Welfare-Optimal Serial Dictatorships Have Polynomial Query Complexity

Ioannis Caragiannis[1], **Kurt Mehlhorn**[2,3], **Nidhi Rathi**[2,3]

[1]Aarhus University, Aarhus, Denmark
[2]Max Planck Institute for Informatics, Saarbrücken
[3]Saarland Informatics Campus, Germany
iannis@cs.au.dk, mehlhorn@mpi-inf.mpg.de, nrathi@mpi-inf.mpg.de

Abstract

Serial dictatorship is a simple mechanism for coordinating agents in solving combinatorial optimization problems according to their preferences. The most representative such problem is one-sided matching, in which a set of n agents have values for a set of n items, and the objective is to compute a matching of the agents to the items of maximum total value (a.k.a., *social welfare*). Following the recent framework of Caragiannis and Rathi (2024), we consider a model in which the agent-item values are not available upfront but become known by querying *agent sequences*. In particular, when the agents are asked to *act* in a sequence, they respond by picking their favorite item that has not been picked by agents who acted before and reveal their value for it. Can we compute an agent sequence that induces a social welfare-optimal matching?

We answer this question affirmatively and present an algorithm that uses polynomial number (specifically, $\mathcal{O}(n^5)$ queries). This solves the main open problem stated by Caragiannis and Rathi (2024). Our analysis uses a *potential function* argument that measures progress towards learning the underlying edge-weight information. Furthermore, the algorithm has a *truthful implementation* by adapting the paradigm of VCG payments.

1 Introduction

In this work, we explore the well-studied problem of finding maximum-weight matchings in complete bipartite graphs[1] through the lens of the *serial dictatorship* (SD) mechanism. A representative example of the SD mechanism can be seen when it is considered as a solution to the *house allocation* problem (e.g., see Abdulkadiroğlu and Sönmez (1998)), where a set of houses have to be matched to agents who have *strict* preferences for them. The algorithm considers the agents in a fixed order (or sequence) and assigns the most-preferred available house to each agent considered. The order in which agents come and pick (i.e., *act*) an item *greedily* is called an *action sequence*. Serial Dictatorship is arguably the simplest algorithm with applications in many varied settings such as one-sided matchings (Filos-Ratsikas, Frederiksen, and Zhang 2014), fair division (Svensson 1999), and assignment problems (Bogomolnaia and Moulin 2001); see Abdulkadiroğlu and Sönmez (1998); Caragiannis et al. (2016); Krysta et al. (2019a); Roth and Sotomayor (1990) for other examples where serial dictatorship (or a variation) is useful.

The simplicity of SD mechanism leads to its computational ease as well, and hence, it is easily comprehensible for the agents. It is *favorable* for every agent to choose their favorite available item on their turn (i.e., be greedy) since SD never considers any feedback from the agents other than what item they pick at their turn, thereby making SD a truthful mechanism as well.

It is relevant to note that the choice of the action sequence is crucial since a sub-optimal action sequence may lead to inefficiencies, i.e., sub-optimal *social welfare* (Caragiannis et al. 2016). For example, consider a scenario where two items a and b are to be assigned to two agents 1 and 2, under the constraint that each agent is assigned exactly one item. Agent 1 values item a at 9 and item b at 1, whereas agent 2 values item a at 10 and item b at 8. Let us execute SD with both sequences of the two agents. When agent 1 is followed by agent 2, we obtain an optimal social welfare of 17, while the reverse sequence yields a (sub-optimal) social welfare of only 11. Therefore, when applying the SD mechanism, the agent ordering plays a crucial role towards the quality of the social welfare achieved in assignment problems. Note that, uniformly choosing over the two action sequences—which is the main idea of *random serial dictatorship*—in the above example still leads to inefficiencies (Abdulkadiroğlu and Sönmez 1998; Bogomolnaia and Moulin 2001; Filos-Ratsikas, Frederiksen, and Zhang 2014; Krysta et al. 2019b).

There are many real-world scenarios where there is some kind of *first-come first-serve policy* at work. For example, consider a shopping mall during a sale or a fresh fish market in a city, where whoever arrives early gets the better pick of the day. Another scenario is when a government wishes to allocate homes to its citizens in a newly-built apartment housing. This housing complex may consist of different kinds of apartments, where some have balconies, some are east-facing and get good sunlight, or some have better views.

[1]A common way to think of the problem is to consider one side of the bipartition representing the n *agents* and the other side representing the n *items*, where the weight on an agent-item edge indicates the preference of that agent for the item.

Here, citizens come in an order (maybe according to their registration), and selects (an available) apartment that they like the most. Note that, the government may aim to find an ordering of the citizens in which they come and choose the apartments so that a higher collective social welfare can be achieved. This may become crucial during their future election campaigns.

Following the recent framework introduced by Caragiannis and Rathi (2024), we consider a model wherein agent-item preferences in a complete bipartite weighted graph are not known upfront to an algorithm, but instead, it *learns* them through *action sequence queries*. The goal is to find an action sequence that induces a (perfect) matching between agents and items with maximum possible weight i.e., highest social welfare. We call such an action sequence *welfare-optimal*. To illustrate the loss one may incur to the social welfare by using action sequences, Caragiannis and Rathi (2024) defined the concept of *price of serial dictatorship* (PoSD). They further proved an interesting structural property that *for complete bipartite weighted graphs, there always exists an action sequence that induces a maximum-weight matching*. Therefore, PoSD of maximum weight perfect matching in bipartite graphs is 1, i.e., any *welfare-optimal action sequence* generates a maximum-weight matching in bipartite weighted graphs[2]. Thus, the following question naturally arises:

> With query access to a complete bipartite weighted graph, can we compute a *welfare-optimal action sequence* using *polynomially-many* action sequence queries?

Note that the space of solutions, i.e., the space of all action sequences, is exponential and querying all the action sequences will solve the problem exactly. Whether this can be done much faster is the challenging question we address.

Our Results and Techniques: We answer the above question affirmatively and develop a novel algorithm (Algorithm 1) that takes input a query access to a graph $G = K_{n,n}$ equipped with agent-item preferences[3] and uses only $\mathcal{O}(n^5)$ action sequence queries to find a welfare-optimal action sequence, i.e., the action sequence that induces a maximum-weight matching in G; see Theorem 4.2. This answers the main open problem listed in (Caragiannis and Rathi 2024), as stated above in box 1. We have essentially shown that, even though there are $n!$ many action sequences, querying only $\mathcal{O}(n^5)$ many of them is enough to learn the necessary agent-item preferences and discover a welfare-optimal action sequence in G. Moreover, we show that if the edges incident to any agent have pairwise distinct weights, then Algorithm 1 requires $\mathcal{O}(n^4)$ many queries only; see Section 3.

Our algorithm (Algorithm 1) progressively learns the agent-item preferences and maintains a proxy graph $G' = K_{n,n}$ with proxy edge-weights and proxy agent ranking information over the items. Throughout the execution of our algorithm, we ensure that the proxy edge-weights are always an over-estimation of the true edge-weights (until the ground truth about an agent-item edge is learnt). With this useful invariant, we prove a structural property between G and G' and identify a *condition of welfare-optimality for* π, see Theorem 4.3. Our analysis uses a *potential-function* based argument that measures progress towards learning the underlying edge-weight and rank information. We prove that as soon as certain parameters in G' are satisfied (see Lemma 4.6), we have learnt every information about G that is necessary for learning a welfare-optimal action sequence in G.

It is relevant to note that the query model that we use in this work is very restrictive and that is where the importance of our work lies in. All that an algorithm under our model can do is, order the agents and then ask them, in turn, for their favorite item (and its value) among the remaining items. So the preferences of an agent are revealed in a piecemeal fashion. In particular, the algorithm cannot ask an agent for the values of all its incident edges. There are other combinatorial problems where the preferences of an agent are revealed in a piece-meal fashion, e.g., stable matchings (Gale and Shapley 1962).

Further Related Work: Serial dictatorship and its variations have been extensively employed in the matching literature (Manlove 2013). The notion of greedy weighted matchings[4] in (Deligkas, Mertzios, and Spirakis 2017) can be seen as optimizing over a restricted set of action sequences. Furthermore, Deligkas, Mertzios, and Spirakis (2017) study the computational complexity of the problem assuming that the graph information (i.e., the edge weights) is given to the algorithm upfront. The notion of picking sequence in fair division (Beynier et al. 2019; Bouveret and Lang 2011; Gourvès, Lesca, and Wilczynski 2021; Kalinowski, Narodytska, and Walsh 2013) is close to the action sequences that we consider here; the crucial difference is that each agent can appear several times in a picking sequence.

Optimizing over serial dictatorships can be thought of as a particular way of exploiting greediness in computation. There have been several attempts to formally define greedy algorithms, including their relations to matroids, which are covered in classical textbooks on algorithm design (Cormen et al. 2009). In relation to combinatorial optimization, the work on incremental priority algorithms (Borodin et al. 2010; Borodin, Nielsen, and Rackoff 2003; Davis and Impagliazzo 2004) has conceptual similarities. Furthermore, for maximum satisfiability, there is an ongoing research line (e.g., see Poloczek et al. (2017)) that aims to design simple greedy-like algorithms that achieve good approximation ratios. In the EconCS literature, cut, party affiliation, constraint satisfaction games (e.g., Bhalgat, Chakraborty, and Khanna (2010)) and boolean games (Wooldridge et al. 2013) are closest to ours among works that use boolean formulae to express logical relations between agents' actions. However, all these studies neglect query complexity questions.

[2]This result is also implied by the work of Abdulkadiroğlu and Sönmez (1998), if the bipartite graph consists of distinct weights.

[3]Every agent has a ranking over all items along-with weight on all the edges incident to her.

[4]A greedy weighted matching is produced when we start from an empty matching and iteratively augment it with an edge of maximum weight that is consistent with it.

2 Preliminaries

We consider a complete bipartite graph $G = K_{n,n}$ over n agents denoted by the set $[n] = \{1, 2, \ldots, n\}$ on the left side of the bipartition and n items on the other. The weight $w(i, j) \geq 0$ denotes the value agent i has for item j, and we represent it via a weight function $w : [n]^2 \to \mathbb{R}_{\geq 0}$. The problem is to assign these items to agents, under the restriction that each agent gets exactly one item. We call such an assignment of items to agents a *matching* in G. We denote a matching M as $\{(i, M(i))\}_{i \in [n]}$, where agent $i \in [n]$ is assigned item $M(i)$.

Each agent i also has a strict ranking r_i that ranks the items in monotone non-increasing order with respect to the value of agent i for them, breaking ties in a predefined (*agent-specific*) manner. We denote the rank of item j for agent i by $r_i(j)$. The items of rank 1 and n for agent i are ones for which she has highest and lowest values, respectively. Also, for any $j_1, j_2 \in [n]$, we have that $w(i, j_1) > w(i, j_2)$ implies $r_i(j_1) < r_i(j_2)$. However, $r_i(j_1) < r_i(j_2)$ implies only $w(i, j_1) \geq w(i, j_2)$.

We say a perfect matching M is *Pareto-optimal* (PO) if there is no other perfect matching M' such that $r_i(M'(i)) \leq r_i(M(i))$ for all $i \in [n]$ and $r_i(M'(i^*)) < r_i(M(i^*))$ for some agent $i^* \in [n]$.

An *action sequence* (respectively, *action subsequence*) S is an ordering $S = (S(1), S(2), \ldots)$ of the agents in $[n]$ (respectively, some of the agents in $[n]$). We write $\mathcal{S}$ to denote the set of all possible $n!$ action sequences and various subsequences. For an agent $i \in [n]$, we write $\mathcal{S}_{-i}$ to denote all possible action subsequences consisting of agents from the set $[n] \setminus \{i\}$. For an action sequence S, we write $S|_t = (S(1), \ldots, S(t))$ for its prefix of length t. For an integer $k \geq 1$, we use O_k to denote the ordered set $(1, 2, \ldots, k)$ of k agents. We reserve the letters S and π for use as action (sub)sequences.

Agents pick their items according to an action sequence. When it is the turn of an agent to act, she picks the item of highest value among those that have not been picked by agents who acted before her. For an action subsequence $S \in \mathcal{S}_{-i}$, denote by $\mu_i(S)$ the item agent i picks when she acts immediately after the agents in S. We use $\mu(S)$ to denote the set of items picked by the agents in S, i.e., $\mu(S) = \{\mu_{S(k)}(S|_{k-1}) : 1 \leq k \leq |S|\}$. Note that, we have $\mu_i(S) = \arg\min_{j \in [n] \setminus \mu(S)} r_i(j)$, which gives her value $v_i(S) = w(i, \mu_i(S))$. Hence, when all agents have acted, the items picked form a *perfect matching* in G.

For an action sequence $\pi \in \mathcal{S}$, we write $M(\pi; G)$ to denote the matching induced by π in G. Similarly, we write $\mathrm{SW}(\pi; G) = w(M(\pi; G))$ to denote the *social welfare* of an action sequence π in G. We call an action sequence *welfare-optimal* if it produces a matching with highest social welfare, i.e, $\max_{\pi \in \mathcal{S}} \mathrm{SW}(\pi; G)$. Note that any welfare-optimal action sequence induces a maximum-weight matching in the underlying graph G (Caragiannis and Rathi 2024).

An instance of OSM (*Optimal Sequence for Matchings*) consists of a graph $G = K_{n,n}$ equipped with the weight function w and rank information $\{r_i\}_i$ and the objective is to compute an action sequence of maximum social welfare using the following query oracle access. For an agent $i \in [n]$ and a (sub)sequence $S \in \mathcal{S}_{-i}$ of agents, the $\mathsf{Query}(i, S)$ returns a tuple (v, t), where $t = \mu_i(S)$ is the item agent i picks when she acts immediately after the agents in S and v is the value she obtains i.e., $v = v_i(S) = w(i, t)$.

An important restriction of our model is that the weight and rank information is not given to the algorithm as part of the input. Instead, the algorithms for solving OSM can learn them by making queries. Note that it is possible that our algorithm may not learn all the $\mathcal{O}(n^2)$ different values using our query model.

Comparison with other query models: One natural query model is to ask an agent $i \in [n]$ for her value for an item $j \in [n]$. Another query model will be to ask an agent $i \in [n]$ for the value of her favorite item in a subset $S \subseteq [n]$ of items. Note that, in both these models, one can recover all the n^2 valuations in n^2 queries, making the query-complexity of the problem trivial.

In contrast, our model is much more restrictive. Consider an OSM instance where no agent's top choice collides with any other agent's top choice. Hence, *any* action sequence reveals only single valuation entry per agent. But, note that, it is not even required to learn rest of the valuation entries, since each action sequence is optimal in this instance.

We remark that our aim is to solve OSM using a polynomial number of queries and hence, understand its *query complexity*. Computational limitations are not our concern in this work and we assume to have unlimited computational resources to process the valuations once these are available.

3 Some Intuition

Before presenting our algorithm, let us give some intuition. For simplicity, we assume in this section that the edges incident to any agent have pairwise distinct weights and defer the treatment of equal weight edges (or ties) to Section 4. For each edge $e \in [n] \times [n]$, we maintain an upper bound $w'(e)$ on the true weight $w(e)$. We initialize $w'(e)$ to ∞ for all e. We also maintain a (growing) set E^* of edges for which we already know the true weight, i.e, $e \in E^*$ implies $w'(e) = w(e)$. Initially, E^* is the empty set. We progressively learn the agent-item preferences and maintain a proxy graph $G' = K_{n,n}$ with proxy edge-weight functions w'.

Let M' be a maximum weight matching with respect to weight function w'. It was shown by Caragiannis and Rathi (2024) that there is an action sequence π constructing M' in graph G'.[5]

We execute π until a query returns an edge $e = (i, j)$ with value $v = w(e)$ that does not belong to $E^* \cap M'$ or

[5] In a maximum weight matching M', there is always an agent that is incident to its most valuable edge. Assume otherwise and let B consist of the most valuable edge for each agent. Then $M' \cup B$ contains an improving alternating cycle. The first query is towards an agent that is incident to its most valuable edge in M'. Then, we continue by induction.
The preceding paragraph glosses over the detail that there might be equal weight edges with respect to w'; see the full version for a detailed description of the above protocol, that we call as MWPO (standing for *maximum-weight Pareto-optimal matching*).

until π is completely executed. If $(i,j) \notin E^*$, we add it to E^* and set $w'(e)$ to v. Note that $w'(e) = w(e)$ in this case. If $(i,j) \in E^* \setminus M'$, we decrease $w'(i, M'(i))$ to v. Note that when the query returns (i,j), the edge $(i, M'(i))$ is also available and since the query returns the highest-weight available edge for agent i, v must be (strictly) smaller than $w'(i, M'(i))$.

We continue until π is completely executed. Then $M' \subseteq E^*$ and hence M' is a maximum weight matching. Also π constructs M'.

How can we bound the number of queries? We can add to E^* at most n^2 times. Also, the w'-value of an edge incident to i is always the w-value of some edge incident to i or infinity. So the w' value of any edge can change at most n times. Identifying each such update requires at most n queries, and hence, with a total of $\mathcal{O}(n^4)$ queries, we can find the welfare-optimal action sequence in the given instance.

The treatment of equal weights will considerably complicate matters, as we see in Section 4.

4 Finding an Optimal Action Sequence

We begin by stating a structural property about maximum-weight matchings proved by Caragiannis and Rathi (2024).

Theorem 4.1. *(Caragiannis and Rathi 2024) Any welfare-optimal action sequence induces a maximum-weight matching in a complete bipartite weighted graph.*

We develop a query-efficient algorithm (Algorithm 1) to compute a welfare-optimal action sequence for OSM instances. We state our main result here and prove it towards the end of this section.

Theorem 4.2. *For n-agent* OSM *instances with query access, Algorithm 1 uses $\mathcal{O}(n^5)$ queries to compute a welfare-optimal action sequence.*

Our algorithm and its analysis is based on a potential-function based argument. We begin by stating a crucial property (in Theorem 4.3) that connects maximum-weight matchings to action sequences. This property works at the core of (the correctness of) our algorithm; we re-write and prove it in Lemmas 4.4 and 4.6.

Theorem 4.3. *Consider two complete weighted bipartite graphs $G, G' = K_{n,n}$ with weight functions w, w' and rank functions $\{r_i\}_i, \{r'_i\}_i$ respectively. Let $E^* \subseteq [n]^2$ be a subset of edges, admitting a perfect matching, such that*

$$w'(e) = \begin{cases} = w(e), & \text{if } e \in E^* \\ \geq w(e), & \text{otherwise} \end{cases}$$

and such that $w'(i,j_1) > w'(i,j_2)$ implies $r'_i(j_1) < r'_i(j_2)$ for all $j_1, j_2 \in [n]$.

Let M^* be a maximum-weight matching using the edges in E^*, $M' \in G'$ be a maximum-weight PO matching, and π be the (welfare-optimal) action sequence corresponding to M' in G'. Then, there exists a pair (M', π) such that $\mathrm{SW}(\pi; G) = w'(M') = w'(M^*)$, and this π is welfare-optimal in G as well.

We will often refer to G' as the proxy graph (to G) with proxy weight w' and rank functions r'_i's. On the other hand, we will refer to w and r_i's (of G) as the true weight and rank functions, respectively.

Overview of our algorithm: On input query access to an n-agent OSM instance over $G = K_{n,n}$, Algorithm 1 makes queries of the form $\mathsf{Query}(i, S)$ to an agent $i \in [n]$ for an action (sub)sequence $S \in \mathcal{S}_{-i}$ in an attempt to learn the true weight and rank functions. It progressively learns the agent-item preferences and maintains a proxy graph $G' = K_{n,n}$ with proxy edge-weight function w' and proxy agent rank information r'_i's. We give an overview of Algorithm 1 in the following.

- Phase 1 and Phase 2 of Algorithm 1 can be thought of as pre-processing that learns the bare minimum information to get started. Phase 1 learns the most-favorite items for every agent and Phase 2 executes the action sequence $(1, 2, \ldots, n)$ so as to ensure that the set of edges whose weights are learnt contains a perfect matching.
- We maintain a set E^*, that contains all the edges whose weights are learnt during the execution of Algorithm 1. We maintain a maximum-weight perfect matching $M^* \in G$ that only uses the edges in E^* and denote its weight by $w'(M^*)$.
- We maintain a proxy graph $G' = K_{n,n}$ with rank information $\{r'_i\}_{i \in [n]}$ and edge-weights $\{w'(i,j)\}_{i,j}$ such that proxy edge-weights are always an over-estimation of the true edge-weights; see Lemma 4.4.
- We run the protocol $\mathsf{MWPO}(\mathsf{G}')$ that outputs a maximum-weight PO matching M' in G' and its corresponding action sequence π, i.e., $M(\pi; G') = M'$. We write $w'(M')$ to denote the weight of matching M' in G'; see Footnote 5 and the full version for more details.
- In the beginning, we set $w'(i,j) = v_i(\emptyset) = \max_{j \in [n]} w(i,j)$ for all $i, j \in [n]$. Hence, at this point, we have $w'(M') = \sum_{i \in [n]} v_i(\emptyset)$, making $w'(M') \geq w'(M^*)$.
- As long as we have $w'(M') \geq w'(M^*)$, we simply run the action sequence π (corresponding to M' in G') in the original instance using the query oracle. This is performed by executing the while-loop in lines 20-40 of Algorithm 1. The queries made during the execution of this while-loop lead to new information, and hence, we update the set E^* and graph G' accordingly; see Lemma 4.7 and 4.8. This, in turn, changes M^*, M', and π as well.
- We obtain $\mathrm{SW}(\pi; G)$ by querying the agents according to the sequence π, we denote it in the algorithm as $\mathsf{Query}(\mathrm{SW}(\pi; G))$. We prove that as soon as we find a matching $M' \in G'$ such that $\mathrm{SW}(\pi; G) = w'(M') = w'(M^*)$, then π must be welfare-optimal for G as well; see Lemma 4.6. We call this as a *condition of welfare-optimality for π*.
- The careful updates performed by our algorithm enable us to produce a *potential-function* based argument to measure the *progress* towards learning the underlying information of G. It is dependent on the size $|E^*|$, distance $\sum_{i,j \in [n]} |w'(i,j) - w(i,j)|$ between w' and w, and distance $\sum_{i,j \in [n]} |r'_i(j) - r_i(j)|$ between r' and r.

  We prove that Algorithm 1 uses $\mathcal{O}(n^5)$ queries to find a welfare-optimal action sequence in G; see Theorem 4.2.

When we execute the welfare-optimal action sequence π (corresponding to a maximum-weight matching M' in G') in G using the query oracle, it guides us so that we learn new information and progressively become closer to the ground truth of G. The goal is to make G' so close to G such that a welfare-optimal action sequence in G' also becomes welfare-optimal in G; see Theorem 4.3.

Throughout this section, we will adhere to the following notations. At the beginning of an arbitrary iteration of the while-loop in lines 20-40 during the execution of Algorithm 1, we write (i) E^* to denote the set of known edges, (ii) M^* to denote a maximum-weight matching using the edges in E^*, with weight $w'(M^*)$, (iii) $M' \in G'$ to denote a maximum-weight PO matching with weight $w'(M')$, and (iv) π to denote the action sequence corresponding to M' in G', i.e., $M(\pi; G') = M'$.

From now on, when we mention the while-loop, we would mean the while-loop in lines 20-40 of Algorithm 1. We begin our analysis with Lemma 4.4 where we prove that the proxy edge-weights in G' are always an over-estimation of the true edge-weights in G, and they coincide for the edges that are already learnt, i.e., for the edges in the set E^*.

Lemma 4.4. *On input n-agent* OSM *instances, Algorithm 1 maintains the following invariant on the proxy edge-weights throughout its execution: for any edge $e \in [n]^2$,*

$$\textit{the proxy edge-weight, } w'(e) = \begin{cases} = w(e), & \textit{if } e \in E^* \\ \geq w(e), & \textit{otherwise} \end{cases}$$

Moreover, we have $w'(i, j_1) > w'(i, j_2)$ implies $r'_i(j_1) < r'_i(j_2)$ for all $j_1, j_2 \in [n]$.

Proof. Fix an edge $e = (i, j) \in [n]^2$. First, note that the stated claim about rank information is true by construction; see line 37.

Now, observe that, whenever e is returned in a query during the execution of Algorithm 1, we set $w'(e) = w(e)$ and add e to the set E^*; see lines 6, 11, and 31.

Therefore, to complete our proof, all we need to show is that $w'(e) \geq w(e)$ if $e \notin E^*$ throughout the execution Algorithm 1. Note that, it holds true in the beginning since $w'(i, j)$ is set to be $v_i(\emptyset) = \max_{j \in [n]} w(i, j)$ in line 6. The only other time when $w'(e)$ is updated even when e is not returned in a query is in line 33. We consider this particular iteration of the while-loop with an action sequence, say π. Recall that π induces the maximum-weight matching $M' \in G'$ and the while-loop queries agents according to the ordering in π (i.e., it runs π in G using the query oracle).

Now, when we make the query for agent i in accordance to π, assume that edge $\bar{e} = (i, \bar{j}) \in E^* \setminus M'$ is returned instead of $e = (i, j) = (i, M'(i)) \in M'$. Note that, we break from an iteration of the repeat-loop as soon as any of the three else-if conditions are met in lines 30, 32, or 34 (where the edge that is returned is either not in E^* or not in M'). Therefore, in this case, if the considered iteration of the while-loop did not break and proceeded to reach agent i, we know that all the preceding queries returned for π must be edges in M'. Therefore, e and $\bar{e}$ are still available for agent i on her turn

Algorithm 1: An efficient algorithm to find welfare-optimal action sequence for OSM

Input: Query access to an n-agent OSM instance over a weighted graph $G = K_{n,n}$
Output: An action sequence π

1: Initialize $w'(i, j) \leftarrow \infty$ for every $i, j \in [n]$, and a set $E^* \leftarrow \emptyset$;
2: Initialize $r'_i := [r'_i(1), r'_i(2), \ldots, r'_i(n)] \leftarrow [1, 2, \ldots, n]$ for every $i \in [n]$;
3: $G' \leftarrow K_{n,n}$ with edge weights $\{w'(i, j)\}_{i,j}$ and rank functions $\{r'_i\}_i$;
-Phase 1: Learning the most-favorite item for every agent-
4: **for** $i \leftarrow 1$ to n **do**
5: $(v, t) \leftarrow \mathsf{Query}(i, \emptyset)$;
6: $w'(i, j) \leftarrow v$ for all $j \in [n]$; $E^* \leftarrow E^* \cup \{(i, t)\}$;
7: Swap $r'_i(t)$ and $r'_i(1)$;
8: **end for**
-Phase 2: Ensuring the existence of a perfect matching in E^*-
9: **for** $i \leftarrow 2$ to n **do**
10: $(v, t) \leftarrow \mathsf{Query}(i, O_{i-1})$;
11: $w'(i, t) \leftarrow v$; $E^* \leftarrow E^* \cup \{(i, t)\}$;
12: **if** $v = w'(i, 1)$ and $t \neq (r'_i)^{-1}(1)$ **then**
13: swap $r'_i(t)$ and $r'_i(2)$;
14: **else if** $v \neq w'(i, 1)$ **then**
15: swap $r'_i(t)$ and $r'_i(n)$;
16: **end if**
17: **end for**
18: $M^* \leftarrow$ a max-weight matching using the edges in E^*;
19: $(M', \pi) \leftarrow \mathsf{MWPO}(G')$;
-Phase 3: Executing π in G using the query oracle-
20: **while** $w'(M') \geq w'(M^*)$ **do**
21: **if** $\mathsf{Query}(\mathrm{SW}(\pi)) = w'(M') = w'(M^*)$ **then**
22: **return** π;
23: **end if**
24: Set $(a_1, a_2, \ldots, a_n) \leftarrow (\pi(1), \pi(2), \ldots, \pi(n))$;
25: $i \leftarrow 0$;
26: **repeat**
27: $i \leftarrow i + 1$;
28: $(v, t) \leftarrow \mathsf{Query}(a_i, \pi|_{i-1})$;
29: **until** edge $(a_i, t) \notin E^* \cap M'$
30: **if** edge $(a_i, t) \notin E^*$ **then**
31: $E^* \leftarrow E^* \cup \{(a_i, t)\}$; $w'(a_i, t) \leftarrow v$;
32: **else if** edge $(a_i, t) \in E^* \setminus M'$ and $v < w'(a_i, M'(a_i))$ **then**
33: $w'(a_i, M'(a_i)) \leftarrow v$;
34: **else if** edge $(a_i, t) \in E^* \setminus M'$ and $v = w'(a_i, M'(a_i))$ **then**
35: swap $r'_{a_i}(t)$ and $r'_{a_i}(M'(a_i))$;
36: **end if**
37: Update r'_i such that $w'(i, j_1) > w'(i, j_2)$ implies $r'_i(j_1) < r'_{a_i}(j_2)$ $\forall\, i, j_1, j_2 \in [n]$, while being consistent with Line 35;
38: $M^* \leftarrow$a max-weight matching using the edges in E^*;
39: $(M', \pi) \leftarrow \mathsf{MWPO}(G')$;
40: **end while**

in π. Since, π is an action sequence corresponding to M' in G', we have

$$w'(i,\bar{j}) < w'(i,j) \text{ or, } (w'(i,\bar{j}) = w'(i,j) \;\&\; r_i(j) < r_i(\bar{j}))$$

Since, $\bar{e}$ (with query value $v = w(i,\bar{j})$) is returned instead of e, we have

$$w(i,j) < w(i,\bar{j}) \text{ or, } (w(i,\bar{j}) = w(i,j) \text{ and } r_i(\bar{j}) < r_i(j))$$

And since, $\bar{e} \in E^*$ we know $w'(i,\bar{j}) = w(i,\bar{j})$. In either case, we have

$$w(i,j) \leq v = w(i,\bar{j}) = w'(i,\bar{j}) \leq w'(i,j) \qquad (1)$$

If $v < w'(i,j)$, we decrease $w'(i,j)$ to v (see Step 33) and if $v = w'(i,j)$, then we swap $r_i(j)$ and $r_i(\bar{j})$ (see Step 35). In either case, the invariant $w(i,j) \leq w'(i,j)$ is maintained and we move closer to the true weight value $w(i,j)$ for the edge (i,j). This completes our proof. □

Corollary 4.5. *For each iteration of the* while-loop *in lines 20-40, we have* $w'(M^*) \leq w'(M')$ *and* $\mathrm{SW}(\pi;G) \leq w'(M')$.

Proof. Lemma 4.4 shows that the proxy weights are always an over-estimation of the true weights. In any iteration of the while-loop, recall that M' denotes a maximum-weight matching in G' while M^* denotes a maximum-weight matching using only a subset of edges of G', i.e., E^*. This implies that we have $w'(M^*) \leq w'(M')$. Moreover, since π induces M' in G', the invariant on the proxy weights in Lemma 4.4 ensures that the social welfare of π in G cannot be greater than $w'(M')$, i.e., $\mathrm{SW}(\pi;G) \leq w'(M')$. □

Note that we obtain $\mathrm{SW}(\pi;G)$ by making n queries to the agents according to the sequence π, we denote it in the algorithm as $\mathsf{Query}(\mathrm{SW}(\pi;G))$; see line 21.

The next result formally states how close do we need to make G' to G so that a welfare-optimal action sequence in G' becomes welfare-optimal in G as well, i.e., we state and prove a *condition of welfare-optimality*.

Lemma 4.6. *On input* n*-agent* OSM *instances, when Algorithm 1 finds a perfect matching* $M' \in G'$ *with* $\mathrm{SW}(\pi;G) = w'(M') = w'(M^*)$*, then* π *must be welfare-optimal in* G.

Proof. According to Corollary 4.5, we know that $w'(M^*) \leq w'(M')$ and $\mathrm{SW}(\pi;G) \leq w'(M')$ hold true for any iteration of the while-loop. Recall that M^* is a maximum-weight matching using the known edges in E^* and w' is an over-estimation of the true weight function w (Lemma 4.4). Therefore, when our algorithm finds two matchings M' and M^* such that $w'(M') = w'(M^*)$, then this value must equal the maximum social welfare value in G.

Now, once we know the maximum social welfare value in G, the task is to ascertain the action sequence that achieves this value in G. Recall that π induces M' in G'. If π induces a matching in G such that $\mathrm{SW}(\pi;G) = w'(M') = w'(M^*)$, then the action sequence π achieves the maximum social welfare in G, making π welfare-optimal in G as well. □

Next, in Lemma 4.7, we analyse all possible events that can occur during the execution of the while-loop in lines 20-40 of Algorithm 1. Our careful choice of updates in these events enables us to develop a potential function to measure the progress of Algorithm 1; see Lemma 4.8.

Lemma 4.7. *Given an* n*-agent* OSM *instance, exactly one of the following conditions is satisfied for any iteration of the* while-loop *in lines 20-40 of Algorithm 1:*
(C1) $\mathrm{SW}(\pi;G) = w'(M') = w'(M^*)$*, in which case,* π *is returned; see line 21*
(C2) An edge $e \notin E^*$ *is returned; see line 30*
(C3) An edge $e = (i,j) \in E^* \setminus M'$ *is returned such that* $w(i,j) < w'(i,M'(i))$*; see line 32*
(C4) An edge $e = (i,j) \in E^* \setminus M'$ *is returned such that* $w(i,j) = w'(i,M'(i))$*; see line 34*
Moreover, in the event of (C3) or (C4), edge $(i,M'(i)) \in M'$ *is available for the agent* i *on her turn while executing* π *in* G *using the query oracle.*

Proof. Using Corollary 4.5, we know that $w'(M^*) \leq w'(M')$ and $\mathrm{SW}(\pi;G) \leq w(M')$ hold true for any iteration of the while-loopẆe will show that these two inequalities satisfies at least one of the four conditions stated.

Let us first consider the case when $\mathrm{SW}(\pi;G) = w'(M') = w'(M^*)$; this satisfies Condition 4.7. Using Lemma 4.6, we know that π is welfare-optimal in G, and hence, the algorithm terminates with π as its output. Otherwise, we have the following two cases, wherein we will prove that either of Conditions 4.7,4.7, or 4.7 must be satisfied:

1. $w'(M^*) < w'(M')$
2. $\mathrm{SW}(\pi;G) < w'(M') = w'(M^*)$

Observe that, when Condition 4.7 is not satisfied, it implies that during the course of executing π in G using the query oracle, it must return an edge $e = (i,j)$ such that either

- $e \in M'$, but $w'(e) \neq w(e)$: this implies that e was not previously in E^* and we learn a new edge, satisfying Condition 4.7.
- $e \notin M'$, where either of Conditions 4.7 or 4.7 is satisfied. Here, since the considered iteration of the while-loop did not break till it returns $e \notin M'$, we know that all preceding queries returned for π must be edges in M'.

Therefore, it is enough to show that occurrence of cases (i) and (ii) implies that not all queries return edges in $E^* \cap M'$.

First, it is easy to see that when we have $w'(M^*) < w'(M')$, every edge that is returned while executing π cannot be in the set $E^* \cap M'$ (because in that case, $w'(M^*) = w'(M')$). Next, when $\mathrm{SW}(\pi;G) < w'(M') = w'(M^*)$, it may be the case that we have found the optimal social welfare value in G (which is equal to $w'(M') = w'(M^*)$), but even then we have not found the optimal action sequence. This is because π is not able to achieve the same social welfare in G as in G', i.e., $\mathrm{SW}(\pi;G) < w'(M')$. And hence, during the course of executing π in G, it must deviate from returning the edges in M' or encounter a new edge $e \notin E^*$ at least once. That is, all returned edges again cannot be in the set $E^* \cap M'$, thereby completing our proof. □

Next, we show that Algorithm 1 makes progress towards learning the true weight or the true rank functions in the underlying graph G with every iteration of its while-loop.

Lemma 4.8. *Given an n-agent* OSM *instance, every iteration of the* while-loop *in lines 20-40 of Algorithm 1 leads to at least one of the following progresses:*

- *The size of the set E^* increases by* 1.
- *There exists an agent $i \in [n]$ such that $w'(i, M'(i)) - w(i, M'(i))$ decreases strictly.*
- *There exists an agent $i \in [n]$ such that $\sum_{j\in[n]} |r'_i(j) - r_i(j)|$ decreases strictly.*

The above conditions ensure that $w(M')$ keeps decreasing, while $w(M^*)$ and $\mathrm{SW}(\pi; G)$ keeps increasing as the algorithm progresses.

Proof. On input an n-agent OSM instance, let us consider an arbitrary iteration of the while-loop during the execution of Algorithm 1. We assume that $\mathrm{SW}(\pi; G) = w'(M') = w'(M^*)$ does not hold true, since otherwise, the while-loop terminates. Let us consider the agent $i \in [n]$ at whose turn this iteration of the while-loop breaks. Lemma 4.7 states that one of the following three events must have occurred then.

Case 1: An edge $e \notin E^*$ is returned.

Here, edge e is added to the set E^* increasing its size by 1; see line 31. Since $|E^*| \leq n^2$, it can be updated for a maximum for n^2 many times throughout the execution of Algorithm 1.

Case 2: An edge $e = (i, j) \in E^* \setminus M'$ is returned such that $w(i, j) < w'(i, M'(i))$.

Here, $w'(i, M'(i))$ is updated to be equal to $w(i, j)$; see line 33. Using inequality (1) in the proof Lemma 4.4, we have $w'(i, M'(i)) > w(i, j) \geq w(i, M'(i))$. Hence, $w'(i, M'(i)) - w(i, M'(i))$ decreases strictly.

Case 3: An edge $e = (i, j) \in E^* \setminus M'$ is returned such that $w(i, j) = w'(i, M'(i))$.

The action sequence π is designed to produce M' in the graph G'. Just before the edge incident to agent i is chosen, both edges $e = (i, j)$ and $(i, M'(i))$ are available. According to r', the latter edge has higher priority, while according to r', the former edge has higher priority for agent i. Therefore, swapping $r'_i(j)$ and $r'_i(M'(i))$ decreases the sum $\sum_{j\in[n]} |r'_i(j) - r_i(j)|$ strictly. □

Corollary 4.9. *On input any n-agent* OSM *instance, Algorithm 1 can update the set $E^* \subseteq [n]^2$ for at most n^2 times.*

Corollary 4.10. *On input any n-agent* OSM *instance, Algorithm 1 updates the proxy weight $w'(i, j)$ of any edge $(i, j) \in [n]^2$ such that $w'(i, j) = w(i, j')$ for some $j' \in [n]$. That is, such an update can happen for a maximum of n times for any edge.*

Observation 4.11. The sum $\sum_{i,j\in[n]} |r'_i(j) - r_i(j)| \leq n^2$.

The proofs of Corollary 4.10 and Observation 4.11 can be found in the full version of the paper. We are now ready to prove our main result.

Theorem 4.2. *For n-agent* OSM *instances with query access, Algorithm 1 uses $\mathcal{O}(n^5)$ queries to compute a welfare-optimal action sequence.*

Proof. Consider an n-agent OSM instance having an underlying graph $G = K_{n,n}$ with edge-weight function w and rank functions r_i's. Algorithm 1 makes queries of the form $\mathsf{Query}(i, S)$ to an agent $i \in [n]$ for an action (sub)sequence $S \in \mathcal{S}_{-i}$ to learn the true weight and rank function. During its execution, our algorithm maintains a proxy graph $G' = K_{n,n}$ with a proxy edge-weight function w' and proxy rank functions r'_i's.

Note that, our algorithm terminates if and only it finds M^*, M', and π such that $\mathrm{SW}(\pi, G) = w'(M') = w'(M^*) > 0$. Lemma 4.6 proves that such an action sequence π must be optimal, establishing the correctness of Algorithm 1.

Potential Function: We measure the progress of our algorithm via building a potential function. We define it is a triple consisting of three quantities arranged in the lexicographic order: size $|E^*|$, distance $\sum_{i,j\in[n]} |w'(i, j) - w(i, j)|$ between w' and w, and distance $\sum_{i,j\in[n]} |r'_i(j) - r_i(j)|$ between r' and r.

Let us now analyse the number of queries Algorithm 1 requires to terminate. First, note that, Phase 1 and Phase 2 requires a total of $2n$ queries. Using Corollary 4.10, it follows that for any edge $(i, j) \in [n]^2$, its proxy weight $w'(i, j)$ is updated at most n times, and each time, it may take n iterations of the while-loop to find its correct rank position. Therefore, using Lemma 4.8, Corollary 4.9, and Observation 4.11, we know that every iteration of the while-loop makes progress in a way it will terminate within n^4 many iterations. Finally, note that any iteration of the while-loop makes n queries of the form $\mathsf{Query}(\mathrm{SW}(\pi))$ to compute the value of $\mathrm{SW}(\pi, G)$. Overall, Algorithm 1 must terminate after making $\mathcal{O}(n^5)$ queries, proving the stated claim. □

5 Conclusion and Open Problems

In this work, we explore the problem of computing maximum-weight matchings in complete bipartite weighted graphs through the lens of serial dictatorships. We resolve the main open problem listed in Caragiannis and Rathi (2024) and develop a novel query-efficient algorithm for n-agent OSM instances that finds a welfare-optimal action sequence. We show that even though the space of solutions is exponential, our algorithm only requires to make $\mathcal{O}(n^5)$ queries to discover a welfare-optimal action sequence.

We remark that following the set-up in Caragiannis and Rathi (2024), we can similarly develop a *truthful implementation* of Algorithm 1 by adapting the paradigm of VCG payments. We omit the details here (for not repeating), and refer our readers to Section 8 in Caragiannis and Rathi (2024).

Our work opens up various research directions, we list two of them here. First, it would be interesting to prove a *lower bound* on the query complexity of finding welfare-optimal action sequences in OSM instances. Furthermore, is it possible to extend and generalize our algorithmic ideas beyond matchings? Our work has given hope to have positive results, especially for those combinatorial optimization problems where the price of serial dictatorship is 1.

Acknowledgments

Caragiannis was partially supported by the Independent Research Fund Denmark (DFF) under grant 2032-00185B.

References

Abdulkadiroğlu, A.; and Sönmez, T. 1998. Random serial dictatorship and the core from random endowments in house allocation problems. *Econometrica*, 66(3): 689–701.

Beynier, A.; Bouveret, S.; Lemaître, M.; Maudet, N.; Rey, S.; and Shams, P. 2019. Efficiency, Sequenceability and Deal-Optimality in Fair Division of Indivisible Goods. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, (AAMAS)*, 900–908.

Bhalgat, A.; Chakraborty, T.; and Khanna, S. 2010. Approximating pure Nash equilibrium in cut, party affiliation, and satisfiability games. In *Proceedings of the 11th ACM conference on Electronic Commerce (EC)*, 73–82.

Bogomolnaia, A.; and Moulin, H. 2001. A new solution to the random assignment problem. *Journal of Economic Theory*, 100(2): 295–328.

Borodin, A.; Boyar, J.; Larsen, K. S.; and Mirmohammadi, N. 2010. Priority algorithms for graph optimization problems. *Theorerical Computer Science*, 411(1): 239–258.

Borodin, A.; Nielsen, M. N.; and Rackoff, C. 2003. Incremental priority algorithms. *Algorithmica*, 37(4): 295–326.

Bouveret, S.; and Lang, J. 2011. A general elicitation-free protocol for allocating indivisible goods. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI)*.

Caragiannis, I.; Filos-Ratsikas, A.; Frederiksen, S. K. S.; Hansen, K. A.; and Tan, Z. 2016. Truthful Facility Assignment with Resource Augmentation: An Exact Analysis of Serial Dictatorship. In *Proceedings of the 12th Conference on Web and Internet Economics (WINE)*, 236–250.

Caragiannis, I.; and Rathi, N. 2024. Optimizing over Serial Dictatorships. *Theory of Computing Systems*, 8: 1180–1206.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2009. *Introduction to Algorithms*. The MIT Press, 3rd edition.

Davis, S.; and Impagliazzo, R. 2004. Models of greedy algorithms for graph problems. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 381–390.

Deligkas, A.; Mertzios, G. B.; and Spirakis, P. G. 2017. The computational complexity of weighted greedy matching. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*, 466–472.

Filos-Ratsikas, A.; Frederiksen, S. K. S.; and Zhang, J. 2014. Social welfare in one-sided matchings: Random priority and beyond. In *Proceedings of the 7th International Symposium on Algorithmic Game Theory (SAGT)*, 1–12.

Gale, D.; and Shapley, L. S. 1962. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1): 9–15.

Gourvès, L.; Lesca, J.; and Wilczynski, A. 2021. On Fairness via Picking Sequences in Allocation of Indivisible Goods. In *Proceedings of the 7th International Conference on Algorithmic Decision Theory (ADT)*, 258–272.

Kalinowski, T.; Narodytska, N.; and Walsh, T. 2013. A Social Welfare Optimal Sequential Allocation Procedure. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI)*, 227–233.

Krysta, P.; Manlove, D. F.; Rastegari, B.; and Zhang, J. 2019a. Size Versus Truthfulness in the House Allocation Problem. *Algorithmica*, 81(9): 3422–3463.

Krysta, P.; Manlove, D. F.; Rastegari, B.; and Zhang, J. 2019b. Size Versus Truthfulness in the House Allocation Problem. *Algorithmica*, 81(9): 3422–3463.

Manlove, D. F. 2013. *Algorithmics of Matching Under Preferences*. World Scientific.

Poloczek, M.; Schnitger, G.; Williamson, D.; and Zuylen, A. 2017. Greedy algorithms for the maximum satisfiability problem: Simple algorithms and inapproximability bounds. *SIAM Journal on Computing*, 46: 1029–1061.

Roth, A. E.; and Sotomayor, M. A. O. 1990. *Two-Sided Matching: A Study in Game-Theoretic Modeling and Analysis*. Cambridge University Press.

Svensson, L.-G. 1999. Strategy-proof allocation of indivisible goods. *Social Choice and Welfare*, 16(4): 557–567.

Wooldridge, M.; Endriss, U.; Kraus, S.; and Lang, J. 2013. Incentive engineering for Boolean games. *Journal of Artificial Intelligence*, 195: 418–439.