Theoretical Computer Science 789 (2019) 64–76

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Algorithms for automatic ranking of participants and tasks in an anonymized contest ☆

Yang Jiao [a,*], R. Ravi [a,*], Wolfgang Gatterbauer [b,*]

[a] Tepper School of Business, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213, United States of America
[b] College of Computer and Information Science, Northeastern University, 440 Huntington Ave #202, Boston, MA 02115, United States of America

ARTICLE INFO

Article history:
Received 26 August 2017
Received in revised form 11 June 2018
Accepted 17 July 2018
Available online 31 July 2018

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

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 participants and a set of 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 participant 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 the final solution is required to be "close" to, 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.

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 task. We aim to obtain rankings of the participants' strengths and the tasks' difficulties. This situation can be modeled by a bipartite graph with participants on one side, tasks on the other side, and edges present if a participant succeeded at the task. From the edges of the bipartite graph, we can infer that a participant a_2 is stronger

☆ This paper is the full version of a paper with the same title presented at the 11th International Conference and Workshops on Algorithms and Computation [16]. It contains all proofs that were omitted in [16]. Updates will be available on http://arxiv.org/abs/1612.04794.

* Corresponding authors.
E-mail addresses: yangjiao@andrew.cmu.edu (Y. Jiao), ravi@andrew.cmu.edu (R. Ravi), wolfgang@ccis.neu.edu (W. Gatterbauer).

https://doi.org/10.1016/j.tcs.2018.07.014
0304-3975/

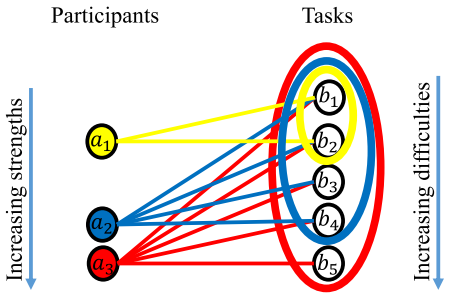

Fig. 1. An "ideal" graph is shown. Participants and tasks may be interpreted as students and questions, or actors and claims. Participant a_1 succeeds at b_1 to b_2; a_2 succeeds at b_1 to b_4; a_3 succeeds at b_1 to b_5. The nesting of neighborhoods here indicate that participant a_1 is weaker than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

than a_1 if the neighborhood of a_1 is strictly contained in (or is strictly "nested in") that of a_2. Similarly, we can infer that a task is easier than another if its neighborhood strictly contains that of the other. If two participants or tasks have the same neighborhood, then they are considered equally strong or equally easy. 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. For such more realistic scenarios, we wish to determine a ranking of the participants and the tasks that is still "close" to the ideal case. In this paper, we define several variants of this problem that are different in what changes can be made (adding, deleting, or adding and deleting edges) and prior knowledge of rankings (exact for one side, no prior knowledge, nearby starting values) that together give rise to varying problem complexities.

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 [15,24,20], common approaches for truth discovery include iterative procedures, optimization methods, and probabilistic graphic models. (1) Iterative methods [9,13,22,27] update trust scores of actors to believability scores of claims, and vice versa, until convergence. Various variants of these methods (such as Hubs and Authorities (or Sums) [18], TruthFinder [27], AverageLog, Investment, and PooledInvestment [22]) have been extensively studied and proven in practice [2]. (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. (3) 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), but we suggest a *discrete objective* for minimization, for which we need to develop new algorithms.

1.1.2. Our motivation: 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). Manual grading of assignments from many students is infeasible. In turn, creating many automatically gradable questions (that are also relevant to the topics of a class) is difficult. Our idea is to crowd-source the creation of automatically gradable questions (in particular, multiple choice items) to students, and have all the students take all questions. In this context, we do not know the difficulty of questions and would like to quickly compute a roughly accurate ordering of the difficulty of the crowd-sourced questions from the answers chosen by the students. Additionally, we also want to rank the strength of the students based on their performance. In an ideal world, stronger participants should succeed at a superset of tasks that weaker participants succeed at, which motivates our nesting property. In reality, it can happen that a stronger participant fails at a task that a weaker participant succeeds at. Our goal is to find a ranking of students and questions that "explains" our observations as much as possible and is thus a close to the ideal case as possible.

1.1.3. Our model

Henceforth, we refer to participants as students and tasks as questions in the rest of the paper. 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 the 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: (*i*) the exact question ranking is specified (which we term the "constrained" case), (*ii*) it must be near a given question ranking (the "both near" case), or (*iii*) 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. Every student attempts every question. Edges between S and Q indicate which questions the students answered correctly. Denote the resulting bipartite graph by $G = (S \cup Q, E)$. Let $n = |S| + |Q|$. 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 $N(x) := \{y \in V : xy \in E\}$. In other words, the neighborhood of a question is the set of student who answered the question correctly. Similarly, the neighborhood of a student is the set of questions that the student answered correctly.

Definition 1.1 *(Strength and difficulty).* We say that student s_1 is *stronger* than student s_2 if $N(s_1) \supset N(s_2)$, and student s_1 is *equivalent* to s_2 if $N(s_1) = N(s_2)$. We say that question q_1 is *harder* than question q_2 if $N(q_1) \subset N(q_2)$, and question q_1 is *equivalent* to question q_2 if $N(q_1) = N(q_2)$. Given an ordering α on the students and β on the questions, $\alpha(s_1) > \alpha(s_2)$ shall indicate that s_1 is stronger than s_2; $\beta(q_1) > \beta(q_2)$ shall indicate that q_1 is harder (more difficult) than q_2; $\alpha(s_1) = \alpha(s_2)$ and $\beta(q_1) = \beta(q_2)$ shall indicate that s_1 is equivalent to s_2 and q_1 is equivalent to q_2, respectively.

Definition 1.2 *(Interval and nesting properties).* An ordering of the questions satisfies the *interval property* if for every student s, its neighborhood $N(s)$ consists of a block of consecutive questions (starting with the easiest question) with respect to the ordering of the questions. An ordering α of the students satisfies the *nesting property* if $\alpha(s_1) \geq \alpha(s_2) \Rightarrow N(s_1) \supseteq N(s_2)$.

Definition 1.3. The objective of the *Ideal Mutual Orderings (IMO)* problem is to order the students and the questions so that they satisfy the interval and nesting 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 1.4. There is a polynomial time algorithm to solve IMO.

Proof. Compare the neighborhood of every pair of students $\{s_1, s_2\} \subseteq S$ and check whether $N(s_1) \subseteq N(s_2)$ or $N(s_1) \supseteq N(s_2)$. If $N(s_1) \cap N(s_2)$ is a strict subset of $N(s_1)$ and $N(s_2)$, then output NO. Now, assuming that every pair $\{s_1, s_2\} \subseteq S$ satisfies $N(s_1) \subseteq N(s_2)$ or $N(s_1) \supseteq N(s_2)$, we know that there is an ordering $\alpha : S \to [|S|]$ such that $\alpha(s_1) \leq \alpha(s_2) \Rightarrow N(s_2) \subseteq N(s_2)$. We easily find such an ordering by sorting the students according to their degrees, i.e., from lowest to highest degree, the students will receive labels from the smallest to the largest. Denote the resulting ordering by π. Since all neighborhoods are subsets or supersets of any other neighborhood and π was sorted by degree, $\pi(s_1) \leq \pi(s_2) \Rightarrow N(s_1) \leq N(s_2)$. So we have satisfied the nesting property.

To satisfy the interval property, we order the questions according to the nesting of the neighborhoods. Recall that we have $N(\pi^{-1}(1)) \subseteq \cdots \subseteq N(\pi^{-1}(|S|))$. Now, we order the questions so that whenever $q_1 \in N(\pi^{-1}(i))$ and $q_2 \in N(\pi^{-1}(j))$ with $i < j$, we have q_1 labeled smaller q_2 according to the ordering. We can do so by labeling the questions in $N(\pi^{-1}(1))$ the smallest numbers (the ordering within the set does not matter), then the questions in $N(\pi^{-1}(2))$ the next smallest, and so on. Call the resulting ordering β. Note that for all $s \in S$, $s = \pi^{-1}(i)$ for some i. So $N(s) = N(\pi^{-1}(i)) \supseteq N(\pi^{-1}(1))$, i.e., s correctly answers the easiest question according to β. Furthermore, $N(s)$ is a block of questions that are consecutive according to the ordering β. So the interval property is also satisfied.

To determine the run time, note that we made $O(n^2)$ comparisons of neighborhoods. Each set intersection of two neighborhoods took $O(n)$ time assuming that each neighborhood was stored as a sorted list of the questions (sorted by any fixed labeling of the questions). Ordering the students by degree took $O(n \log n)$ time and ordering the questions took $O(n)$ time. So the total run time is $O(n^2)$. □

Next, observe that the nesting 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 nesting property in subsequent variants of the problem.

Proposition 1.5. 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 nesting property.

Proof. First, we prove the forward direction. Assume that $G = (S \cup Q, E)$ satisfies the interval property with respect to the ordering β on Q. By definition of interval property, for every $u \in S$, we have $N(u) = \{\beta^{-1}(1), \ldots, \beta^{-1}(j)\}$ for some $j \in [|Q|]$. Then for every $u_1, u_2 \in S$, we have $N(u_1) \subseteq N(u_2)$ or $N(u_2) \subseteq N(u_1)$. Let α be an ordering of S by degree of each $u \in S$. Then the nesting property holds with respect to α.

Second, we prove the backward direction. Assume that $G = (S \cup Q, E)$ satisfies the nesting property with respect to α on S. Then $N(\alpha^{-1}(1)) \subseteq \cdots \subseteq N(\alpha^{-1}(|S|))$. Using the algorithm in the proof of Proposition 1.4 for IMO, we obtain an ordering β on Q so that the interval property holds with respect to β. □

Next, we define three variants of IMO, which model the possible ways we would allow changes to the edges in the graph in order to achieve the nesting property: allowing edges to be added, or deleted, or both.

Definition 1.6 *(Chain Editing (CE)).* 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 nesting 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 that they know how to solve but never answer a hard problem correctly by luck, e.g., in numerical entry questions.

Definition 1.7 *(Chain Addition (CA)).* 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 nesting property.

On the other hand, weak students may accidentally solve hard questions correctly when the questions are multiple choice or true/false. Chain Deletion models such situations.

Definition 1.8 *(Chain Deletion (CD)).* In the Chain Deletion (CD) problem, we are given a bipartite graph representing student-question relations and asked to find a minimum set of edge deletions that admits an ordering of the students satisfying the nesting property.

Among the three problems, Chain Addition and Chain Deletion are isomorphic, i.e., solving one enables us to solve the other. The key property that connects Chain Addition with Chain Deletion is that a graph satisfies the nesting property if and only if its complement satisfies the nesting property. To solve Chain Deletion on a graph G, consider the complement $\overline{G}$ of G and solve Chain Addition on $\overline{G}$. Let F be the set of edges in an optimal solution for Chain Addition on $\overline{G}$. By definition of complement, F must have been a subset of the edges in G. Since $\overline{G} \cup F$ satisfies the nesting property, its complement $\overline{\overline{G} \cup F} = G \setminus F$ must also satisfy the nesting property. So F is an optimal solution for Chain Deletion on G. A symmetric argument applies to solve Chain Addition from Chain Deletion. Since the addition and the deletion cases are isomorphic, we consider only the addition and the more general edition, which – together with the three constraint variants from subsection 1.1.3 – give rise to our 6 problem formulations.

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 1.9. 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.

Proof. Consider the special case that one side of the correct ordering is given to us, say the questions are given in hardest to easiest order $v_1 \geq \cdots \geq v_q$. Then we can find the minimum number of errors needed to satisfy the required conditions by correcting the edges incident to each student u individually.

We know by the interval property that every student u must correctly answer either a set of consecutive questions starting from v_1 or no questions at all. For each $u \in S$, and for each v_j, simply compute the number of edge edits required so that the neighborhood of u becomes $\{v_1, \ldots, v_j\}$. Select the question v_u that minimizes the cost of enforcing $\{v_1, \ldots, v_j\}$ to be the neighborhood of u. Once the edges are corrected, order the students by the containment relation of their neighborhoods.

The algorithm correctly calculates the minimum edge edits since the interval property was satisfied at the minimum cost possible per student. The algorithm finds the neighborhood of each student by trying at most $|Q| < n$ difficulty thresh-

olds v_j, and the cost of calculation for each threshold takes $O(1)$, by using the value calculated from the previous thresholds tried. Summing over the $|S| < n$ students gives a total running time no more than $O(n^2)$. □

1.2.2. k-near variants of Chain Editing or Addition

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 1.10 *(k-near CE or CA).* In the *k-near* problem, we are given an initial ordering $\alpha : S \to [|S|]$ and a nonnegative integer k. A *feasible* solution exhibits a set of edge edits (additions) attaining the nesting 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 consider problems where the question side is not allowed to be arbitrarily ordered.

Definition 1.11 *(Unconstrained k-near CE or CA).* 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 nesting property.

Definition 1.12 *(Constrained k-near CE or CA).* 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 nesting property and respects the interval property according to the given question ordering.

Definition 1.13 *(Both k-near CE or CA).* 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 nesting 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. Additionally, we provide an $O(kn)$ additive approximation algorithm for the NP-hard case. 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 1.9. If both sides are unconstrained, this is exactly the Chain Editing (or Addition) problem, which are both known to be NP-hard (see below). Fig. 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.

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 and an $O(kn)$ additive approximation 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. Since many results parameterize in terms of the value of an optimal solution to their problem, we use OPT to denote the optimal value, where the problem solved depends on the context. Before stating the results, we define a couple of problems closely related to Chain Addition. The

Questions \ Students	Unconstrained	k-near		Constrained
		Editing	Addition	
Unconstrained	NP-hard [26, 10]	NP-hard Thm 3.4, $\mathcal{O}(kn)$-approx Thm 3.5	$\mathcal{O}(n^3 2^{4k} k^{4k})$ Thm 3.3	$\mathcal{O}(n^2)$ Prop 1.9
k-near Editing	NP-hard Thm 3.4, $\mathcal{O}(kn)$-approx Thm 3.5	$\mathcal{O}(n^3 2^{8k} k^{8k+4})$ Thm 3.6		$\mathcal{O}(n^3 2^{4k} k^{4k+2})$ Thm 3.2
k-near Addition	$\mathcal{O}(n^3 2^{4k} k^{4k})$ Thm 3.3		$\mathcal{O}(n^3 2^{8k} k^{8k+4})$ Thm 3.7	
Constrained	$\mathcal{O}(n^2)$ Prop 1.9	$\mathcal{O}(n^3 2^{4k} k^{4k+2})$ Thm 3.2		$\mathcal{O}(n^2)$

Fig. 2. All variants of the decision version of the problems are shown with their respective complexities. The complexity of Unconstrained/Unconstrained Addition [26] and Editing [10] were derived before. More detailed results for these cases will be shown in Fig. 3. 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.

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] gave 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] claimed 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] gave 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] gave an FPT in time $O(2^{O(OPT)} + OPT^2 nm)$. Fomin and Villanger [12] showed the first subexponential FPT for Chordal Completion, in time $O(2^{O(\sqrt{OPT} \log OPT)} + OPT^2 nm)$. Cao and Marx [7] studied a generalization of Chordal Completion, where three operations are allowed: vertex deletion, edge addition, and edge deletion. There, they gave 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] showed an FPT in time $2^{O(\sqrt{OPT} \log OPT)} + \text{poly}(n)$, where $\text{poly}(n)$ represents a polynomial function with respect to n. They also showed the same result holds for a related problem called Threshold Editing.

On the other side, Drange et al. [10] showed 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] excluded the possibility of $2^{O(\sqrt{n}/\log n)}$ and $2^{O(OPT^{\frac{1}{4}}/\log^c OPT)} n^{O(1)}$ 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 excluded the possibility of a PTAS for Chordal Completion assuming $P \neq NP$. Wu et al. [25] showed that no constant approximation is possible for Chordal Completion assuming the Small Set Expansion Conjecture. Fig. 3 summarizes the known results for the aforementioned graph modification problems.

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. An abbreviated version of this paper appeared in the proceedings of the 11th International Conference and Workshops on Algorithms and Computation [16].

	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]

Fig. 3. This table shows existing results for the case that both sides are unconstrained, which are all known to be NP-hard from the upper left block of Fig. 2.

3. Polynomial time algorithms for k-near orderings

We present our polynomial time algorithm for the Constrained k-near Addition and Editing problems, the Both k-near Addition and Editing problems, and the Unconstrained k-near Addition problem. We also show the NP-hardness of the Unconstrained k-near Editing problem and provide a $O(kn)$ additive approximation algorithm for it.

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.

3.1. Constrained k-near

We will solve the Constrained k-near Editing and Addition problems in time $O(n^3 2^{4k}k^{4k+2})$ by dynamic programs. First, we will solve the Constrained k-near Editing problem. Then we modify the algorithm to solve the Constrained k-near Addition problem.

3.1.1. Constrained k-near Editing

Theorem 3.1 *(Constrained k-near Editing). Constrained k-near Editing can be solved in time $O(n^3 2^{4k}k^{4k+2})$.*

Proof. Assume that the students are given in k-near order $1,\dots,|S|$ and that the questions are given in exact order $1 \le \cdots \le |Q|$. 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 $\left[\max\{1, i-k\}, \min\{|S|, i+k-1\}\right] \setminus \{u_i\}$. Let $\boldsymbol{F_{i,u_i}} := \left\{\{\pi^{-1}(1),\dots,\pi^{-1}(k)\} : \pi \in P_{i,u_i}, \pi(a) \in [a-k, a+k], \forall a \in \left[\max\{1, i-k\}, \min\{|S|, i+k-1\}\right] \setminus \{u_i\}\right\}$. 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 $U_i \setminus \left[1, \max\{1, i-k-1\}\right]$ 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 $i \ge 2$, U_i already must include all of $\left[1, \max\{1, i-k-1\}\right]$ since any student initially at position $\le i-k-1$ cannot move beyond position $i-1$ in a feasible solution. If $i = 1$, we have $U_1 = \emptyset$. From now on, we assume $i \ge 2$ and treat the base case $i = 1$ at the end. So the set $U_i \setminus \left[1, \max\{1, i-k-1\}\right]$ will uniquely determine U_i. We know that U_i cannot include any students with initial label $[k+i, |S|]$ since students of labels $\ge k+i$ must be assigned to positions i or later. So the only uncertainty remaining is which elements in $\left[\max\{1, i-k\}, \min\{|S|, i+k-1\}\right] \setminus \{u_i\}$ make up the set $U_i \setminus \left[1, \max\{1, i-k-1\}\right]$. We may determine all possible candidates for $U_i \setminus \left[1, \max\{1, i-k-1\}\right]$ by trying all permutations of $\left[\max\{1, i-k\}, \min\{|S|, i+k-1\}\right] \setminus \{u_i\}$ 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 $\boldsymbol{S_i} = \left\{(u_i, U_i, v_{j_i}) : u_i \in \left[\max\{1, i-k\}, \min\{|S|, i+k\}\right], U_i \setminus \left[1, \max\{1, i-k-1\}\right] \in F_{i,u_i}, v_{j_i} \in Q \cup \{0\}\right\}$. 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 $\boldsymbol{R_{i-1,u_i,U_i,v_{j_i}}} := \{(u_{i-1}, U_{i-1}, v_{j_{i-1}}) \in S_{i-1} : v_{j_{i-1}} \le v_{j_i}, U_i = \{u_{i-1}\} \cup U_{i-1}\}$. 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

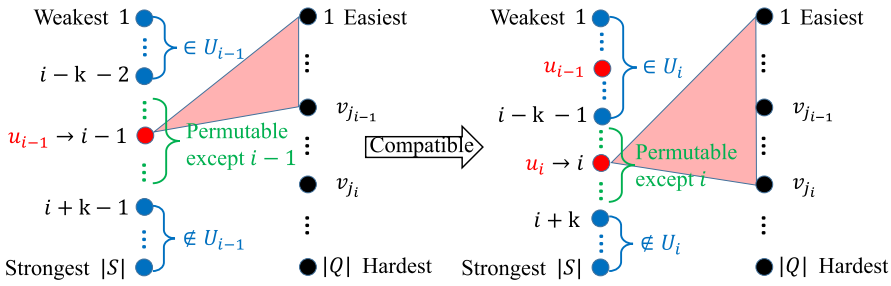

Fig. 4. Subproblem $(i-1, u_{i-1}, U_{i-1}, v_{j_{i-1}})$ is compatible with subproblem (i, u_i, U_i, v_{j_i}) if and only if $v_{j_{i-1}}$ is no harder than v_{j_i} and $U_i = \{u_{i-1}\} \cup U_{i-1}$. The cost of (i, u_i, U_i, v_{j_i}) is the sum of the minimum cost among feasible compatible subproblems of the form $(i-1, u_{i-1}, U_{i-1}, v_{j_{i-1}})$ and the number of edits incident to u_i to make its neighborhood exactly $\{1, \ldots, v_{j_i}\}$.

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. 4.

The dynamic program. Finally, we define $\boldsymbol{c_{u_i,v_{j_i}}}$ to be the 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 $(2k)! = O(2^{2k}k^{2k})$ 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 $O(k \cdot 2^{2k}k^{2k}k^2 \cdot |Q|)$ 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 $O(|S|k^3 2^{2k}k^{2k}|Q|) = O(n^2 2^{2k}k^{2k+3})$.

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 $O(\sum_{i \in S, (u_i,U_i,v_{j_i}) \in S_i} |R_{i-1,u_i,U_i,v_{j_i}}|) = O(|S||S_i||S_{i-1}|) = O(n(k \cdot 2^{2k}k^{2k} \cdot n)^2) = O(n^3 2^{4k}k^{4k+2})$. □

3.1.2. Constrained k-near Addition

We use the same framework as Constrained k-near Editing to solve the Constrained k-near Addition. We change the definitions of the subproblem, the relevant sets, and the costs appropriately to adapt to the Addition problem.

Theorem 3.2 *(Constrained k-near Addition). Constrained k-near Addition can be solved in time* $O(n^3 2^{4k}k^{4k+2})$.

Proof. First, redefine $\boldsymbol{C(i, u_i, U_i, v_{j_i})}$ to be the smallest cost of adding edges incident to the weakest i positions so 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.

The sets $\boldsymbol{P_{i,u_i}}$ and $\boldsymbol{F_{i,u_i}}$ will stay the same as before. We redefine $\boldsymbol{S_i} := \Big\{(u_i, U_i, v_{j_i}) : u_i \in \big[\max\{1, i-k\}, \min\{|S|, i+k\}\big], U_i \setminus \big[1, \max\{1, i-k-1\}\big] \in F_{i,u_i}, v_{j_i} \in Q \cup \{0\}, v_{j_i} \ge \max N_G(\{u_i\} \cup U_i)\Big\}$. Requiring that v_{j_i} is at least as hard as $N_G(\{u_i\} \cup U_i)$ ensures that the final solution will satisfy the interval property with respect to the given question order. It was not needed in the Editing problem because wherever v_{j_i} landed, the edges that reach questions harder than v_{j_i} were deleted. The definition of $\boldsymbol{R_{i-1,u_i,U_i,v_{j_i}}}$ will stay the same as before, but using the new definition of S_{i-1} from this section. Finally, the cost $\boldsymbol{c_{u_i,v_{j_i}}}$ will become the number of edge additions incident to u_i so that the neighborhood of u_i becomes $\{1, \ldots, v_{j_i}\}$, i.e., $c_{u_i,v_{j_i}} := |\{1, \ldots, v_{j_i}\} \setminus N_G(u_i)|$.

The recurrence relation from Constrained k-near Editing still applies here. However, the base cases become $C(1, u_1, U_1, v_{j_1}) = |\{1, \ldots, v_{j_1}\} \setminus N_G(u_1)|$ if $v_{j_1} > 0$, and $C(1, u_1, U_1, v_{j_1}) = 0$ if $v_{j_1} = 0$.

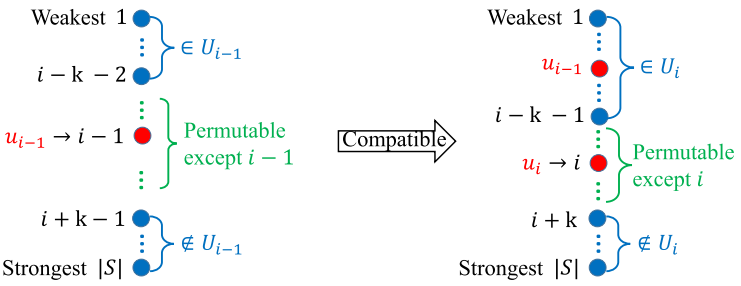

Fig. 5. Subproblem $(i-1, u_{i-1}, U_{i-1})$ is compatible with subproblem (i, u_i, U_i) if and only if $U_i = \{u_{i-1}\} \cup U_{i-1}$. The cost of (i, u_i, U_i) is sum of the minimum cost among feasible compatible subproblems of the form $(i-1, u_{i-1}, U_{i-1})$ and the number of additions incident to u_i to make its neighborhood the smallest set of questions containing the existing neighbors of U_i.

The run time is still dominated by the dynamic program since the time to construct S_i becomes only $|Q|$ times larger (to enforce the additional constraint that v_{j_i} is hard enough). Hence the total time to solve this problem remains $O(n^3 2^{4k} k^{4k+2})$. □

3.2. Unconstrained k-near

First, we solve the Unconstrained k-near Addition problem in time $O(n^3 2^{4k} k^{4k})$. Second, we show that the Unconstrained k-near Editing problem is NP-hard.

Assume that the students are given in k-near order $1, \ldots, |S|$. The questions are allowed to be ordered arbitrarily in the final solution.

3.2.1. Unconstrained k-near Addition

Theorem 3.3 *(Unconstrained k-near Addition). Unconstrained k-near Addition can be solved in time* $O(n^3 2^{4k} k^{4k})$.

Proof. We introduce subproblems of the form (i, u_i, U_i). Define $\boldsymbol{C(i, u_i, U_i)}$ to be the smallest number of edges incident to the weakest i positions that must be added so that u_i is in position i and U_i is the set of the $i-1$ weakest students.

We use the same $\boldsymbol{P_{i,u_i}}$ and $\boldsymbol{F_{i,u_i}}$ as defined for Constrained k-near Editing to bound the search space for U_i given that u_i is in position i. Define $\boldsymbol{S_i} := \left\{(u_i, U_i) : u_i \in \left[\max\{1, i-k\}, \min\{|S|, i+k\}\right], U_i \setminus [1, \max\{1, i-k-1\} \in F_{i,u_i}\right\}$.

Next, define $\boldsymbol{R_{i-1,u_i,U_i}} := \left\{(u_{i-1}, U_{i-1}) \in S_{i-1} : U_i = \{u_{i-1}\} \cup U_{i-1}\right\}$. The set R_{i-1,u_i,U_i} ensures that the smaller subproblems have prefixes that are compatible with those assigned in the bigger subproblems they came from. Compatibility is illustrated in Fig. 5.

Lastly, define $\boldsymbol{c_{u_i,U_i}}$ to be the number of edge additions incident to u_i so that the neighborhood of u_i becomes the smallest set of questions containing $N_G(U_i \cup \{u_i\})$, i.e., $c_{u_i,U_i} := |N_G(U_i \cup \{u_i\}) \setminus N_G(u_i)|$.

Using the above definitions, we have the following recurrence:

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

The base cases are $C(1, u_1, U_1) = |N_G(U_1) \setminus N_G(u_1)|$ for all $(u_1, U_1) \in S_1$, since u_1 must add edges to the questions that the weaker students correctly answered.

The final solution to Unconstrained k-near Addition is $\min_{(u_{|S|}, U_{|S|}) \in S_{|S|}} C(|S|, u_{|S|}, U_{|S|})$.

To bound the run time, note that generating S_i takes $O(n \cdot 2^{2k} k^{2k} k^2)$ time. The dynamic program will dominate the run time again. In the dynamic program, each subproblem (i, u_i, U_i) takes $O(|R_{i-1,u_i,U_i}|)$ time. So the total time is $O(\sum_{i \in S, (u_i,U_i) \in S_i} |R_{i-1,u_i,U_i}|) = O(|S||S_i||S_{i-1}|) = O(n(n2^{2k} k^{2k})^2) = O(n^3 2^{4k} k^{4k})$. □

3.2.2. Unconstrained k-near Editing

The Unconstrained k-near Editing problem is NP-hard even for $k = 1$. We closely follow the proof of Drange et al. [9] for the NP-hardness of Threshold Editing to show that Unconstrained k-near Editing is NP-hard. In Drange et al.'s construction, they specified a partial order for which the cost of Threshold Editing can only worsen if the output ordering deviates from it. We crucially use this property to prove NP-hardness for Unconstrained 1-near Editing.

Theorem 3.4 *(Unconstrained k-near Editing). Unconstrained k-near Editing is NP-hard.*

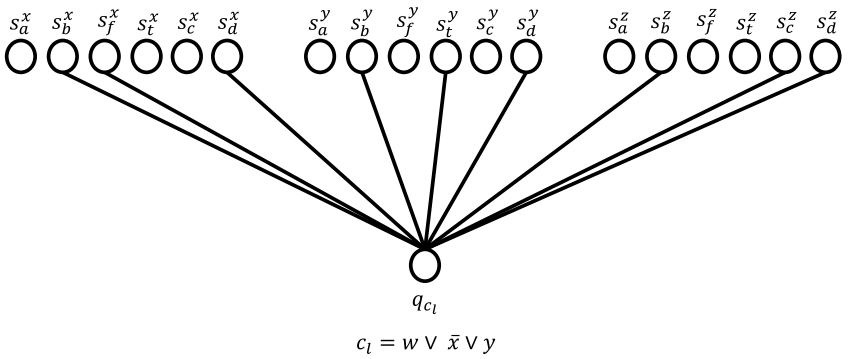

Fig. 6. Each set of six vertices represents the students corresponding to a variable x, y, or z. The bottom vertex represents a question corresponding to the clause $c_l = w \vee \bar{x} \vee y$.

Proof. Let $G = (S, Q, E)$ be a bipartite graph with initial student ordering π. Consider the decision problem Π of determining whether there is a 1-near unconstrained editing of at most t edges for the instance (G, π). We reduce from 3-SAT to Π. Let Φ be an instance for 3-SAT with clauses $C = \{c_1, \ldots, c_m\}$ and variables $V = \{v_1, \ldots, v_n\}$. We construct the corresponding instance $\Pi = (G_\Phi, \pi_\Phi, t_\Phi)$ for 1-near unconstrained editing as follows. First we order the variables in an arbitrary order and use this order to define π. For each variable v_i, create six students $s_a^i, s_b^i, s_f^i, s_t^i, s_c^i, s_d^i$. Next, we define a partial ordering P that the initial order π_Φ shall obey. Define P to be the partial order satisfying $s_a^i > s_b^i > s_f^i, s_t^i > s_c^i > s_d^i$ for all $i \in [n]$ and $s_\alpha^i > s_\beta^j$ for all $i < j$, $\alpha, \beta \in \{a, b, c, d, f, t\}$. Define π_Φ to be the linear ordering satisfying all relations of P for the variables in the initial arbitrary order, and additionally $s_f^i > s_t^i$. We remark that the proof works regardless of whether we set $s_f^i > s_t^i$ or $s_f^i < s_t^i$ in π_Φ. We shall impose that optimal solutions satisfy all of the relations of P. To do so, for every $s > s'$, we add $t_\Phi + 1$ new questions each with edges to s and no edges to s', and with edges to all $r > s$ in π_Φ. Then whenever an editing solution switches the order of s and s', it must edit at least $t_\Phi + 1$ edges. After adding the necessary questions to ensure feasible solutions must preserve the partial order P, we create a question q_{c_l} for each clause c_l. If a variable v_i appears positively in c_l, then add the edge $q_{c_l} s_t^i$. If v_i appears negatively in c_l, then add the edge $q_{c_l} s_f^i$. If v_i does not occur in c_l, then add the edge $q_{c_l} s_c^i$. For all variables v_i and clauses c_l, add the edges $q_{c_l} s_b^i$ and $q_{c_l} s_d^i$. Finally, define $t_\Phi = |C|(3|V| - 1)$. Refer to Fig. 6 for an illustration of the construction.

Now, we show that there is a satisfying assignment if and only if there is a 1-near editing of at most t_Φ edges. First, we prove the forward direction. Assume there is a satisfying assignment $f : V \to \{T, F\}$. Let c_l be a clause. One of the literals v_i in c_l is set to T under the assignment f. If v_i occurs positively, then edit the neighborhood of q_{c_l} to be all students s such that $s \geq s_t^i$ according to P and impose $s_t^i > s_f^i$ in the solution. If v_i occurs negatively in q_{c_l}, then edit the neighborhood of q_{c_l} to be all students s such that $s \geq s_f^i$ and keep the initial order that $s_f^i > s_t^i$. In both cases, the neighborhood of q_{c_l} changed by 2 among the six students corresponding the variable v_i and changed by 3 for the remaining groups of six students. So the number of edge edits incident to each (clause) question is $3|V| - 1$. Note that the neighborhoods of the extra questions we added to impose P are already nested because each time a new question was added, it received edges to all students who are stronger than a particular student according to P. So only the questions that came from clauses potentially need to edit their neighborhoods to achieve nesting. Hence, the total number of edge edits is $|C|(3|V| - 1) = t_\Phi$.

Second, we prove the backward direction. Assume there is an unconstrained 1-near editing of $|C|(3|V| - 1)$ edges to obtain a chain graph. Let c_l be a clause. For any variable v_j not occurring in c_l, the original edges that q_{c_l} has to the six students corresponding to v_j are to s_b^j, s_c^j, s_d^j. If the cut-off point of the edited neighborhood of q_{c_l} is among $s_a^j, s_b^j, s_f^j, s_t^j, s_c^j, s_d^j$, then the edges incident to q_{c_l} must change by at least three among those six, which means that q_{c_l} would have at least $3|V|$ edges incident to it. If the cut-off point of the edited neighborhood of q_{c_l} is among the six students corresponding to a variable v_i that occurs in c_l, then the edges incident to q_{c_l} must change by at least two (by switching the order of s_f^i and s_t^i when needed) among those six students and at least three for the students corresponding to the remaining variables. Thus q_{c_l} has at least $3|V| - 1$ edges edits incident to it for every c_l. So the smallest number of edge edits possible is at least $|C|(3|V| - 1)$. By the assumption, G_Φ has a feasible editing of at most $|C|(3|V| - 1)$ edges. Then each q_{c_l} must have exactly $3|V| - 1$ edits incident to it. So the cut-off point for the edited neighborhood of each q_{c_l} must occur among the six students corresponding to a variable v_i occurring inside c_l. If the occurring variable v_i is positive, then the cut-off point must have been at s_t^i and required $s_t^i > s_f^i$ since all other cut-offs incur at least three edits. Similarly, if v_i is negative, then the cut-off point must have been at s_f^i and required $s_f^i > s_t^i$. All clauses must be consistent in their choice of the ordering between s_f^i and s_t^i for all $i \in [n]$ since the editing solution was feasible. Hence, we obtain a satisfying assignment by setting each variable v_i true if and only if $s_t^i > s_f^i$. □

Next, we show a simple $O(kn)$ additive approximation algorithm for Unconstrained k-near Editing.

Theorem 3.5 *(Approximation for Unconstrained k-near Editing). Unconstrained k-near Editing has an $O(kn)$ additive approximation algorithm.*

Proof. Fix the student side to the initial ordering $\sigma : S \to [|S|]$ given for the k-near condition and solve the corresponding Constrained Unconstrained Editing problem exactly. Denote by F the edge edits found from solving the Constrained Unconstrained Editing problem. Let σ^* be the ordering for S in an optimal solution to the original k-near Unconstrained problem. Let H be the minimum size edge edits corresponding to σ^*. It suffices to show that for each $q \in Q$, $|N_F(q)| - |N_H(q)| \leq 2k-2$, since this inequality would imply that $|F| - |H| \leq (2k-2)|Q| \leq 2kn$.

For $q \in Q$, let $p(q)$ be the position of the weakest student who answers q correctly according to the ordering σ^*. By the k-near condition, any student more than $k-1$ positions after $p(q)$ cannot be ordered before $p(q)$ and vice versa. If $p(q)$ remains the position of the weakest student who correctly answers q according to the ordering σ, then the edge edits required would be the same as H, except for possibly those edges from q to students who are within $k-1$ positions of $p(q)$. For each q, F is determined by choosing the cut-off position for the neighborhood of q that minimizes the number of edits needed. Then $N_F(q)$ should differ from $N_H(q)$ no more than the case where the cut-off point for q stays the same position as $p(q)$. So $|N_F(q)| - |N_H(q)| \leq 2(k-1)$. Hence $|F| - |H| = O(kn)$. □

3.3. Both k-near

We will solve the Both k-near Editing and Addition problems in time $O(n^3 2^{8k} k^{8k+4})$. We first show our solution for the Editing problem and then adapt it to the Addition problem.

Assume that the students and questions are both given in k-near order with student labels $1, \ldots, |S|$, and question labels $1, \ldots, |Q|$.

3.3.1. Both k-near Editing

Theorem 3.6 *(Both k-near Editing). Both k-near Editing can be solved in time $O(n^3 2^{8k} k^{8k+4})$.*

Proof. We consider subproblems of the form $(i, u_i, U_i, j_i, v_{j_i}, V_{j_i})$. Define $\boldsymbol{C(i, u_i, U_i, j_i, v_{j_i}, V_{j_i})}$ to be the smallest number of edges incident to the weakest i students that must be edited so that student u_i is in position i, U_i is the set of the $i-1$ weakest students, j_i is the position of the hardest question correctly answered by $U_i \cup \{u_i\}$, v_{j_i} is the question in position j_i, and V_{j_i} is the set of the $j_i - 1$ easiest questions.

Feasible and compatible subproblems. Next, we define the search space for $(u_i, U_i, j_i, v_{j_i}, V_{j_i})$ given that u_i is in position i. We use the same $\boldsymbol{P_{i,u_i}}$ and $\boldsymbol{F_{i,u_i}}$ defined in the proof for Constrained k-near Editing. Define $\boldsymbol{S_i} := \Big\{(u_i, U_i, j_i, v_{j_i}, V_{j_i}) : u_i \in \big[\max\{1, i-k\}, \min\{|S|, i+k\}\big], U_i \setminus \big[1, \max\{1, i-k-1\}\big] \in F_{i,u_i}, v_{j_i} \in \big[\max\{1, j_i - k\}, \min\{|Q|, j_i + k\}\big], V_{j_i} \setminus \big[1, \max\{1, j_i - k - 1\}\big] \in F_{j_i, v_{j_i}}\Big\}$. Here, we need to constrain both the student side and the question side to make sure that all elements are k-near as opposed to only enforcing the k-nearness on the students in Constrained k-near Editing.

To bound the search space for subproblems to be compatible with the bigger subproblems they came from, we define $\boldsymbol{R_{i-1,u_i,U_i,j_i,v_{j_i},V_{j_i}}} := \Big\{(u_{i-1}, U_{i-1}, j_{i-1}, v_{j_{i-1}}, V_{j_{i-1}}) \in S_{i-1} : U_i = U_{i-1} \cup \{u_{i-1}\}, j_i \geq j_{i-1}, V_{j_i} \cup \{v_{j_i}\} \supseteq V_{j_{i-1}} \cup \{v_{j_{i-1}}\}, j_i > j_{i-1} \Rightarrow V_{j_i} \supseteq V_{j_{i-1}} \cup \{v_{j_{i-1}}\}\Big\}$. The constraints in the set $R_{i-1,u_i,U_i,j_i,v_{j_i},V_{j_i}}$ ensure that the prefixes of position i and position j_i in the smaller subproblem will be compatible with the bigger subproblem that it came from. Furthermore, $j_i \geq j_{i-1}$ ensures that stronger students correctly answer all questions that weaker students correctly answered. We demonstrate compatibility in Fig. 7.

The dynamic program. Finally, define $\boldsymbol{c_{u_i, v_{j_i}, V_{j_i}}}$ to be the number of edge edits incident to u_i so that the neighborhood of u_i becomes exactly $V_{j_i} \cup \{v_{j_i}\}$, i.e., $c_{u_i, v_{j_i}, V_{j_i}} := |N_G(u_i) \triangle V_{j_i} \cup \{v_{j_i}\}|$.

Using the above definitions, we obtain the following recurrence.

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

The base cases are $C(1, u_1, U_1, j_1, v_{j_1}, V_{j_1}) = |N_G(u_1) \triangle \{v_{j_1}\} \cup V_{j_1}|$ for all $(u_1, U_1, j_1, v_{j_1}, V_{j_1}) \in S_1$.

The final solution is $\min_{(u_{|S|}, U_{|S|}, j_{|S|}, v_{j_{|S|}}, V_{j_{|S|}}) \in S_{|S|}} C(|S|, u_{|S|}, U_{|S|}, j_{|S|}, v_{j_{|S|}}, V_{j_{|S|}})$.

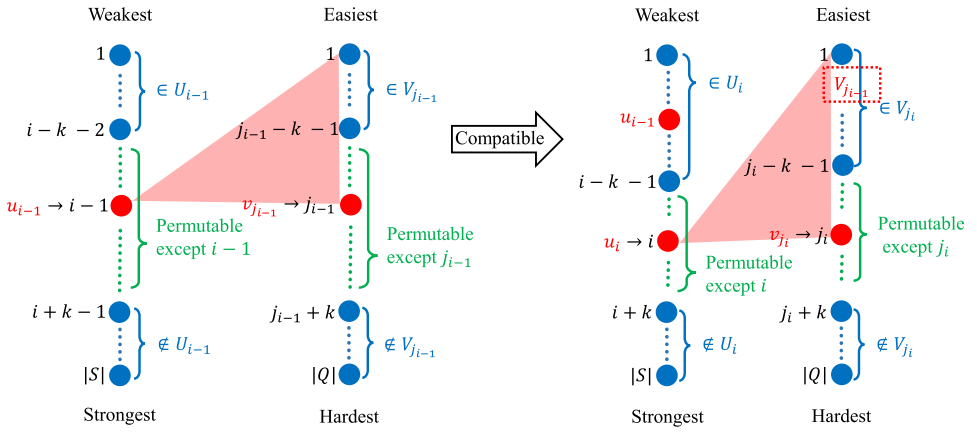

Fig. 7. Subproblem $(i-1, u_{i-1}, U_{i-1}, j_{i-1}, v_{j_{i-1}}, V_{j_{i-1}})$ is compatible with subproblem $(i, u_i, U_i, j_i, v_{j_i}, V_{j_i})$ if and only if $U_i = \{u_i\} \cup U_{i-1}$, j_{i-1} represents a position no harder than j_i, $V_{j_i} \cup \{v_{j_i}\}$ contains $V_{j_{i-1}} \cup \{v_{j_{i-1}}\}$, and j_{i-1} strictly easier than j_i implies that V_{j_i} contains $V_{j_{i-1}} \cup \{v_{j_{i-1}}\}$. The cost of $(i, u_i, U_i, j_i, v_{j_i}, V_{j_i})$ is the sum of the minimum cost among feasible compatible states of the form $(i-1, u_{i-1}, U_{i-1}, j_{i-1}, v_{j_{i-1}}, V_{j_{i-1}})$ and the number of edits incident to u_i that makes its neighborhood $V_{j_i} \cup \{v_{j_i}\}$.

Running time. First, observe that $|S_i| = O(k^2 2^{4k} k^{4k} |Q|)$, since there are $O(k)$ choices for u_i and v_i, $O(2^{2k}k^{2k})$ choices for U_i and V_{j_i}, and $|Q|$ choices for j_i. To build S_i, we need to build F_{i,u_i} and $F_{j_i,v_{j_i}}$. In Section 3, we saw that each of the F_{i,u_i} takes $O(k^2 2^{2k} k^{2k})$ time to build. Then building the set S_i is upper bounded by $O(k \cdot 2^{2k}k^{2k}k^2 \cdot |Q| \cdot k \cdot 2^{2k}k^{2k}k^2)$ per i, where we are over-counting the time to generate all possible U_i and V_{j_i} by the time it takes to build F_{i,u_i} and $F_{j_i,v_{j_i}}$. Building the set $R_{i-1,u_i,U_i,j_i,v_{j_i},V_{j_i}}$ while building S_i will take $O(|S| + |Q|)$ to check the conditions that restrict S_{i-1} to $R_{i-1,u_i,U_i,j_i,v_{j_i},V_{j_i}}$. Due to the size of S_i, the construction of sets will still be dominated by the time to solve the dynamic program. Specifically, each subproblem $(i, u_i, U_i, j_i, v_{j_i}, V_{j_i})$ takes $O(|R_{i-1,u_i,U_i,j_i,v_{j_i},V_{j_i}}|)$ time. So the total time is $O(\sum_{i\in S,(u_i,U_i,j_i,v_{j_i},V_{j_i})\in S_i} |R_{i-1,u_i,U_i,j_i,v_{j_i},V_{j_i}}|) = O(|S||S_i||S_{i-1}|) = O(n(k^2 \cdot 2^{4k}k^{4k}n)^2) = O(n^3 2^{8k}k^{8k+4})$. □

3.3.2. Both k-near Addition

To solve the Addition version, we apply the method from the solution for Both k-near Editing.

Theorem 3.7 *(Both k-near Addition). Both k-near Addition can be solved in time* $O(n^3 2^{8k}k^{8k+4})$.

Proof. We redefine $\boldsymbol{C(i, u_i, U_i, j_i, v_{j_i}, V_{j_i})}$ to be the smallest number of edges incident to the weakest i students that must be added so that student u_i is in position i, U_i is the set of the $i-1$ weakest students, j_i is the position of the hardest question correctly answered by $U_i \cup \{u_i\}$, v_{j_i} is the question in position j_i, and V_{j_i} is the set of the $j_i - 1$ easiest questions.

We keep $\boldsymbol{P_{i,u_i}}$ and $\boldsymbol{F_{i,u_i}}$ the same as in the proof for Constrained k-near Editing. Redefine $\boldsymbol{S_i} := \Big\{(u_i, U_i, j_i, v_{j_i}, V_{j_i}) : u_i \in \big[\max\{1, i-k\}, \min\{|S|, i+k\}\big], U_i \setminus \big[1, \max\{1, i-k-1\}\big] \in F_{i,u_i}, v_{j_i} \in \big[\max\{1, j_i - k\}, \min\{|Q|, j_i + k\}\big], V_{j_i} \setminus \big[1, \max\{1, j_i - k - 1\}\big] \in F_{j_i,v_{j_i}}, V_{j_i} \cup \{v_{j_i}\} \supseteq N_G(\{u_i\} \cup U_i)\Big\}$. The addition constraint $V_{j_i} \cup \{v_{j_i}\} \supseteq N_G(\{u_i\} \cup U_i)$ is added here to ensure that the interval property induced by the current student ordering is satisfied every step. It was not needed in section 3.3.1 because existing edges to questions outside $V_{j_i} \cup \{v_{j_i}\}$ could be deleted. The definition of $\boldsymbol{R_{i-1,u_i,U_i,j_i,v_{j_i},V_{j_i}}}$ remains the same as section 3.3.1, but using the newly defined S_{i-1}. Lastly, redefine $\boldsymbol{c_{u_i,v_{j_i},V_{j_i}}}$ to be the number of edge additions incident to u_i so that the neighborhood of u_i becomes exactly $V_{j_i} \cup \{v_{j_i}\}$, i.e., $c_{u_i,v_{j_i},V_{j_i}} := |V_{j_i} \cup \{v_{j_i}\} \setminus N_G(u_i)|$.

The general recurrence relation of Section 3.3.1 stays the same. The base cases change to $C(1, u_1, U_1, j_1, v_{j_1}, V_{j_1}) = |\{v_{j_1}\} \cup V_{j_1} \setminus N_G(u_1)|$, with the convention that $j_1 = 0$ means $V_{j_1} = \emptyset$ and v_{j_1} is omitted from the count $|\{v_{j_1}\} \cup V_{j_1}|$.

Although the time to construct S_i is larger by a factor of $|Q|$, the total run time is dominated by the dynamic program, which takes $O(n^3 2^{8k}k^{8k+4})$. □

It is possible that the above running times for the five "easy" problems could improve. Our dynamic programs are designed based on the intuitiveness of the states and not necessarily optimized for time complexity.

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 a 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 and an $O(kn)$ additive approximation for the remaining one.

Some open questions still remain for the NP-hard problems in Fig. 2. For Chain Editing when both sides are unconstrained, there are no known approximation algorithms. For the corresponding Chain Addition problem, can a constant approximation can be achieved? For the Unconstrained k-near Editing problem, can the $O(kn)$ additive approximation be improved?

Acknowledgements

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

References

[1] A. Agrawal, P. Klein, R. Ravi, Cutting down on fill using nested dissection: provably good elimination orderings, in: Graph Theory and Sparse Matrix Computation, Springer, 1993, pp. 31–55.
[2] R. Andersen, C. Borgs, J. Chayes, U. Feige, A. Flaxman, A. Kalai, V. Mirrokni, M. Tennenholtz, Trust-based recommendation systems: an axiomatic approach, in: WWW, ACM, 2008, pp. 199–208.
[3] B. Aydin, Y. Yilmaz, Y. Li, Q. Li, J. Gao, M. Demirbas, Crowdsourcing for multiple-choice question answering, in: IAAI, 2014, pp. 2946–2953.
[4] E. Balas, N. Simonetti, Linear time dynamic-programming algorithms for new classes of restricted TSPs: a computational study, INFORMS J. Comput. 13 (2000) 56–75.
[5] D.P. Bertsekas, Non-linear Programming, Athena Scientific, 1999.
[6] I. Bliznets, M. Cygan, P. Komosa, L. Mach, M. Pilipczuk, Lower bounds for the parameterized complexity of minimum fill-in and other completion problems, in: SODA, 2016, pp. 1132–1151.
[7] Y. Cao, D. Marx, Chordal editing is fixed-parameter tractable, Algorithmica 75 (2016) 118–137.
[8] Y. Cao, R.B. Sandeep, Minimum fill-in: inapproximability and almost tight lower bounds, CoRR, arXiv:1606.08141, 2016.
[9] X.L. Dong, L. Berti-Equille, D. Srivastava, Integrating conflicting data: the role of source dependence, Proc. VLDB Endow. 2 (2009) 550–561.
[10] P.G. Drange, M.S. Dregi, D. Lokshtanov, B.D. Sullivan, On the threshold of intractability, in: ESA, 2015, pp. 411–423.
[11] T. Feder, H. Mannila, E. Terzi, Approximating the minimum chain completion problem, Inform. Process. Lett. 109 (2009) 980–985.
[12] F.V. Fomin, Y. Villanger, Subexponential parameterized algorithm for minimum fill-in, in: SODA, 2012, pp. 1737–1746.
[13] A. Galland, S. Abiteboul, A. Marian, P. Senellart, Corroborating information from disagreeing views, in: WSDM, ACM, 2010, pp. 131–140.
[14] W. Gatterbauer, D. Suciu, Data conflict resolution using trust mappings, in: SIGMOD, 2010, pp. 219–230.
[15] M. Gupta, J. Han, Heterogeneous network-based trust analysis: a survey, SIGKDD Explor. Newsl. 13 (2011) 54–71.
[16] Y. Jiao, R. Ravi, W. Gatterbauer, Algorithms for automatic ranking of participants and tasks in an anonymized contest, in: WALCOM, 2017, pp. 335–346.
[17] H. Kaplan, R. Shamir, R.E. Tarjan, Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs, SIAM J. Comput. 28 (1999) 1906–1922.
[18] J.M. Kleinberg, Authoritative sources in a hyperlinked environment, J. ACM 46 (1999) 604–632.
[19] Q. Li, Y. Li, J. Gao, B. Zhao, W. Fan, J. Han, Resolving conflicts in heterogeneous data by truth discovery and source reliability estimation, in: SIGMOD, 2014, pp. 1187–1198.
[20] Y. Li, J. Gao, C. Meng, Q. Li, L. Su, B. Zhao, W. Fan, J. Han, A survey on truth discovery, SIGKDD Explor. Newsl. 17 (2015) 1–16.
[21] A. Natanzon, R. Shamir, R. Sharan, A polynomial approximation algorithm for the minimum fill-in problem, SIAM J. Comput. 30 (2000) 1067–1079.
[22] J. Pasternack, D. Roth, Knowing what to believe (when you already know something), in: COLING, 2010, pp. 877–885.
[23] J. Pasternack, D. Roth, Latent credibility analysis, in: WWW, 2013, pp. 1009–1021.
[24] J. Pasternack, D. Roth, V.V. Vydiswaran, Information trustworthiness, in: AAAI Tutorial, 2013.
[25] Y.L. Wu, P. Austrin, T. Pitassi, D. Liu, Inapproximability of treewidth, one-shot pebbling, and related layout problems, J. Artificial Intelligence Res. 49 (2014) 569–600.
[26] M. Yannakakis, Computing the minimum fill-in is NP-complete, SIAM J. Algebr. Discrete Methods 2 (1981) 77–79.
[27] X. Yin, J. Han, P.S. Yu, Truth discovery with multiple conflicting information providers on the web, IEEE Trans. Knowl. Data Eng. 20 (2008) 796–808.