_v(\boldsymbol{e}, \boldsymbol{a}).$$

More, we can relate the expected number of edges $(u, v) \in \partial(u)$ which are included in $\mathcal{N}$:

$$\sum_{v \in V} \sum_{a \in \mathcal{A}} q_{u,v}(a) \widetilde{z}_{u,v}(a) = \sum_{v \in V} \mathbb{P}[(u, v) \in \mathcal{N}] = \sum_{v \in V} \sum_{\substack{i \in [\ell_v], (\boldsymbol{e}, \boldsymbol{a}) \in \mathcal{C}_v : \\ e_i = (u,v)}} q_{u,v}(a_i) \cdot \prod_{j < i} (1 - q_{e_j}(a_j)) \cdot z_v(\boldsymbol{e}, \boldsymbol{a}).$$

By applying constraints (2.3) and (2.2) of LP-C, the remaining properties hold, and so the lemma follows. □

## B.2 Proof of Lemma 3.2

As mentioned following Algorithm 2, for any $u \in U$, $\psi_u$ is passed the input

$$(\mathcal{A}, \partial(u), (q_{u,v}(a))_{v \in V, a \in \mathcal{A}}, (\widetilde{z}_{u,v}(a))_{v \in V, a \in \mathcal{A}}),$$

which satisfies the required inequalities in Definition 1, due to Lemma 3.1. We now verify that random variables used in line 8. are drawn via $(q_{u,v}(a))_{v \in V, a \in \mathcal{A}}$ and $(\widetilde{z}_{u,v}(a))_{v \in V, a \in \mathcal{A}}$ in the required way as specified in Definition 1.

Clearly, the random variables $(Q_{u,v}(a))_{a \in \mathcal{A}, v \in V}$ are independent and have the correct marginals. On the other hand, notice that since each $v \in V$ draws $(\boldsymbol{e}, \boldsymbol{a}) \in \mathcal{C}_v$ independently, $((\widetilde{Z}_{u,v}(a))_{a \in \mathcal{A}})_{v \in V}$ are independent across $v \in V$. Moreover, $\sum_{a \in \mathcal{A}} \widetilde{Z}_{u,v}(a) \leq 1$ for each $v \in V$, as $(u, v)$ is queried by at most once action. We next argue that $(\widetilde{Z}_e(a))_{e \in E, a \in \mathcal{A}}$ have the correct marginals, i.e., for each $e \in E$ and $a \in \mathcal{A}$, $\mathbb{P}[\widetilde{Z}_e(a) = 1] = \widetilde{z}_e(a)$, which will ensure Algorithm 2 is well-defined, and complete the proof.

Let us fix first fix $v \in V$, and define $\mathcal{P}_v \in \mathcal{C}_v$ to be the random policy drawn by $v$ in Algorithm 2. More, for each $e \in \partial(v)$ and $a \in \mathcal{A}$, set $F_e(a) := 1$ if and only if $e$ is queried via $a$ by Algorithm 2. Define the vector $\boldsymbol{F}^{(v)} := (F_e(a))_{a \in \mathcal{A}, e \in \partial(v)}$ for convenience.

Now, consider a fixed $u^* \in U$ and $a^* \in \mathcal{A}$, as well as $(\boldsymbol{e}, \boldsymbol{a}) \in \mathcal{C}_v$ (a possible instantiation of $\mathcal{P}_v$). We refer to $(\boldsymbol{e}, \boldsymbol{a})$ as *permissible* (for $(u^*, v)$ and $a^*$), provided $e_i = (u^*, v)$ and $a_i = a^*$ for some $1 \leq i \leq |\boldsymbol{e}|$. First observe that if $(\boldsymbol{e}, \boldsymbol{a})$ is *not* permissible, then

$$\mathbb{P}[\widetilde{Z}_{u^*,v}(a^*) = 1 \mid \mathcal{P}_v = (\boldsymbol{e}, \boldsymbol{a})] = 0 \tag{B.1}$$

Thus, shall compute $\mathbb{P}[\widetilde{Z}_{u^*,v}(a^*) = 1 \mid \mathcal{P}_v = (\boldsymbol{e}, \boldsymbol{a})]$ for each $(\boldsymbol{e}, \boldsymbol{a})$ which is permissible. Let us assume that $e_i = (u^*, v)$ and $a_i = a^*$ for some $1 \leq i \leq |\boldsymbol{e}|$. Finally, we consider an arbitrary vector $\boldsymbol{f}^{(v)} = (f_e(a))_{a \in \mathcal{A}, e \in \partial(v)}$, where $f_e(a) \in \{0, 1\}$.

Suppose now that we condition on $\mathcal{P}_v = (\boldsymbol{e}, \boldsymbol{a})$, and $\boldsymbol{F}^{(v)} = \boldsymbol{f}^{(v)}$. Observe then that $(u^*, v)$ is queried via $a^*$ if and only if $Q_{e_j}(a_j) = 0$ for each $j < i$ with $f_{e_j}(a_j) = 1$ (i.e., all "real" queries fail) and $\widetilde{Q}_{e_{j'}}(a_{j'}) = 0$ for all $j' < i$ with $f_{e_{j'}}(a_{j'}) = 0$ (i.e., all "fake" queries fail). Thus, we can write $\mathbb{E}[\widetilde{Z}_{u^*,v}(a^*) \mid \mathcal{P}_v = (\boldsymbol{e}, \boldsymbol{a}), \boldsymbol{F}^{(v)} = \boldsymbol{f}^{(v)}]$ as:

$$\mathbb{E}\left[ \prod_{j < i : f_{e_j}(a_j) = 1} (1 - Q_{e_j}(a_j)) \prod_{j' < i : f_{e_{j'}}(a_{j'}) = 0} (1 - \widetilde{Q}_{e_{j'}}(a_{j'})) \mid \mathcal{P}_v = (\boldsymbol{e}, \boldsymbol{a}), \boldsymbol{F}^{(v)} = \boldsymbol{f}^{(v)} \right]. \tag{B.2}$$

On the other hand, we can further simplify (B.2) as:

$$\prod_{\substack{j<i:\\ f_{e_j}(a_j)=1}} \mathbb{E}[1-Q_{e_{j'}}(a_j) \mid \mathcal{P}_v=(\boldsymbol{e},\boldsymbol{a}), \boldsymbol{F}^{(v)}=\boldsymbol{f}^{(v)}] \prod_{\substack{j'<i:\\ f_{e_{j'}}(a_{j'})=0}} \mathbb{E}[1-\widetilde{Q}_{e_{j'}}(a_{j'}) \mid \mathcal{P}_v=(\boldsymbol{e},\boldsymbol{a}), \boldsymbol{F}^{(v)}=\boldsymbol{f}^{(v)}]$$

$$=\prod_{j<i}(1-q_{e_j}(a_j)).$$

Here we've used that Algorithm 2 decides whether or not to query any $e \in \partial(v)$ via $a$, prior to revealing $Q_e(a)$ or $\widetilde{Q}_e(a)$. The last equality uses this fact once more, as well as that $Q_e(a)$ and $\widetilde{Q}_e(a)$ are distributed identically. After averaging over the instantiations of $\boldsymbol{F}^{(v)}$, we get that

$$\mathbb{P}[\widetilde{Z}_{u^*,v}(a^*)=1 \mid \mathcal{P}_v=(\boldsymbol{e},\boldsymbol{a})]=\prod_{j<i}(1-q_{e_j}(a_j)).$$

Finally, after averaging over all permissible $(\boldsymbol{e},\boldsymbol{a})$ for $(u^*,v)$ and $a^*$, we get that

$$\mathbb{P}[\widetilde{Z}_{u^*,v}(a^*)=1]=\sum_{\substack{i\in[\ell_v],(\boldsymbol{e},\boldsymbol{a})\in\mathcal{C}_v:\\ e_i=(u^*,v),a_i=a^*}} \prod_{j<i}(1-q_{e_j}(a_j)) \cdot z_v(\boldsymbol{e},\boldsymbol{a})=\widetilde{z}_{u^*,v}(a^*),$$

where the final equality follows by (3.2). □

### B.3 Proof of Lemma 3.5

Suppose that $(\mathcal{A},n,\ell,(p_i(a))_{a\in\mathcal{A},i\in[n]},(x_i(a))_{a\in\mathcal{A},i\in[n]})$ is a P-RCRS input for an arbitrary action set $\mathcal{A}$. We first construct a P-RCRS input $(n,\ell,(p_i)_{i=1}^n,(x_i)_{i=1}^n)$ on the trivial action set, where for each $i\in[n]$

$$x_i:=\sum_{a\in\mathcal{A}} x_i(a) \text{ and } p_i:=\frac{\sum_{a\in\mathcal{A}} p_i(a)x_i(a)}{\sum_{a\in\mathcal{A}} x_i(a)}. \tag{B.3}$$

First observe that $(n,\ell,(x_i)_{i=1}^n,(p_i)_{i=1}^n)$ is indeed a valid P-RCRS input, as

$$\sum_{i=1}^n x_i=\sum_{i=1}^n\sum_{a\in\mathcal{A}} x_i(a)\le\ell,\ \sum_{i=1}^n p_ix_i=\sum_{i=1}^n\sum_{a\in\mathcal{A}} p_i(a)x_i(a)\le 1, \text{ and } x_i=\sum_{a\in\mathcal{A}} x_i(a)\le 1, \tag{B.4}$$

where the inequalities follow since $(\mathcal{A},n,\ell,(p_i(a))_{a\in\mathcal{A},i\in[n]},(x_i(a))_{a\in\mathcal{A},i\in[n]})$ is a P-RCRS input.

Now suppose that $\psi$ is a P-RCRS which is $\alpha$-selectable on $(n,\ell,(p_i)_{i=1}^n,(x_i)_{i=1}^n)$. We shall use it to design a P-RCRS $\psi'$ for $(\mathcal{A},n,\ell,(p_i(a))_{a\in\mathcal{A},i\in[n]},(x_i(a))_{a\in\mathcal{A},i\in[n]})$. Let us first assume that $(P_i(a))_{a\in\mathcal{A},i\in[n]}$ and $(X_i(a))_{a\in\mathcal{A},i\in[n]}$ are the states and suggestions for the latter input, as draw in Definition 1.

We now define the random variables $(P_i,X_i)_{i=1}^n$ with the following properties:

1. $(P_i,X_i)_{i\in[n]}$ are independent,
2. $X_i=\sum_{a\in\mathcal{A}} X_i(a)$ and $P_iX_i=\sum_{a\in\mathcal{A}} P_i(a)X_i(a)$ for all $i\in[n]$.

We shall use these as the states and suggestions for the execution of $\psi$ on $(n,\ell,(x_i)_{i=1}^n,(p_i)_{i=1}^n)$. To see that these properties are attainable, draw $\widetilde{P}_i\sim\mathrm{Ber}(p_i)$ independently for each $i\in[n]$. Then, we define $P_i$ in the following way:

$$P_i=\begin{cases}1 & \text{if } \sum_{a\in\mathcal{A}} P_i(a)X_i(a)=1 \text{ and } \sum_{a\in\mathcal{A}} X_i(a)=1\\ 0 & \text{if } \sum_{a\in\mathcal{A}} P_i(a)X_i(a)=0 \text{ and } \sum_{a\in\mathcal{A}} X_i(a)=1,\\ \widetilde{P}_i & \text{otherwise.}\end{cases}$$

It is easy to see that this construction satisfies the above properties. More, if $\pi$ is the arrival order of $[n]$, then $(P_i, X_i)_{i\in[n]}$ can be constructed in an online fashion with respect to $\pi$ as a P-RCRS executes. Specifically, when $i \in [n]$ arrives and $(X_i(a))_{a\in\mathcal{A}}$ are revealed, we set $X_i = \sum_{a\in\mathcal{A}} X_i(a)$. More, if $\sum_{a\in\mathcal{A}} X_i(a) = 0$, then we can clearly drawn $\widetilde{P}_i$ and set $P_i = \widetilde{P}_i$. On the other hand, if $\sum_{a\in\mathcal{A}} X_i(a) = 1$ and we query $i$ via some action $a$, then we can set $P_i \in \{0, 1\}$, depending on if $Q_i(a) = 0$ or $Q_i(a) = 1$. If we do *not* query $i$ via any action, then since we will never output $(i, a)$ for any action $a \in \mathcal{A}$, we can still reveal $(Q_i(a))_{a\in\mathcal{A}}$ and assign $P_i$ in this way.

Our P-RCRS $\psi'$ for $(\mathcal{A}, n, \ell, (p_i(a))_{a\in\mathcal{A},i\in[n]}, (x_i(a))_{a\in\mathcal{A},i\in[n]})$ executes $\psi$ on $(n, \ell, \boldsymbol{x}, \boldsymbol{p})$ in the same order $\pi$, using the coupled random variables $(P_i, X_i)_{i=1}^n$ to make its decisions. Specifically, when processing $i \in [n]$, it learns $(X_i(a))_{a\in\mathcal{A}}$. If $X_i(a) = 1$ for some $a \in \mathcal{A}$, then $X_i = 1$, and it asks $\psi$ whether to make a query. If 'yes', then it queries $i$ via $a$, learns $P_i(a)$, and outputs $(i, a)$ if $P_i(a) = 1$. If 'no', then it still updates $P_i$ as previously described, and moves to the next element with respect to $\pi$. If $\sum_{a\in\mathcal{A}} X_i(a) = 0$, then $X_i = 0$, and it sets $P_i = \widetilde{P}_i$, and moves to the next element.

Now, $\psi$ is $\alpha$-selectable by assumption, so for each $i \in [n]$,

$$\Pr[i \text{ is queried by } \psi \mid X_i = 1] \geq \alpha$$

On the other hand, the decisions of $\psi'$ are completely determined by $(P_i, X_i)_{i=1}^n$ and do not actually depend on the particular instantiations of $(P_i(a))_{a\in\mathcal{A},i\in[n]}$ and $(X_i(a))_{a\in\mathcal{A},i\in[n]}$. Thus, for each $a \in \mathcal{A}$,

$$\Pr[i \text{ is queried via } a \text{ by } \psi' \mid X_i(a) = 1] = \Pr[i \text{ is queried by } \psi \mid X_i = 1].$$

By combining the above equations, we have show that $\psi'$ is $\alpha$-selectable on the required input. The lemma is thus proven. □

## B.4 Proof of Lemma 3.6

We provide a sketch of the argument indicating how the main details proceed as in Lemma 2 of [BGMS24]. We focus on outlining why we may assume without loss that $p_1 = 1$ and $p_j \in \{0, 1\}$ for all $j \in [n] \setminus \{1\}$ when lower bounding (3.10). We omit the explanation for why we may assume $\sum_{i\in[n]} x_i = \ell$, as this can be attained at the very end, where we add additional elements $j$ with $p_j = 0$ and $x_j = 1$, and apply a simple coupling argument.

Begin with the arbitrary input $\mathcal{I}$, and an arbitrary element of $i \in [n] \setminus \{1\}$ with $0 < p_i < 1$, which we denote by $i = n$ for simplicity. We construct a new input $\widetilde{\mathcal{I}} = (n + 1, \ell, (\widetilde{x}_i)_{i=1}^{n+1}, (\widetilde{p}_i)_{i=1}^{n+1})$ with $\widetilde{p}_1 = \widetilde{p}_n = 1$, and an additional element $n + 1$ with $\widetilde{p}_{n+1} = 0$. Specifically,

1. $(\widetilde{p}_1, \widetilde{x}_1) = (1, p_1x_1)$.
2. $(\widetilde{p}_i, \widetilde{x}_i) = (p_i, x_i)$ for $i = 2, \ldots, n - 1$.
3. $(\widetilde{p}_n, \widetilde{x}_n) = (1, p_nx_n)$ and $(\widetilde{p}_{n+1}, \widetilde{x}_{n+1}) = (0, (1 - p_n)x_n)$

Observe first that since $\mathcal{I}$ is a feasible input, $\sum_{i=1}^{n+1} \widetilde{p}_i\widetilde{x}_i = p_1x_1 + p_nx_n + \sum_{i=2}^{n-1} p_ix_i \leq 1$, and $\sum_{i=1}^{n+1} \widetilde{x}_i = p_1x_1 + \sum_{i=2}^{n-1} x_i + p_nx_n + (1 - p_n)x_n \leq \sum_{i=1}^{n} x_i \leq \ell$, so $\widetilde{\mathcal{I}}$ is a feasible input. Let us thus define the random variables $(\widetilde{B}_i)_{i\in[n+1]}$, $(\widetilde{X}_i)_{i\in[n+1]}$, and $(\widetilde{Y}_i)_{i\in[n+1]}$ for $\widetilde{\mathcal{I}}$ in analogy to $(B_i)_{i\in[n]}$, $(X_i)_{i\in[n]}$, and $(Y_i)_{i\in[n]}$ for $\mathcal{I}$ (here $B_i \sim \mathrm{Ber}(b(p_ix_i))$ is the attenuation bit for element $i$ used by Algorithm 3). Our goal is to show that for all $y \in [0, 1]$,

$$\mathbb{P}[1 \text{ of } \mathcal{I} \text{ is queried} \mid Y_1 = y, X_1 = 1] \geq \mathbb{P}[1 \text{ of } \widetilde{\mathcal{I}} \text{ is queried} \mid \widetilde{Y}_1 = y, \widetilde{X}_1 = 1] \tag{B.5}$$

If we can establish (B.5), then we can iterate the argument a finite number of times until we're left with an input $\widetilde{\mathcal{I}}$ with $\widetilde{p}_1 = 1$, and where each other element $i$ has $\widetilde{p}_i \in \{0, 1\}$. Thus, for the remainder of the proof, we focus on explaining how (B.5) relates to Lemma 2 of [BGMS24].

Denote $\mathsf{safe}_1(\mathcal{I})$ as the event that 1 is safe during the execution on input $\mathcal{I}$, and recall that $\mathbb{P}[1 \text{ of } \mathcal{I} \text{ is queried} \mid Y_1 = y, X_1 = 1] = b(p_1 x_1)\mathbb{P}[\mathsf{safe}_1(\mathcal{I}) \mid Y_1 = y]$. Now, $p_1 x_1 = \widetilde{x}_1$, and so $b(p_1 x_1) = b(\widetilde{x}_1)$. Thus, in order to establish (B.5), it suffices to prove that

$$\mathbb{P}[\mathsf{safe}_1(\mathcal{I}) \mid Y_1 = y, X_1 = 1] \geq \mathbb{P}[\mathsf{safe}_1(\widetilde{\mathcal{I}}) \mid \widetilde{Y}_1 = y, \widetilde{X}_1 = 1].$$

This is exactly the statement proven in Lemma 2 of [BGMS24], and so we defer the remaining details of the argument. □

## B.5 Proof of Lemma 3.7

Given the input $\mathcal{I} = (n, \ell, (x_i)_{i=}^n, (p_i)_{i=1}^n)$, we first construct a new input $\widetilde{\mathcal{I}} = (n+1, \ell, (\widetilde{x}_i)_{i=1}^{n+1}, (\widetilde{p}_i)_{i=1}^{n+1})$, with an additional element $n+1$, and which modifies $\mathcal{I}$ in the following way:

1. $\widetilde{p}_n = 1$ and $\widetilde{x}_n := x_n/2$,
2. $\widetilde{p}_{n+1} = 1$ and $\widetilde{x}_{n+1} = x_n/2$,
3. $(\widetilde{x}_i, \widetilde{p}_i) = (x_i, p_i)$ for all $i \in [n-1]$.

Clearly, $\widetilde{\mathcal{I}}$ is a feasible input, which satisfies the conditions of Lemma 3.6. I.e., $\widetilde{p}_j \in \{0, 1\}$ for all $j \in [n+1]$, $\widetilde{p}_1 = 1$ and $\sum_{j \in [n+1]} \widetilde{x}_j = \ell$. For each $i \in [n+1]$, we draw $\widetilde{X}_i \sim \mathrm{Ber}(\widetilde{x}_i)$ independently. Our goal is to argue that

$$\mathbb{P}[1 \text{ from } \mathcal{I} \text{ is queried} \mid X_1 = 1] - \mathbb{P}[1 \text{ from } \widetilde{\mathcal{I}} \text{ is queried} \mid \widetilde{X}_1 = 1] \geq 0, \tag{B.6}$$

as this will complete the proof of Lemma 3.7. Now, recall that (3.11) ensures

$$\mathbb{P}[1 \text{ from } \mathcal{I} \text{ is queried} \mid X_1 = 1] = b(x_1) \int_0^1 \mathbb{P}[L_0 \leq \ell - 1 \mid Y_1 = y] \prod_{j \in N_1} (1 - y b(x_j) x_j) dy. \tag{B.7}$$

Thus, if we define $\widetilde{N}_0$, $\widetilde{N}_1$, $\widetilde{L}_0$, and $\widetilde{Y}_1$ in the analogous way as done for $N_0, N_1$, $L_0$, and $Y_1$ of $\mathcal{I}$, then observe that $\widetilde{x}_1 = x_1$, $\widetilde{N}_0 = N_0$, $\widetilde{N}_1 = N_1 \cup \{n+1\}$. More, we may assume that $\widetilde{X}_1 = X_1$ and $\widetilde{Y}_1 = Y_1$, and it is clear that $\mathbb{P}[\widetilde{L}_0 \leq \ell - 1 \mid Y_1 = y] = \mathbb{P}[L_0 \leq \ell - 1 \mid \widetilde{Y}_1 = y]$. Thus, using the same argument for deriving (3.11), we can write $\mathbb{P}[1 \text{ from } \widetilde{\mathcal{I}} \text{ is queried} \mid \widetilde{X}_1 = 1]$ as

$$\begin{aligned} &b(\widetilde{x}_1) \int_0^1 \mathbb{P}[\widetilde{L}_0 \leq \ell - 1 \mid \widetilde{Y}_1 = y] \prod_{j \in \widetilde{N}_1} (1 - y b(\widetilde{x}_j) \widetilde{x}_j) dy. \\ &= b(x_1) \int_0^1 (1 - y x_n b(x_n/2)/2)^2 \prod_{j \in N_1 \setminus \{n\}} (1 - y b(x_j) x_j) \mathbb{P}[L_0 \leq \ell - 1 \mid Y_1 = y] dy, \end{aligned} \tag{B.8}$$

where the second line follows from the aforementioned relations between $\mathcal{I}$ and $\widetilde{\mathcal{I}}$. Now, applying (B.7) and (B.8), we can rewrite the left-hand side of (B.6) as

$$b(x_1) \int_0^1 \left( (1 - y x_n b(x_n)) - (1 - y x_n b(x_n/2)/2)^2 \right) \prod_{j \in N_1 \setminus \{n\}} (1 - y b(x_j) x_j) \mathbb{P}[L_0 \leq \ell - 1 \mid Y_1 = y] dy \tag{B.9}$$

The final property of Proposition 3.8 ensures that there exists $y_{x_n} \in [0,1]$ such that the function $y \to (1 - yx_n b(x_n)) - (1 - yx_n b(x_n/2)/2)^2$ is non-negative for $y \in [0, y_{x_n}]$, and non-positive for $y \in (y_{x_n}, 1]$. Thus, since both $y \to \prod_{j \in N_1 \setminus \{n\}} (1 - yb(x_j)x_j)$ and $y \to \mathbb{P}[L_0 \leq \ell - 1 \mid Y_1 = y]$ are non-negative and non-increasing functions of $y$, we may conclude that

$$\int_0^1 \left((1 - yx_n b(x_n)) - (1 - yx_n b(x_n/2)/2)^2\right) \mathbb{P}[L_0 \leq \ell - 1 \mid Y_1 = y] \prod_{j \in N_1 \setminus \{n\}} (1 - yb(x_j)x_j) dy$$

$$\geq \mathbb{P}[L_0 \leq \ell - 1 \mid Y_1 = y_{x_n}] \prod_{j \in N_1 \setminus \{n\}} (1 - y_{x_n} b(x_j)x_j) \int_0^1 \left((1 - yx_n b(x_n)) - (1 - yx_n b(x_n/2)/2)^2\right) dy.$$

This lower bound is non-negative, as $\int_0^1 \left((1 - yx_n b(x_n)) - (1 - yx_n b(x_n/2)/2)^2\right) dy \geq 0$, by the final property of Proposition 3.8. Thus, since $b(x_1) \geq 0$, (B.6) holds, and so the proof is complete. □

### B.6 Proof of Proposition 3.8

The first property follows immediately from the definition of $b$. In order to see the the final two property, consider the function

$$f(y) := (1 - yzb(z)) - (1 - yzb(z/2)/2)^2 = -y^2 z^2 b(z/2)^2/4 + y(zb(z/2) - zb(z)).$$

Now,

$$\int_0^1 f(y)dy = (zb(z/2) - zb(z))/2 - z^2 b(z/2)^2/12, \tag{B.10}$$

and for our choice of $b$, we verify that (B.10) is non-negative for all $z \in [0,1]$. To see the final property, clearly $f$ is a quadratic function in $y$, and it is easy to see that its roots are 0 and $4(b(z/2) - b(z))/zb(z/2)^2$. Now, since $b$ is decreasing, $b(z/2) - b(z) \geq 0$, and so the latter root is non-negative. Finally, the leading term is $-z^2 b(z/2)^2/4$, which is negative. Thus, $f(y) \geq 0$ for $y \in [0, 4(b(z/2) - b(z))/zb(z/2)^2]$, and $f(y) \leq 0$ for $y \in [0, 4(b(z/2) - b(z))/zb(z/2)^2]$. □

## C Details of Section 4

### C.1 Proof of Proposition 4.1

Given $\varepsilon > 0$, consider the *relaxed dual*:

$$\text{minimize} \qquad \sum_{u \in U} \alpha_u + \sum_{u \in U} \ell_u \gamma_u + \sum_{v \in V} \beta_v \qquad \text{(r-D-LP-c)}$$

$$\text{s.t.} \qquad \beta_v \geq (1 - \varepsilon)\text{val}_{\boldsymbol{r},\boldsymbol{q}}(\boldsymbol{e}, \boldsymbol{a}) - \sum_{i=1}^{|\boldsymbol{e}|} (q_{e_i}(a_i)\alpha_{u_i} + \gamma_{u_i}) \cdot \prod_{j<i} (1 - q_{e_j}(a_j)) \qquad \forall v \in V, (\boldsymbol{e}, \boldsymbol{a}) \in \mathcal{C}_v :$$

$$e_i = (u_i, v) \quad \forall i \in [|\boldsymbol{e}|]$$

$$\alpha_u, \gamma_u \geq 0 \qquad \forall u \in U$$

$$\beta_v \geq 0 \qquad \forall v \in V$$

(Here the only difference from D-LP-c is that in the constraint corresponding to $v \in V$ and $(\boldsymbol{e}, \boldsymbol{a})$, the term $\text{val}_{\boldsymbol{r},\boldsymbol{q}}(\boldsymbol{e}, \boldsymbol{a})$ is multiplied by $(1 - \varepsilon)$). Now, a *strong approximate separation oracle* is presented an arbitrary selection of dual variables $((\alpha_u)_{u \in U}, (\gamma_u)_{u \in U}, (\beta_v)_{v \in V})$, and does one of the following:

1. Asserts that $((\alpha_u)_{u\in U}, (\gamma_u)_{u\in U}, (\beta_v)_{v\in V})$ is a feasible solution to r-D-LP-c.

2. Outputs a constraint of D-LP-c violated by $((\alpha_u)_{u\in U}, (\gamma_u)_{u\in U}, (\beta_v)_{v\in V})$.

Observe that if $\varepsilon = 0$, then this is the usual definition of a strong separation oracle defined in Chapter 4 of [KV12]. It is shown in Theorem 5.1 of [Jan03] that if one can provide a strong approximate separation oracle, then the ellipsoid algorithm can be used to efficiently compute a solution to LP-C with value at least $(1-\varepsilon)\mathrm{LP}(G)$. More, its runtime will be its usual runtime, multiplied by the runtime of the strong approximate separation oracle.

For the remainder of the section, we explain how to establish 1. or 2. for $((\alpha_u)_{u\in U}, (\gamma_u)_{u\in U}, (\beta_v)_{v\in V})$ under the assumptions of Proposition 4.1.

Observe first that we can efficiently check whether $((\alpha_u)_{u\in U}, (\gamma_u)_{u\in U}, (\beta_v)_{v\in V})$ are non-negative (and indicate if this is false as required by 2.). Thus, we assume that $((\alpha_u)_{u\in U}, (\gamma_u)_{u\in U}, (\beta_v)_{v\in V})$ are non-negative in what follows. Now, recall the edge rewards $\hat{\boldsymbol{r}} = (r_{u,v})_{u\in U, v\in V}$ from (4.2), whose definition ensures that for all $v \in V$ and $(\boldsymbol{e}, \boldsymbol{a}) \in \mathcal{C}_v$ where $e_i = (u_i, v)$ for $i = 1, \ldots, |\boldsymbol{e}|$,

$$\mathrm{val}_{\hat{\boldsymbol{r}},\boldsymbol{q}}(\boldsymbol{e}, \boldsymbol{a}) = \mathrm{val}_{\boldsymbol{r},\boldsymbol{q}}(\boldsymbol{e}, \boldsymbol{a}) - \sum_{i=1}^{|\boldsymbol{e}|} (q_{e_i}(a_i)\alpha_{u_i} + \gamma_{u_i}) \cdot \prod_{j<i} (1 - q_{e_j}(a_j)). \tag{C.1}$$

As in the assumption of Proposition 4.1, suppose that for each $v \in V$, we can efficiently compute $(\boldsymbol{e}^{(v)}, \boldsymbol{a}^{(v)}) \in \mathcal{C}_v$ such that

$$\mathrm{val}_{\hat{\boldsymbol{r}},\boldsymbol{q}}(\boldsymbol{e}^{(v)}, \boldsymbol{a}^{(v)}) \geq (1-\varepsilon) \max_{(\boldsymbol{e},\boldsymbol{a})\in\mathcal{C}_v} \mathrm{val}_{\hat{\boldsymbol{r}},\boldsymbol{q}}(\boldsymbol{e}, \boldsymbol{a}). \tag{C.2}$$

There are two cases to consider. First, suppose that $\mathrm{val}_{\hat{\boldsymbol{r}},\boldsymbol{q}}(\boldsymbol{e}^{(v)}, \boldsymbol{a}^{(v)}) > \beta_v$ for some $v \in V$. In this case, by (C.1), the constraint of D-LP-c corresponding to $v$ and $(\boldsymbol{e}^{(v)}, \boldsymbol{a}^{(v)}) \in \mathcal{C}_v$ is violated. Thus, our strong approximate separation oracle may output this constraint and satisfy 2. On the other hand, suppose that $\mathrm{val}_{\hat{\boldsymbol{r}},\boldsymbol{q}}(\boldsymbol{e}^{(v)}, \boldsymbol{a}^{(v)}) \leq \beta_v$ for all $v \in V$. Then, by (C.2), we know that for all $v \in V$ and $(\boldsymbol{e}, \boldsymbol{a}) \in \mathcal{C}_v$, where $e_i = (u_i, v)$ for $i = 1, \ldots, |\boldsymbol{e}|$:

$$\begin{aligned}
\beta_v &\geq (1-\varepsilon)\mathrm{val}_{\hat{\boldsymbol{r}},\boldsymbol{q}}(\boldsymbol{e}, \boldsymbol{a}) \\
&= (1-\varepsilon)(\mathrm{val}_{\boldsymbol{r},\boldsymbol{q}}(\boldsymbol{e}, \boldsymbol{a}) - \sum_{i=1}^{|\boldsymbol{e}|} (q_{e_i}(a_i)\alpha_{u_i} + \gamma_{u_i}) \cdot \prod_{j<i} (1 - q_{e_j}(a_j))) \\
&\geq (1-\varepsilon)\mathrm{val}_{\boldsymbol{r},\boldsymbol{q}}(\boldsymbol{e}, \boldsymbol{a}) - \sum_{i=1}^{|\boldsymbol{e}|} (q_{e_i}(a_i)\alpha_{u_i} + \gamma_{u_i}) \cdot \prod_{j<i} (1 - q_{e_j}(a_j))).
\end{aligned} \tag{C.3}$$

Here the first equality follows by (C.1), and the final inequality follows since $((\alpha_u)_{u\in U}, (\gamma_u)_{u\in U}, (\beta_v)_{v\in V})$ are non-negative. As such, by (C.3), in this case our approximate separation oracle may assert that $((\alpha_u)_{u\in U}, (\gamma_u)_{u\in U}, (\beta_v)_{v\in V})$ is a feasible solution to r-D-LP-c, and thus satisfy 1. □

## C.2 Proof of Lemma 4.5

To establish the feasibility of IP-TH, we construct an assignment that satisfies all the constraints. For each bucket $B_j$ that is a jump bucket, there is at most one edge $e_t^*$ assigned to it. Set $\xi^*_{je_t^*} = 1$ and $\xi^*_{ji} = 0$ for all other $i$. Then for each stable bucket $B_j$, if it consists of edges $e_t^*, ..., e_{t+h}^*$, we set $\xi^*_{je_t^*}, ..., \xi^*_{je_{t+h}^*} = 1$ and $\xi^*_{ji} = 0$ for all other $i$. These are all binary (*constraint* (4.14)), each job is assigned to at most one machine (*constraint* (4.15), each jump machine gets at most 1 job

assigned to it (*constraint* (4.16)), and at most $\ell$ jobs are assigned to machines (*constraint*(4.18)). For *constraint* (4.17) we consider 2 cases:

*Case 1*: Jump Buckets: for a jump bucket $B_j$, let the edge assigned to it to be $e_p^*$. Let $a_{opt}$ be the optimal action for $e_p^*$ that maximizes the expected future value. By definition, we have:

$$\begin{aligned} R^*_{e^*_p} - R^*_{e^*_{p+1}} &= r_{e^*_p}(a_{opt}) \cdot q_{e^*_p}(a_{opt}) - R^*_{e^*_{p+1}} \cdot q_{e^*_p}(a_{opt}) \\ &= (r_{e^*_p}(a_{opt}) - R^*_{e^*_{p+1}}) \cdot q_{e^*_p}(a_{opt}) \\ = (r_{e^*_p}(a_{opt}) - \text{BaseVal}(B_j)) \cdot q_{e^*_p}(a_{opt}) &= \max_{a \in \mathcal{A}} \Big\{ (r_{e^*_p}(a) - \text{BaseVal}(B_j)) \cdot q_{e^*_p}(a)] \Big\} \end{aligned}$$

Then we have:

$$\begin{aligned} &\sum_{j=1}^{n} \max_{a \in \mathcal{A}} \Big\{ (r_{e^*_p}(a) - \text{BaseGuess}(B_i)) \cdot q_j(a)] \Big\} \xi_{ij} \\ \geq \max_{a \in \mathcal{A}} \Big\{ (r_{e^*_p}(a) - \text{BaseGuess}(B_j)) \cdot q_{e_p}(a)] \Big\} &\geq \max_{a \in \mathcal{A}} \Big\{ (r_{e^*_p}(a) - \text{BaseVal}(B_j)) \cdot q_{e^*_p}(a)] \Big\} \\ = R^*_{e^*_p} - R^*_{e^*_{p+1}} &\geq \text{DeltaVal}(B_j) \geq \text{DeltaGuess}(B_j) \end{aligned}$$

*Case 2: Stable Buckets*: Let $B_i$ be a stable bucket containing consecutive positions $\{p, p+1, \ldots, p+y\}$ with corresponding buyers $\{e_p^*, \ldots, e_{p+y}^*\}$. For each position $z \in \{p, \ldots, p+y\}$, denote the optimal action by $a_z^*$.

The aggregate load contribution is:

$$\begin{aligned} &\sum_{z=p}^{p+y} \max_{a \in \mathcal{A}} \{ (r_{e^*_p}(a) - \text{BaseGuess}(B_i)) \cdot q_{e^*_z}(a) \} \\ &\geq \sum_{z=p}^{p+y} (r_{e^*_p}(a^*_z) - \text{BaseGuess}(B_i)) \cdot q_{e^*_z}(a^*_z) \end{aligned}$$

For each position $z$ in the stable bucket, the optimal action $a_z^*$ satisfies:

$$R^*_z - R^*_{z+1} = (r_{e^*_p}(a^*_z) - R^*_{z+1}) \cdot q_{e^*_z}(a^*_z)$$

Since all positions $q \in \{p, \ldots, p+y\}$ belong to the same stable bucket $B_i$, we have $R^*_{q+1} \geq \text{BaseVal}(B_i)$ for all such positions. Therefore, for each position $z$ in the stable bucket:

$$\begin{aligned} R^*_z - R^*_{z+1} &= (r_{e^*_p}(a^*_z) - R^*_{z+1}) \cdot q_{e^*_z}(a^*_z) \\ &\leq (r_{e^*_p}(a^*_z) - \text{BaseVal}(B_i)) \cdot q_{e^*_z}(a^*_z) \\ &\leq (r_{e^*_p}(a^*_z) - \text{BaseGuess}(B_i)) \cdot q_{e^*_z}(a^*_z) \end{aligned}$$

Summing over all positions in the stable bucket:

$$\begin{aligned} &\sum_{z=p}^{p+y} (r_{e^*_p}(a^*_z) - \text{BaseGuess}(B_i)) \cdot q_{e^*_z}(a^*_z) \\ &\geq \sum_{z=p}^{p+y} (R^*_z - R^*_{z+1}) \\ &= R^*_p - R^*_{p+y+1} \\ &= \text{DeltaVal}(B_i) \\ &\geq \text{DeltaGuess}(B_i) \end{aligned}$$

This completes the proof that *constraint* (4.17) is satisfied for all stable buckets, and thus IP-TH admits a feasible solution. □

### C.3 Proof of Lemma 4.7

By the recursive definition:

$$\tilde{R}_{i_{\min}} = \max_{a \in \mathcal{A}} \left\{ r_{e_{i_{\min}}}(a) \cdot q_{e_{i_{\min}}}(a) + \tilde{R}_{i_{\min}+1} \cdot (1 - q_{e_{i_{\min}}}(a)) \right\}$$

Since we are ahead of schedule at position $i_{\min}$, we have $\tilde{R}_{i_{\min}+1} \geq \text{BaseGuess}(B^{-1}_{e_{i_{\min}}})$. Therefore:

$$\begin{aligned}
\tilde{R}_{i_{\min}} &\geq \max_{a \in \mathcal{A}} \left\{ r_{e_{i_{\min}}}(a) \cdot q_{e_{i_{\min}}}(a) + \text{BaseGuess}(B^{-1}_{e_{i_{\min}}}) \cdot (1 - q_{e_{i_{\min}}}(a)) \right\} \\
&= \text{BaseGuess}(B^{-1}_{e_{i_{\min}}}) + \max_{a \in \mathcal{A}} \left\{ (r_{e_{i_{\min}}}(a) - \text{BaseGuess}(B^{-1}_{e_{i_{\min}}})) \cdot q_{e_{i_{\min}}}(a) \right\}
\end{aligned}$$

The term $\max_{a \in \mathcal{A}} \left\{ (r_{e_{i_{\min}}}(a) - \text{BaseGuess}(B^{-1}_{e_{i_{\min}}})) \cdot q_{e_{i_{\min}}}(a) \right\}$ is exactly the load contribution $c_{i_{\min}, e_{i_{\min}}}$ of buyer $e_{i_{\min}}$ to bucket $B^{-1}_{e_{i_{\min}}}$ in our Santa Claus formulation.

When $B^{-1}_{e_{i_{\min}}}$ is a jump bucket, our capacity constraints ensure that edge $e_{i_{\min}}$ is the only one assigned, so by Corollary 4.6:

$$c_{i_{\min}, e_{i_{\min}}} \geq (1 - \epsilon) \cdot \text{DeltaGuess}(B^{-1}_{e_{i_{\min}}})$$

When $B^{-1}_{e_{i_{\min}}}$ is a stable bucket, we have $\text{DeltaGuess}(B^{-1}_{e_{i_{\min}}}) \leq \epsilon \cdot \text{OPT}$ since this bucket does not contain any reward jump of size $\geq \epsilon \cdot \text{OPT}$. In either case:

$$\begin{aligned}
\tilde{R}_{i_{\min}} &\geq \text{BaseGuess}(B^{-1}_{e_{i_{\min}}}) + (1 - \epsilon) \cdot \text{DeltaGuess}(B^{-1}_{e_{i_{\min}}}) - \epsilon \cdot \text{OPT} \\
&\geq \text{BaseVal}(B^{-1}_{e_{i_{\min}}}) + (1 - \epsilon) \cdot \text{DeltaGuess}(B^{-1}_{e_{i_{\min}}}) - 2\epsilon \cdot \text{OPT}
\end{aligned}$$

□

### C.4 Proof of Lemma 4.8

For each $i < i_{\min}$, by the recursive definition:

$$\tilde{R}_i - \tilde{R}_{i+1} = \max_{a \in \mathcal{A}} \left\{ (r_{e_i}(a) - \tilde{R}_{i+1}) \cdot q_{e_i}(a) \right\}$$

Since we are behind schedule at position $i < i_{\min}$, we have $\tilde{R}_{i+1} < \text{BaseGuess}(B^{-1}_{e_i})$, so:

$$\tilde{R}_i - \tilde{R}_{i+1} \geq \max_{a \in \mathcal{A}} \left\{ (r_{e_i}(a) - \text{BaseGuess}(B^{-1}_{e_i})) \cdot q_{e_i}(a) \right\}$$

Summing over all edges assigned to bucket $j$:

$$\begin{aligned}
\sum_{i : B^{-1}_{e_i} = j} (\tilde{R}_i - \tilde{R}_{i+1}) &\geq \sum_{i : B^{-1}_{e_i} = j} \max_{a \in \mathcal{A}} \{ (r_{e_i}(a) - \text{BaseGuess}(B_j)) \cdot q_{e_i}(a) \} \\
&= \sum_{y : \xi_{jy} = 1} c_{jy} \\
&\geq (1 - \epsilon) \cdot \text{DeltaGuess}(B_j)
\end{aligned}$$

Thus:

$$\begin{aligned}
\sum_{i<i_{\min}} (\tilde{R}_i - \tilde{R}_{i+1}) &\geq \sum_{j>B^{-1}_{e_{i_{\min}}}} \sum_{i:B^{-1}_{e_i}=j} (\tilde{R}_i - \tilde{R}_{i+1}) \\
&\geq (1-\epsilon) \cdot \sum_{j>B^{-1}_{e_{i_{\min}}}} \text{DeltaGuess}(B_j) \\
&\geq (1-\epsilon) \cdot \sum_{j>B^{-1}_{e_{i_{\min}}}} (\text{BaseVal}(B_{j+1}) - \text{BaseVal}(B_j)) - \epsilon^2 \cdot \mathcal{E} \cdot (2K+1) \\
&\geq (1-\epsilon) \cdot \sum_{j>B^{-1}_{e_{i_{\min}}}} (\text{BaseVal}(B_{j+1}) - \text{BaseVal}(B_j)) - 3\epsilon \cdot \text{OPT},
\end{aligned}$$

where we've used that $K \leq 1/\epsilon$ and $\mathcal{E} \leq \text{OPT}$.