Algorithms for Automatic Ranking of Participants and Tasks in an Anonymized Contest

Yang Jiao(✉), R. Ravi, and Wolfgang Gatterbauer

Tepper School of Business, Carnegie Mellon University,
5000 Forbes Avenue, Pittsburgh, PA 15213, USA
yangjiao@andrew.cmu.edu

Abstract. We introduce a new set of problems based on the *Chain Editing problem*. In our version of Chain Editing, we are given a set of anonymous participants and a set of undisclosed tasks that every participant attempts. For each participant-task pair, we know whether the participant has succeeded at the task or not. We assume that participants vary in their ability to solve tasks, and that tasks vary in their difficulty to be solved. In an ideal world, stronger participants should succeed at a superset of tasks that weaker participants succeed at. Similarly, easier tasks should be completed successfully by a superset of participants who succeed at harder tasks. In reality, it can happen that a stronger participant fails at a task that a weaker participants succeeds at. Our goal is to find a *perfect nesting of the participant-task relations* by flipping a minimum number of participant-task relations, implying such a "nearest perfect ordering" to be the one that is closest to the truth of participant strengths and task difficulties. Many variants of the problem are known to be NP-hard.

We propose six natural k-near versions of the Chain Editing problem and classify their complexity. The input to a k-near Chain Editing problem includes an initial ordering of the participants (or tasks) that we are required to respect by moving each participant (or task) at most k positions from the initial ordering. We obtain surprising results on the complexity of the six k-near problems: Five of the problems are polynomial-time solvable using dynamic programming, but one of them is NP-hard.

Keywords: Chain Editing · Chain Addition · Truth discovery · Massively open online classes · Student evaluation

1 Introduction

1.1 Motivation

Consider a contest with a set S of participants who are required to complete a set Q of tasks. Every participant either succeeds or fails at completing each

S.-H. Poon et al. (Eds.): WALCOM 2017, LNCS 10167, pp. 335–346, 2017.
DOI: 10.1007/978-3-319-53925-6_26

task. The identities of the participants and the tasks are anonymous. We aim to obtain rankings of the participants' strengths and the tasks' difficulties. This situation can be modeled by an unlabeled bipartite graph with participants on one side, tasks on the other side, and edges defined by whether the participant succeeded at the task. From the edges of the bipartite graph, we can infer that a participant a_2 is stronger than a_1 if the neighborhood of a_1 is contained in (or is "nested in") that of a_2. Similarly, we can infer that a task is easier than another if its neighborhood contains that of the other. See Fig. 1 for a visualization of strengths of participants and difficulties of tasks. If all neighborhoods are nested, then this nesting immediately implies a ranking of the participants and tasks. However, participants and tasks are not perfect in reality, which may result in a bipartite graph with "non-nested" neighborhoods. In more realistic scenarios, we wish to determine a ranking of the participants and the tasks when the starting graph is not ideal, which we define formally in Sect. 1.2.

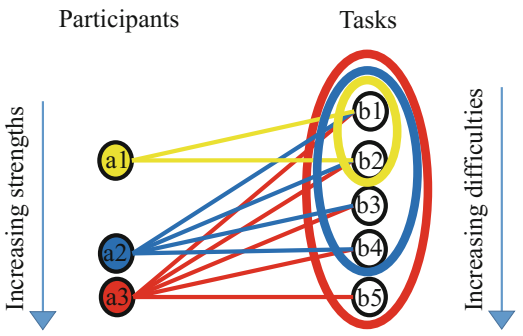

Fig. 1. An ideal graph is shown. Participants and tasks may be interpreted as students and questions, or actors and claims. Participant a_1 succeeds at b_1 to b_2; a_2 succeeds at b_1 to b_4; a_3 succeeds at b_1 to b_5. The nesting of neighborhoods here indicate that participant a_1 is weaker than a_2, who is weaker than a_3, and task b_1 and b_2 are easier than b_3 and b_4, which in turn are easier than b_5.

1.1.1 Relation to Truth Discovery

A popular application of unbiased rankings is computational "truth discovery." *Truth discovery* is the determination of trustworthiness of conflicting pieces of information that are observed often from a variety of sources [24] and is motivated by the problem of extracting information from networks where the trustworthiness of the actors are uncertain [15]. The most basic model of the problem is to consider a bipartite graph where one side is made up of actors, the other side is made up of their claims, and edges denote associations between actors and claims. Furthermore, claims and actors are assumed to have "trustworthiness" and "believability" scores, respectively, with known a priori values. According to a number of recent surveys [15,20,24], common approaches for truth discovery include iterative procedures, optimization methods, and probabilistic graphic

models. Iterative methods [9,13,22,27] update trust scores of actors to believability scores of claims, and vice versa, until convergence. Variants of these methods (such as Sums, Hubs and Authorities [18], AverageLog, TruthFinder, Investment, and PooledInvestment) have been extensively studied and proven in practice [2]. Optimization methods [3,19] aim to find truths that minimize the total distance between the provided claims and the output truths for some specified continuous distance function; coordinate descent [5] is often used to obtain the solution. Probabilistic graphical models [23] of truth discovery are solved by expectation maximization. Other methods for truth discovery include those that leverage trust relationships between the sources [14]. Our study is conceptually closest to optimization approaches (we minimize the number of edge additions or edits), however we suggest a *discrete objective* for minimization, for which we need to develop new algorithms.

1.1.2 Our Context: Massively Open Online Courses

Our interest in the problem arises from trying to model the problem of automatic grading of large number of students in the context of MOOCs (massively open online courses). Our idea is to crowd-source the creation of automatically gradable questions (like multiple choice items) to students, and have all the students take all questions. From the performance of the students, we would like to quickly compute a roughly accurate ordering of the difficulty of the crowd-sourced questions. Additionally, we may also want to efficiently rank the strength of the students based on their performance. Henceforth, we refer to participants as students and tasks as questions in the rest of the paper.

1.1.3 Our Model

We cast the ranking problem as a discrete optimization problem of minimizing the number of changes to a given record of the students' performance to obtain nested neighborhoods. This is called the Chain Editing problem. It is often possible that some information regarding the best ranking is already known. For instance, if the observed rankings of students on several previous assignments are consistent, then it is likely that the ranking on the next assignment will be similar. We model known information by imposing an additional constraint that the changes made to correct the errors to an ideal ranking must result in a ranking that is near a given base ranking. By near, we mean that the output position of each student should be within at most k positions from the position in the base ranking, where k is a parameter. Given a nearby ranking for students, we consider all possible variants arising from how the question ranking is constrained. The question ranking may be constrained in one of the following three ways: the exact question ranking is specified (which we term the "constrained" case), it must be near a given question ranking (the "both near" case), or the question ranking is unconstrained (the "unconstrained" case). We provide the formal definitions of these problems next.

1.2 Problem Formulations

Here, we define all variants of the ranking problem. The basic variants of Chain Editing are defined first and the k-near variants are defined afterward.

1.2.1 Basic Variants of Chain Editing

First, we introduce the problem of recognizing an ideal input. Assume that we are given a set S of students, and a set Q of questions, and edges between S and Q that indicate which questions the students answered correctly - note that we assume that every student attempts every question. Denote the resulting bipartite graph by $G = (S \cup Q, E)$. For every pair $(s, q) \in S \times Q$, we are given an edge between s and q if and only if student s answered question q correctly. For a graph (V, E), denote the neighborhood of a vertex x by $N(x) := \{y \in V : xy \in E\}$.

Definition 1. *We say that student s_1 is* stronger *than s_2 if $N(s_1) \supset N(s_2)$. We say that question q_1 is* harder *than q_2 if $N(q_1) \subset N(q_2)$. Given an ordering α on the students and β on the questions, $\alpha(s_1) \geq \alpha(s_2)$ shall indicate that s_1 is stronger than s_2, and $\beta(q_1) \geq \beta(q_2)$ shall indicate that q_1 is harder than q_2.*

Definition 2. *An ordering of the questions satisfies the* interval property *if for every s, its neighborhood $N(s)$ consists of a block of consecutive questions (starting with the easiest question) with respect to our ordering of the questions. An ordering α of the students is* nested *if $\alpha(s_1) \geq \alpha(s_2) \Rightarrow N(s_1) \supseteq N(s_2)$.*

Definition 3. *The objective of the* Ideal Mutual Orderings (IMO) *problem is to order the students and the questions so that they satisfy the nested and interval properties respectively, or output NO if no such orderings exist.*

Observe that IMO can be solved efficiently by comparing containment relation among the neighborhoods of the students and ordering the questions and students according to the containment order.

Proposition 1. *There is a polynomial time algorithm to solve IMO.*

All missing proofs are in the full version of the paper [16]. Next, observe that the nested property on one side is satisfiable if and only if the interval property on the other side is satisfiable. Hence, we will require only the nested property in subsequent variants of the problem.

Proposition 2. *A bipartite graph has an ordering of all vertices so that the questions satisfy the interval property if and only if it has an ordering with the students satisfying the nested property.*

Next, we define several variants of IMO.

Definition 4. *In the* Chain Editing (CE) *problem, we are given a bipartite graph representing student-question relations and asked to find a minimum set of edge edits that admits an ordering of the students satisfying the nested property.*

A more restrictive problem than Chain Editing is Chain Addition. Chain Addition is variant of Chain Editing that allows only edge additions and no deletions. Chain Addition models situations where students sometimes accidentally give wrong answers on questions they know how to solve but never answer a hard problem correctly by luck, e.g. in numerical entry questions.

Definition 5. *In the* Chain Addition (CA) *problem, we are given a bipartite graph representing student-question relations and asked to find a minimum set of edge additions that admits an ordering of the students satisfying the nested property.*

Analogous to needing only to satisfy one of the two properties, it suffices to find an optimal ordering for only one side. Once one side is fixed, it is easy to find an optimal ordering of the other side respecting the fixed ordering.

Proposition 3. *In Chain Editing, if the best ordering (that minimizes the number of edge edits) for either students or questions is known, then the edge edits and ordering of the other side can be found in polynomial time.*

1.2.2 *k*-near Variants of Chain Editing

We introduce and study the nearby versions of Chain Editing or Chain Addition. Our problem formulations are inspired by Balas and Simonetti's [4] work on k-near versions of the TSP.

Definition 6. *In the* k-near *problem, we are given an initial ordering* $\alpha : S \to [|S|]$ *and a positive integer* k*. A* feasible *solution exhibits a set of edge edits (additions) attaining the nested property so that the associated ordering* π*, induced by the neighborhood nestings, of the students satisfies* $\pi(s) \in [\alpha(s) - k, \alpha(s) + k]$*.*

Next, we define three types of k-near problems. In the subsequent problem formulations, we bring back the interval property to our constraints since we will consider problems where the question side is not allowed to be arbitrarily ordered.

Definition 7. *In* Unconstrained k-near *Chain Editing (Addition), the student ordering must be* k*-near but the question side may be ordered any way. The objective is to minimize the number of edge edits (additions) so that there is a* k*-near ordering of the students that satisfies the nested property.*

Definition 8. *In* Constrained k-near *Chain Editing (Addition), the student ordering must be* k*-near while the questions have a fixed initial ordering that must be kept. The objective is to minimize the number of edge edits (additions) so that there is* k*-near ordering of the students that satisfies the nested property and respects the interval property according to the given question ordering.*

Definition 9. *In* Both k-near *Chain Editing (Addition), both sides must be* k*-near with respect to two given initial orderings on their respective sides. The objective is to minimize the number of edge edits (additions) so that there is a* k*-near ordering of the students that satisfies the nested property and a* k*-near ordering of the questions that satisfies the interval property.*

1.3 Main Results

In this paper, we introduce k-near models to the Chain Editing problem and present surprising complexity results. Our k-near model captures realistic scenarios of MOOCs, where information from past tests is usually known and can be used to arrive at a reliable initial nearby ordering.

We find that five of the k-near Editing and Addition problems have polynomial time algorithms while the Unconstrained k-near Editing problem is NP-hard. Our intuition is that the Constrained k-near and Both k-near problems are considerably restrictive on the ordering of the questions, which make it easy to derive the best k-near student ordering. The Unconstrained k-near Addition problem is easier than the corresponding Editing problem because the correct neighborhood of the students can be inferred from the neighborhoods of all weaker students in the Addition problem, but not for the Editing version.

Aside from restricting the students to be k-near, we may consider all possible combinations of whether the students and questions are each k-near, fixed, or unconstrained. The remaining (non-symmetric) combinations not covered by the above k-near problems are both fixed, one side fixed and the other side unconstrained, and both unconstrained. The both fixed problem is easy as both orderings are given in the input and one only needs to check whether the orderings are consistent with the nesting of the neighborhoods. When one side is fixed and the other is unconstrained, we have already shown that the ordering of the unconstrained side is easily derivable from the ordering of the fixed side via Proposition 3. If both sides are unconstrained, this is exactly the Chain Editing (or Addition) problem, which are both known to be NP-hard (see below). Figure 2 summarizes the complexity of each problem, including our results for the k-near variants, which are starred. Note that the role of the students and questions are symmetric up to flipping the orderings.

Questions \ Students	Unconstrained	k-near		Constrained
		Editing	Addition	
Unconstrained	NP-hard [26,10]	NP-hard	$\mathcal{O}(n^3k^{2k+2})$	$\mathcal{O}(n^2)$
k-near Editing	NP-hard	$\mathcal{O}(n^3k^{4k+4})$		$\mathcal{O}(n^3k^{2k+2})$
k-near Addition	$\mathcal{O}(n^3k^{2k+2})$		$\mathcal{O}(n^3k^{4k+4})$	$\mathcal{O}(n^3k^{2k+2})$
Constrained	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3k^{2k+2})$	$\mathcal{O}(n^3k^{2k+2})$	$\mathcal{O}(n^2)$

Fig. 2. All variants of the problems are shown with their respective complexities. The complexity of Unconstrained/Unconstrained Addition [26] and Editing [10] were derived before. All other results are given in this paper. Most of the problems have the same complexity for both Addition and Editing versions. The only exception is the Unconstrained k-near version where Editing is NP-hard while Addition has a polynomial time algorithm.

To avoid any potential confusion, we emphasize that our algorithms are not fixed-parameter tractable algorithms, as our parameter k is not a property of

problem instances, but rather is part of the constraints that are specified for the outputs to satisfy.

The remaining sections are organized as follows. Section 2 discusses existing work on variants of Chain Editing that have been studied before. Section 3 shows the exact algorithms for five of the k-near problems and includes the NP-hardness proof for the last k-near problem. Section 4 summarizes our main contributions.

2 Related Work

The earliest known results on hardness and algorithms tackled Chain Addition. Before stating the results, we define a couple of problems closely related to Chain Addition. The *Minimum Linear Arrangement* problem considers as input a graph $G = (V, E)$ and asks for an ordering $\pi : V \to [|V|]$ minimizing $\sum_{vw \in E} |\pi(v) - \pi(w)|$. The *Chordal Completion* problem, also known as the *Minimum Fill-In* problem, considers as input a graph $G = (V, E)$ and asks for the minimum size set of edges F to add to G so that $(V, E \cup F)$ has no chordless cycles. A *chordless* cycle is a cycle $(v_1, \ldots, v_r, v_1)$ such that for every i, j with $|i - j| > 1$ and $\{i, j\} \neq \{1, r\}$, we have $v_i v_j \notin E$. Yannakakis [26] proved that Chain Addition is NP-hard by a reduction from Linear Arrangement. He also showed that Chain Addition is a special case of Chordal Completion on graphs of the form $(G = U \cup V, E)$ where U and V are cliques. Recently, Chain Editing was shown to be NP-hard by Drange et al. [10].

Another problem called *Total Chain Addition* is essentially identical to Chain Addition, except that the objective function counts the number of total edges in the output graph rather than the number of edges added. For Total Chain Addition, Feder et al. [11] give a 2-approximation. The total edge addition version of Chordal Completion has an $O(\sqrt{\Delta} \log^4(n))$-approximation algorithm [1] where Δ is the maximum degree of the input graph. For Chain Addition, Feder et al. [11] claim an $8d + 2$-approximation, where d is the smallest number such that every vertex-induced subgraph of the original graph has some vertex of degree at most d. Natanzon et al. [21] give an $8OPT$-approximation for Chain Addition by approximating Chordal Completion. However, no approximation algorithms are known for Chain Editing.

Modification to chordless graphs and to chain graphs have also been studied from a fixed-parameter point of view. A *fixed-parameter tractable (FPT)* algorithm for a problem of input size n and parameter p bounding the value of the optimal solution, is an algorithm that outputs an optimal solution in time $O(f(p)n^c)$ for some constant c and some function f dependent on p. For Chordal Completion, Kaplan et al. [17] give an FPT in time $O(2^{O(OPT)} + OPT^2 nm)$. Fomin and Villanger [12] show the first subexponential FPT for Chordal Completion, in time $O(2^{O(\sqrt{OPT} \log OPT)} + OPT^2 nm)$. Cao and Marx [7] study a generalization of Chordal Completion, where three operations are allowed: vertex deletion, edge addition, and edge deletion. There, they give an FPT in time $2^{O(OPT \log OPT)} n^{O(1)}$, where OPT is now the minimum total number of the three

operations needed to obtain a chordless graph. For the special case of Chain Editing, Drange et al. [10] show an FPT in time $2^{O(\sqrt{OPT}\log OPT)} + \text{poly}(n)$. They also show the same result holds for a related problem called Threshold Editing.

On the other side, Drange et al. [10] show that Chain Editing and Threshold Editing do not admit $2^{o(\sqrt{OPT})}\text{poly}(n)$ time algorithms assuming the Exponential Time Hypothesis (ETH). For Chain Completion and Chordal Completion, Bliznets et al. [6] exclude the possibility of $2^{O(\sqrt{n}/\log n)}$ and $2^{O(OPT^{\frac{1}{4}}/\log^c k)}n^{O(1)}$ time algorithms assuming ETH, where c is a constant. For Chordal Completion, Cao and Sandeep [8] showed that no algorithms in time $2^{O(\sqrt{OPT}-\delta)}n^{O(1)}$ exist for any positive δ, assuming ETH. They also exclude the possibility of a PTAS for Chordal Completion assuming $P \neq NP$. Wu et al. [25] show that no constant approximation is possible for Chordal Completion assuming the Small Set Expansion Conjecture. Table 1 summarizes the known results for the aforementioned graph modification problems.

Table 1. Known results

	Chordal	Chain
Editing	Unknown approximation, FPT [9]	Unknown approximation, FPT [9]
Addition	$8OPT$-approx [21], FPT [9]	$8OPT$-approx [21], $8d+2$-approx [11], FPT [9]
Total addition	$O(\sqrt{\Delta}\log^4(n))$-approx [1], FPT [9]	2-approx [11], FPT [9]

For the k-near problems, we show that the Unconstrained k-near Editing problem is NP-hard by adapting the NP-hardness proof for Threshold Editing from Drange et al. [9]. The remaining k-near problems have not been studied.

3 Polynomial Time Algorithms for k-near Orderings

We present our polynomial time algorithm for the Constrained k-near Addition problem and state similar results for the Constrained k-near Editing problem, the Both k-near Addition and Editing problems, and the Unconstrained k-near Addition problem. The algorithms and analyses for the other polynomial time results use similar ideas as the one for Constrained k-near Addition. They are provided in detail in the full paper [16]. We also state the NP-hardness of the Unconstrained k-near Editing problem and provide the proof in the full paper [16].

We assume correct orderings label the students from weakest (smallest label) to strongest (largest label) and label the questions from easiest (smallest label) to hardest (largest label). We associate each student with its initial label given by the k-near ordering. For ease of reading, we boldface the definitions essential to the analysis of our algorithm.

Theorem 1 (Constrained k-near Editing). *Constrained k-near Editing can be solved in time $O(n^3 k^{2k+2})$.*

Proof. Assume that the students are given in k-near order $1, \ldots, |S|$ and that the questions are given in exact order $1 \leq \cdots \leq |Q|$. We construct a dynamic program for Constrained k-near Editing. First, we introduce the subproblems that we will consider. Define $\boldsymbol{C(i, u_i, U_i, v_{j_i})}$ to be the smallest number of edges incident to the weakest i positions that must be edited such that u_i is in position i, U_i is the set of students in the weakest $i-1$ positions, and v_{j_i} is the hardest question correctly answered by the i weakest students. Before deriving the recurrence, we will define several sets that bound our search space within polynomial size of $n = |S| + |Q|$.

Search Space for U_i. Given position i and student u_i, define $\boldsymbol{P_{i,u_i}}$ to be the set of permutations on the elements in $\big[\max\{1, i-k\}, \min\{|S|, i+k-1\}\big] \setminus \{u_i\}$. Let $\boldsymbol{F_{i,u_i}} := \Big\{\{\pi^{-1}(1), \ldots, \pi^{-1}(k)\} : \pi \in P_{i,u_i}, \pi(a) \in [a-k, a+k] \forall a \in \big[\max\{1, i-k\}, \min\{|S|, i+k-1\}\big] \setminus \{u_i\}\Big\}$. The set P_{i,u_i} includes all possible permutations of the $2k$ students centered at position i, and the set F_{i,u_i} enforces that no student moves more than k positions from its label. We claim that every element of F_{i,u_i} is a candidate for $U_i \setminus \big[1, \max\{1, i-k-1\}\big]$ given that u_i is assigned to position i. To understand the search space for U_i given i and u_i, observe that for all $i \geq 2$, U_i already must include all of $\big[1, \max\{1, i-k-1\}\big]$ since any student initially at position $\leq i-k-1$ cannot move beyond position $i-1$ in a feasible solution. If $i = 1$, we have $U_1 = \emptyset$. From now on, we assume $i \geq 2$ and treat the base case $i = 1$ at the end. So the set $U_i \setminus \big[1, \max\{1, i-k-1\}\big]$ will uniquely determine U_i. We know that U_i cannot include any students with initial label $[k+i, |S|]$ since students of labels $\geq k+i$ must be assigned to positions i or later. So the only uncertainty remaining is which elements in $\big[\max\{1, i-k\}, \min\{|S|, i+k-1\}\big] \setminus \{u_i\}$ make up the set $U_i \setminus \big[1, \max\{1, i-k-1\}\big]$. We may determine all possible candidates for $U_i \setminus \big[1, \max\{1, i-k-1\}\big]$ by trying all permutations of $\big[\max\{1, i-k\}, \min\{|S|, i+k-1\}\big] \setminus \{u_i\}$ that move each student no more than k positions from its input label, which is exactly the set F_{i,u_i}.

Feasible and Compatible Subproblems. Next, we define $\boldsymbol{S_i} = \Big\{(u_i, U_i, v_{j_i}) : u_i \in \big[\max\{1, i-k\}, \min\{|S|, i+k\}\big], U_i \setminus \big[1, \max\{1, i-k-1\}\big] \in F_{i,u_i}, v_{j_i} \in Q \cup \{0\}\Big\}$. The set S_i represents the search space for all possible vectors (u_i, U_i, v_{j_i}) given that u_i is assigned to position i. Note that u_i is required to be within k positions of i by the k-near constraint. So we encoded this constraint into S_i. To account for the possibility that the i weakest students answer no questions correctly, we allow v_{j_i} to be in position 0, which we take to mean that $U_i \cup \{u_i\}$ gave wrong answers to all questions.

Now, we define $\boldsymbol{R_{i-1,u_i,U_i,v_{j_i}}} := \{(u_{i-1}, U_{i-1}, v_{j_{i-1}}) \in S_{i-1} : v_{j_{i-1}} \leq v_{j_i}, U_i = \{u_{i-1}\} \cup U_{i-1}\}$. The set $R_{i-1,u_i,U_i,v_{j_i}}$ represents the search space for smaller subproblems that are compatible with the subproblem (i, u_i, U_i, v_{j_i}). More precisely, given that u_i is assigned to position i, U_i is the set of students

assigned to the weakest $i-1$ positions, and v_{j_i} is the hardest question correctly answered by $U_i \cup u_i$, the set of subproblems of the form $(i-1, u_{i-1}, U_{i-1}, v_{j_{i-1}})$ which do not contradict the aforementioned assumptions encoded by (i, u_i, U_i, v_{j_i}) are exactly those whose $(u_{i-1}, U_{i-1}, v_{j_{i-1}})$ belongs to $R_{i-1,u_i,U_i,v_{j_i}}$. We illustrate compatibility in Fig. 3.

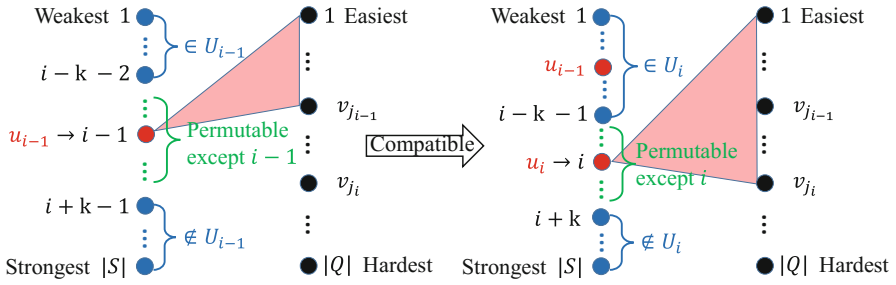

Fig. 3. Subproblem $(i-1, u_{i-1}, U_{i-1}, v_{j_{i-1}})$ is compatible with subproblem (i, u_i, U_i, v_{j_i}) if and only if $v_{j_{i-1}}$ is no harder than v_{j_i} and $U_i = \{u_{i-1}\} \cup U_{i-1}$. The cost of (i, u_i, U_i, v_{j_i}) is the sum of the minimum cost among feasible compatible subproblems of the form $(i-1, u_{i-1}, U_{i-1}, v_{j_{i-1}})$ and the minimum number of edits incident to u_i to make its neighborhood exactly $\{1, \ldots, v_{j_i}\}$.

The Dynamic Program. Finally, we define $c_{u_i, v_{j_i}}$ to be the smallest number of edge edits incident to u_i so that the neighborhood of u_i becomes exactly $\{1, \ldots, v_{j_i}\}$, i.e. $c_{u_i,v_{j_i}} := |N_G(u_i) \triangle \{1, \ldots, v_{j_i}\}|$. We know that $c_{u_i,v_{j_i}}$ is part of the cost within $C(i, u_i, U_i, v_{j_i})$ since v_{j_i} is the hardest question that $U_i \cup \{u_i\}$ is assumed to answer correctly and u_i is a stronger student than those in U_i who are in the positions before i. We obtain the following recurrence.

$$C(i, u_i, U_i, v_{j_i}) = \min_{(u_{i-1}, U_{i-1}, v_{j_{i-1}}) \in R_{i-1,u_i,U_i,v_{j_i}}} \{C(i-1, u_{i-1}, U_{i-1}, v_{j_{i-1}})\} + c_{u_i,v_{j_i}}$$

The base cases are $C(1, u_1, U_1, v_{j_1}) = |N_G(u_1) \triangle \{1, \ldots, v_{j_1}\}|$ if $v_{j_1} > 0$, and $C(1, u_1, U_1, v_{j_1}) = |N_G(u_1)|$ if $v_{j_1} = 0$ for all $u_1 \in [1, 1+k], v_{j_1} \in Q \cup \{0\}$.

By definition of our subproblems, the final solution we seek is $\min_{(u_{|S|}, U_{|S|}, v_{j_{|S|}}) \in S_{|S|}} C(|S|, u_{|S|}, U_{|S|}, v_{j_{|S|}})$.

Running Time. Now, we bound the run time of the dynamic program. Note that before running the dynamic program, we build the sets P_{i,u_i}, F_{i,u_i}, S_i, $R_{i-1,u_i,U_i,v_{j_i}}$ to ensure that our solution obeys the k-near constraint and that the smaller subproblem per recurrence is compatible with the bigger subproblem it came from. Generating the set P_{i,u_i} takes $(2k)! = O(k^k)$ time per (i, u_i). Checking the k-near condition to obtain the set F_{i,u_i} while building P_{i,u_i} takes k^2 time per (i, u_i). So generating S_i takes $O(k \cdot k^k k^2 \cdot |Q|)$ time per i. Knowing S_{i-1}, generating $R_{i-1,u_i,U_i,v_{j_i}}$ takes $O(|S|)$ time. Hence, generating all of the sets is dominated by the time to build $\cup_{i \le |S|} S_i$, which is $O(|S| k^3 k^k |Q|) = O(n^2 k^{k+3})$.

After generating the necessary sets, we solve the dynamic program. Each subproblem (i, u_i, U_i, v_{j_i}) takes $O(|R_{i-1,u_i,U_i,v_{j_i}})|$ time. So the total time to solve the dynamic program is $O(\sum_{i \in S,(u_i,U_i,v_{j_i}) \in S_i} |R_{i-1,u_i,U_i,v_{j_i}}|) = O(|S||S_i||S_{i-1}|) = O(n(k \cdot k^k \cdot n)^2) = O(n^3 k^{2k+2})$. □

Theorem 2 (Constrained k-near Addition). *Constrained k-near Addition can be solved in time $O(n^3 k^{2k+2})$.*

Theorem 3 (Unconstrained k-near Addition). *Unconstrained k-near Addition can be solved in time $O(n^3 k^{2k+2})$.*

Theorem 4 (Unconstrained k-near Editing). *Unconstrained k-near Editing is NP-hard.*

Theorem 5 (Both k-near Editing). *Both k-near Editing can be solved in time $O(n^3 k^{4k+4})$.*

Theorem 6 (Both k-near Addition). *Both k-near Addition can be solved in time $O(n^3 k^{4k+4})$.*

We present the proofs of the above theorems in the full paper [16].

4 Conclusion

We proposed a new set of problems that arise naturally from ranking participants and tasks in competitive settings and classified the complexity of each problem. First, we introduced six k-near variants of the Chain Editing problem, which capture the common scenario of having partial information about the final orderings from past rankings. Second, we provided polynomial time algorithms for five of the problems and showed NP-hardness for the remaining one.

Acknowledgments. This work was supported in part by the US National Science Foundation under award numbers CCF-1527032, CCF-1655442, and IIS-1553547.

References

1. Agrawal, A., Klein, P., Ravi, R.: Cutting down on fill using nested dissection: provably good elimination orderings. In: George, A., Gilbert, J.R., Liu, J.W.H. (eds.) Graph Theory and Sparse Matrix Computation, pp. 31–55. Springer, Heidelberg (1993)
2. Andersen, R., Borgs, C., Chayes, J., Feige, U., Flaxman, A., Kalai, A., Mirrokni, V., Tennenholtz, M.: Trust-based recommendation systems: an axiomatic approach. In: WWW, pp. 199–208. ACM (2008)
3. Aydin, B., Yilmaz, Y., Li, Y., Li, Q., Gao, J., Demirbas, M.: Crowdsourcing for multiple-choice question answering. In: IAAI, pp. 2946–2953 (2014)
4. Balas, E., Simonetti, N.: Linear time dynamic-programming algorithms for new classes of restricted TSPs: a computational study. INFORMS J. Comput. **13**(1), 56–75 (2000)

5. Bertsekas, D.P.: Non-linear Programming. Athena Scientific, Belmont (1999)
6. Bliznets, I., Cygan, M., Komosa, P., Mach, L., Pilipczuk, M.: Lower bounds for the parameterized complexity of minimum fill-in and other completion problems. In: SODA, pp. 1132–1151 (2016)
7. Cao, Y., Marx, D.: Chordal editing is fixed-parameter tractable. Algorithmica **75**(1), 118–137 (2016)
8. Cao, Y., Sandeep, R.B.: Minimum fill-in: inapproximability and almost tight lower bounds. CoRR abs/1606.08141 (2016). http://arxiv.org/abs/1606.08141
9. Dong, X.L., Berti-Equille, L., Srivastava, D.: Integrating conflicting data: the role of source dependence. PVLDB **2**(1), 550–561 (2009)
10. Drange, P.G., Dregi, M.S., Lokshtanov, D., Sullivan, B.D.: On the threshold of intractability. In: ESA, pp. 411–423 (2015)
11. Feder, T., Mannila, H., Terzi, E.: Approximating the minimum chain completion problem. Inf. Process. Lett. **109**(17), 980–985 (2009)
12. Fomin, F.V., Villanger, Y.: Subexponential parameterized algorithm for minimum fill-in. In: SODA, pp. 1737–1746 (2012)
13. Galland, A., Abiteboul, S., Marian, A., Senellart, P.: Corroborating information from disagreeing views. In: WSDM, pp. 131–140. ACM (2010)
14. Gatterbauer, W., Suciu, D.: Data conflict resolution using trust mappings. In: SIGMOD, pp. 219–230 (2010)
15. Gupta, M., Han, J.: Heterogeneous network-based trust analysis: a survey. ACM SIGKDD Explor. Newsl. **13**(1), 54–71 (2011)
16. Jiao, Y., Ravi, R., Gatterbauer, W.: Algorithms for automatic ranking of participants and tasks in an anonymized contest. CoRR abs/1612.04794 (2016). http://arxiv.org/abs/1612.04794
17. Kaplan, H., Shamir, R., Tarjan, R.E.: Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. SIAM J. Comput. **28**(5), 1906–1922 (1999)
18. Kleinberg, J.M.: Authoritative sources in a hyperlinked environment. JACM **46**(5), 604–632 (1999)
19. Li, Q., Li, Y., Gao, J., Zhao, B., Fan, W., Han, J.: Resolving conflicts in heterogeneous data by truth discovery and source reliability estimation. In: SIGMOD, pp. 1187–1198 (2014)
20. Li, Y., Gao, J., Meng, C., Li, Q., Su, L., Zhao, B., Fan, W., Han, J.: A survey on truth discovery. ACM SIGKDD Explor. Newsl. **17**(2), 1–16 (2015)
21. Natanzon, A., Shamir, R., Sharan, R.: A polynomial approximation algorithm for the minimum fill-in problem. SIAM J. Comput. **30**(4), 1067–1079 (2000)
22. Pasternack, J., Roth, D.: Knowing what to believe (when you already know something). In: COLING, pp. 877–885 (2010)
23. Pasternack, J., Roth, D.: Latent credibility analysis. In: WWW, pp. 1009–1021 (2013)
24. Pasternack, J., Roth, D., Vydiswaran, V.V.: Information trustworthiness. AAAI Tutorial (2013)
25. Wu, Y.L., Austrin, P., Pitassi, T., Liu, D.: Inapproximability of treewidth, one-shot pebbling, and related layout problems. J. Artif. Int. Res. **49**(1), 569–600 (2014)
26. Yannakakis, M.: Computing the minimum fill-in is NP-complete. SIAM J. Algebr. Discret. Methods **2**(1), 77–79 (1981)
27. Yin, X., Han, J., Yu, P.S.: Truth discovery with multiple conflicting information providers on the web. TKDE **20**(6), 796–808 (2008)