On the Tradeoff between Stability and Fit

Edith Cohen [*] Graham Cormode [†] Nick Duffield [‡] Carsten Lund [§]

Abstract

In computing, as in many aspects of life, changes incur cost. Many optimization problems are formulated as a one-time instance starting from scratch. However, a common case that arises is when we already have a set of prior assignments, and must decide how to respond to a new set of constraints, given that each change from the current assignment comes at a price. That is, we would like to maximize the fitness or efficiency of our system, but we need to balance it with the changeout cost from the previous state.

We provide a precise formulation for this tradeoff and analyze the resulting *stable extensions* of some fundamental problems in measurement and analytics. Our main technical contribution is a stable extension of Probability Proportional to Size (PPS) weighted random sampling, with applications to monitoring and anomaly detection problems. We also provide a general framework that applies to top-k, minimum spanning tree, and assignment. In both cases, we are able to provide exact solutions, and discuss efficient incremental algorithms that can find new solutions as the input changes.

1 Introduction

Most textbook optimization problems have a clean statement: given a set of constraints and an optimization target, we are free to choose any feasible solution, and so can strive for the optimal result. However, when applying these techniques, we may find that we do not begin with a clean slate. Rather, we often have some (partial) assignment, perhaps resulting from a previous optimization over a prior input. This assignment is unlikely to be optimal for the current instance, but each modification to the current assignment can come with a cost. We therefore have to balance the improvement from a new assignment (the fit) with the cost of changing from the current assignment to the new one (the stability).

Consider a network operator who is leasing network connections to meet customer demand. When there are no existing leases, this can be modeled as a standard graph optimization problem. But after an initial solution is found, customer demand may change. The operator then has to solve a more complex optimization, since there is a (fixed) cost to breaking an existing lease and establishing a new one. This cannot easily be represented as an instance of the original problem, due to the mixture of pricing schemes.

Formalizing this setting, we target applications where the *input state* $x \in \mathcal{X}$ changes over time. We maintain an *output* $S \in \mathcal{S}$ that can be changed in response to changes in the input. We are interested in the tradeoff between the *fit* (or *efficiency*) of the output with respect to the current input and the *changeout cost* from the previous output. This requirement for 'stability' arises in a variety of different applications:

[*] Google Research, Mountain View, CA, USA and Tel Aviv University, Israel edith@cohenwang.com

[†] University of Warwick, UK g.cormode@warwick.ac.uk . Supported in part by European Research Council grant ERC-2014-CoG 647557 and a Royal Society Wolfson Research Merit Award.

[‡] Texas A&M University, TX, USA duffieldng@tamu.edu

[§] AT&T Labs-Research, NJ, USA. lund@research.att.com

Human-interpretable output: In many settings devoted to monitoring and managing a system, there is a "dashboard" that presents information to an operator. For example, this could be details of the most congested links in a network, the heaviest users of the system, locations with anomalous activity, and so on. The underlying information may be rapidly changing, so that constantly updating the display would lead to a confusing mess of information. Instead, the operator should be presented with a view which is reasonably stable, so that they can absorb the information, and track the change in properties of items over time. At the same time, the output should be close to optimal in detailing the most important items.

The overhead of change: In a client-server assignment or a routing problem, changes in the output may correspond to cache swaps, job migration, or modifications to a routing table. In all of these cases, each change incurs a cost, and very rapid change can lead to poor performance: delays, due to routes changing midway, or low throughput due to constant reassignment of machines. We are therefore willing to settle for a solution that is sub-optimal at any instant, but which changes only slowly.

Audience-aware advertising: Consider a roadside electronic billboard that can respond to the set of road users passing by. We can choose an advertisement to display in response to demographics of the users, but constantly changing the ad will make it impossible for any of them to absorb.

Allocating resources to monitoring: Analysis such as change or anomaly detection requires tracking behaviour of items over time. In high-throughput systems, it is feasible to collect detailed statistics on only a representative subset of active items. Picking the items to monitor that are currently the most 'interesting' (top-k or a random sample) may fail to keep any around long enough to build sufficient history for particular items, so we prefer to retain monitored items while they are useful. Moreover, to efficiently store historic summaries for an extended time period, we only need to record the modifications to the summary made over time, so limiting modifications means that less storage is required.

Overview of contributions

We formalize and motivate the algorithmic problem of balancing fitness and stability. A precise model is presented in Section 3. We then study the *stable extension* of some fundamental problems in measurement and analytics.

In Section 4 we consider a class of optimization problems where the inputs are vectors and the outputs are a subset of the entries of a specified fixed size k. The class includes *top-k* set, where entries of the input vector correspond to elements, *Minimum Spanning Tree* (MST) where entries correspond to edges of a graph, and *Assignment*, where entries correspond to edges of the bipartite graph. The values of the entries are the respective weights of the edges/elements. The respective sets of outputs are all subsets of size k for *top-k*, all maximum size bipartite matchings for *assignment*, and all spanning trees for MST. The *best fit* solutions are the output with maximum sum of weights (for MST we look at negated weights, and thus the minimum sum of edge weights).

The stable extensions of problems in our general class have a fitness function that is additive, expressed as a linear combination of the weights of included entries. Fit values can be computed to all valid outputs. The changeout cost is also additive, and expressed as sum of the costs of new entries. With top-k, assignment, and MST, the fitness is the sum of weights of included entries (negated weights for MST) and the changeout cost is the number of entries we replace in the output.

We establish a relation between the stable extension of an optimization problem in this class and instances of the original optimization problem on a parametrically specified modified input. This relation allows us to apply an algorithm for the original optimization problem as a black box to obtain a stable solution. Moreover, we establish a mapping between a dynamic version of the stable extension (where we must

efficiently modify the output in response to incremental updates to the input) and the dynamic version of the original optimization problem. This mapping similarly allows off-the-shelf applications of existing dynamic algorithms to efficiently recompute a stable solution when the input is modified.

In Sections 5–7 we study sampling with stability, focusing on Probability Proportional to Size (PPS) weighted random samples [11]. PPS sampling is probably the oldest and most widely applied weighted sampling scheme. This is due to its simplicity, being based on independent sampling, and its optimality over independent sampling schemes.

The input is a vector w of nonnegative numbers which correspond to weights assigned to items in some population. Consider a sampling scheme specified by a vector p of probabilities. The sample is obtained by including each entry i independently with probability p_i. The expected size of such a sample is the L_1 norm of p. We measure the "fit" of p for an input vector w by the (negated) sum over entries of the variance $w_i^2(1-1/p_i)$ of the (inverse probability [13]) estimate we can obtain from the sample on its weight. This fit corresponds to the quality of estimates of subset sum statistics that are obtained from the sample.

For a specified expected sample size k, the PPS probabilities p are such that $p_i \propto w_i$ (proportional to the weight of the entry), except that probabilities are truncated to be at most 1. In is known that the PPS probabilities are "optimal" in that they maximize fit for a given (expected) sample size [11].

Our focus here is on applications where the weights w change over time. In the stable extension, we track the output vector p together with a sample taken according to p. We measure the changeout cost by the expectation of the set difference of samples between the two sample distributions. Under the appropriate procedure, we show that the cost can be kept to the minimal value, which is the L_1 difference between the initial and target probability vectors.

This model is suitable for applications with the classic context of survey sampling [19, 21]: There is a "processing cost" associated with "surveying" the sampled items, but we can maintain the vector p. The purpose of sampling in this context (think of households or road segments) is to replace the cost of surveying the full population by that of surveying only sampled items. Change adds another consideration: The best fit sample for given weights is the PPS sample, but with change, we aim to balance the fitness of a sample with the cost of modifying it (the number of new items we need to survey). This addresses many of the motivating settings described above.

In terms of efficiency, while stable PPS is naively formulated as a non-linear optimization problem, we first show how to decompose it into two convex programs with linear constraints, and then, using insights into the structure of the parametric solution, give an $O(n \log n)$ combinatorial algorithm which computes the full tradeoff (the Pareto frontier), expressed as a piecewise rational function. We also present a dynamic variant with polylogarithmic amortized time per modified entry in the input.

An empirical study of our methods (Section 8) demonstrates their advantages over alternative approaches. In Section 9, we discuss additional classic problems with natural stable extensions.

2 Related models

Our formulation of computing a stable solution at each step is appropriate when a continuously good fit is required, as in the motivating scenarios we outlined. There is a very large number of models and approaches to computing over changing data. These models target different settings and it is not possible to mention all of them. Two hugely popular models are *metrical task systems*, where the goal is to maximize *average fit per average change* [3] and *regret minimization* used in decision theory.

A natural first attempt at providing stability of output is to stabilize the input, which is commonly performed using some time-decayed average (via an exponential, polynomial, or sliding window decay function) over recent input states [6]. Decay models are successful in capturing the current state when

inputs can be modeled as noisy samples from a slowly changing underlying distribution. In this case, the time-decayed (moving) average better captures the "real" current state than the current input.

When using decay models directly to obtain stability of output, we can compute the best-fit output for the time-decayed average. This approach lacks, however, when similarity of the input does not guarantee similarity, or stability, of the output. Moreover, when the output function *is* insensitive to small perturbations in the input, we cannot fine tune the tradeoff of fit and stability by controlling the decay parameter: the stability of the output is not necessarily an increasing function of the decay parameter.

The stability-fit tradeoff can be cast as a multi-objective (bicriteria) optimization problem. The points on the tradeoff are the Pareto optimal solutions – the set of solutions so that any solution which strictly improves on one objective would perform strictly worse on the others. Our model, however, is tailored to maintain the tradeoff across a sequence of inputs. We provide novel efficient algorithms for computing the tradeoff and maintaining an appropriate solution "dynamically" when only a small number of entries change in the input.

A closely related notion is *reoptimization*, which is also about computing a close to optimal solution with a small change in the current solution. Reoptimization had mostly been looked at for specific problems, but recently was defined and studied as a general notion by Shachnai et al. [20]. An important difference between their work and the present paper is that we are aiming for the *instance-optimal* tradeoff whereas the work of [20] explores the existence of algorithms (or hardness of finding such algorithms) which guarantee a certain relation between the approximation factor of the new solution and its cost of change *relative to the cost of change to arrive at an optimal solution*. We believe our formulation is better suited for many applications. Also, the positive results presented in [20] for the MST and related problems (which we also consider here) are for computing a new *optimal* solution with minimum change. This is not satisfying when there is a slightly suboptimal solution which is much cheaper to change to. Our algorithms compute any solution on the Pareto front.

Proactive re-optimization was explored by Babu et al. [2] in the context of query execution plans: The goal is to come up with plans that are robust to small changes in the parameters and are adjusted when the parameters change "sufficiently." This line of work is somewhat related as we also seek to optimize the tradeoff between the cost of change and the fitness of the solution, but the model and context are different. In particular, we do not make assumptions on how the parameters may change but instead make a plan that allows us to respond to any possible change of parameters by optimizing the cost of changing the plan against the cost of being farther from the optimum. We also explore a different set of optimization problems.

3 Model

The fit of output $S \in \mathcal{S}$ to input $x \in \mathcal{X}$ is measured by a *fitness function* $\phi : \mathcal{S} \times \mathcal{X} \to R$. The *best-fit* $\textsc{opt}(x)$ is the output with maximum fitness for input x:

$$\textsc{opt}(x) = \arg\max_{S \in \mathcal{S}} \phi(S, x) \ .$$

Concrete examples of output domains $\mathcal{S}$ and corresponding best-fits are assignments (minimum cost assignment), routing tables (shortest path routing), subsets of size k (top-k set), spanning trees (MST), and sampling distributions (PPS sampling).

The *changeout cost* from output S_1 to S_2 is measured by a distance function on $\mathcal{S} \times \mathcal{S}$: $d(S_1, S_2) \geq 0$.

When presented with a new input x, and a current output S, we are interested in a Δ-*stable* optimum,

which is the best-fit output subject to a limit D on changeout cost, and its respective fitness:

$$\Delta\text{-OPT}(S,x,D) = \arg \max_{S' | d(S,S') \leq D} \phi(S',x)$$
$$\Delta\text{-}\phi(S,x,D) = \max_{S' | d(S,S') \leq D} \phi(S',x)$$

When fixing x and S, and varying the allotted change D, we obtain the optimal tradeoff between fit and changeout cost using the points $(D, \Delta\text{-}\phi(S,x,D))$.

The α-*stable* optimum $\alpha\text{-OPT}(S,x,a)$ is defined with respect to a parameter a, which specifies a linear relation between changeout cost and improvement in fitness:

$$\alpha\text{-OPT}(S,x,a) = \arg\max_{S'} \left(\phi(S',x) - a\, d(S,S')\right) \tag{1}$$
$$\alpha\text{-}\phi(S,x,a) = \phi(\alpha\text{-OPT}(S,x,a),x)$$

By sweeping the Lagrange multiplier a, we obtain the tradeoff using the points

$$\left(d(S, \alpha\text{-OPT}(S,x,a)), \alpha\text{-}\phi(S,x,a)\right) . \tag{2}$$

The fitness of the α-stable output decreases with a and that of a Δ-optimal one increases with D. For $a = 0$, we allow D to be arbitrarily large, and the α-stable optimum is the current best-fit $\text{OPT}(x)$. When a is sufficiently large, D is forced to be small and the α-stable optimum is closer to the previous S. Two desirable properties satisfied by the problems we consider are concavity and output-monotonicity of the tradeoff. These properties simplify the computation of the tradeoff and also mean that the choice of the best tradeoff point is predictable and tuneable.

Concave tradeoff: Concavity means that the function $\Delta\text{-}\phi(S,x,D)$ is concave in D, so that the marginal improvement in fitness for increase in allowed changeout cost is non-increasing. When the function is continuous and differentiable in D, the marginal improvement is the first derivative $\frac{\partial \Delta\text{-}\phi(S,x,D)}{\partial D}$ and concavity means $\frac{\partial^2 \Delta\text{-}\phi(S,x,D)}{\partial^2 D} < 0$. In this case, when the parameters satisfy $\Delta\text{-OPT}(S,x,D) = \alpha\text{-OPT}(S,x,a)$, then $a = \frac{\partial \Delta\text{-}\phi(S,x,D)}{\partial D}$.

Output monotonicity: When the output can be expressed as a vector, output monotonicity means that each entry (inclusion, sampling probability, etc.) is monotone when sweeping a and looking at the α-stable optimum (or equivalently, when sweeping D and looking at the Δ-stable optimum). In particular, when output entries are binary, then S can be viewed as a set. For each $i \in S$, there is at most one threshold value τ so that $i \notin \alpha\text{-OPT}(S,x,a) \iff a < \tau$ and for each $i \notin S$ there is at most one threshold value of τ so that $i \in \alpha\text{-OPT}(S,x,a) \iff a < \tau$.

Problem formulations

The applications we target have a sequence of inputs. A natural aim is to seek outputs that are α-stable in each step with respect to the same parameter a — meaning that we "price" the changeout cost the same across steps. Alternatively or additionally, we may need to limit the changeout at each step.

Stable solutions and tradeoff: The underlying basic algorithmic problem is to compute, for a given input x and current output S, the respective Δ-stable and the α-stable optimum solutions.

Incremental updates: When in each step the input is provided as a small update to the previous one, such as a rewrite, increment, or decrement of a single entry of the vector, we would like to efficiently compute a new α-stable optimum instead of applying the batch algorithm to compute it from scratch.

4 Additive fitness and changeout cost

We begin by describing a general approach for a broad class of optimization problems. We say an optimization problem with input domain of the vectors $\mathcal{X} \subset \mathbb{R}^n$ and outputs $\mathcal{S}$ that are subsets $S \subset [n]$ has *additive fitness* if its objective is to maximize the fitness function $\phi(S,x) = \sum_{i \in S} x_i$:

$$\text{OPT}(x) = \arg\max_{S \in \mathcal{S}} \sum_{i \in S} x_i \ .$$

The changeout cost between two outputs is *additive* if there are constants c_i such that $d(S,S') = \sum_{i \in S' \setminus S} c_i$. The constant c_i can be interpreted as the cost of "bringing in" i to the output. When the optimization problem has additive fitness and changeout cost, the α-stable optimum w.r.t. current output S can be posed as a best-fit instance on a modified input vector with an appropriate quantity subtracted from entries in $i \notin S$:

Theorem 4.1 *Let* OPT *be an optimization problem with additive fitness and changeout. The α-stable optimum satisfies α-OPT$(S,x,a) =$ OPT(y) , where $y_i = x_i - I_{i \notin S} c_i a$ and I is the indicator function.*

Proof From (1),

$$\alpha\text{-OPT}(S,x,a) = \arg\max_{S' \in \mathcal{S}} \Big(-a \sum_{i \in S' \setminus S} c_i + \sum_{i \in S'} x_i \Big) \qquad = \arg\max_{S' \in \mathcal{S}} \sum_{i \in S'} y_i = \text{OPT}(y) \ .$$

■

We now consider maintaining an α-stable optimum efficiently under incremental updates. We show that this can be done using an off-the-shelf dynamic OPT algorithm:

Theorem 4.2 *If fitness and changeout are additive, then when the input is modified, the α-stable solution can be maintained by applying a dynamic* OPT *algorithm with a corresponding number of modified entries.*

Proof For an input x and current output $\mathbf{S}$, we consider the vector y such that $y_i = x_i - I_{i \notin S} a c_i$. Recall that the α-stable optimum is the best-fit with respect to input y. It therefore suffices to translate the modifications in x to the modifications in y. When an entry of x is modified, we apply a corresponding modification to the same entry in y. When the algorithm modifies the output, we increase by ac_i the value of new entries inserted to S and decrease by ac_i the value of entries that are deleted from S. The total number of updated entries in y is the sum of the updates to the input and the size of the set difference between the α-stable outputs. ■

Fixed-size output and uniform costs.

We now study problems that possess additional structure, namely *fixed-size output*, $\forall S \in \mathcal{S}, |S| = k$ and *uniform costs* $c_i = 1$. In particular, this formulation applies to matroids. We can equivalently treat changeout costs as those of excluding entries currently in the output. Examples of such problems include Top-k, MST, and assignment, which we study individually below.

Specializing Theorems 4.1 and 4.2, the α-stable optimum can be computed by applying OPT to a vector y obtained by taking x and adding the value a to entries that are in S: α-OPT$(S,x,a) =$ OPT(y) , where $y_i = x_i + I_{i \in S} a$:

$$\begin{aligned} \alpha\text{-OPT}(S,x,a) &= \arg\max_{S' \in \mathcal{S}} \Big(-a|S' \setminus S| + \sum_{i \in S'} x_i \Big) = \arg\max_{S' \in \mathcal{S}} \Big(a|S' \cap S| + \sum_{i \in S'} x_i \Big) \\ &= \arg\max_{S' \in \mathcal{S}} \sum_{i \in S'} y_i = \text{OPT}(y) \ . \end{aligned}$$

We can use this reduction when the input domain includes $y \geq x$ when it includes x.
We now consider the structure of the objective of α-OPT as a function of a:

Lemma 4.1 *The function* $\max_{S' \in \mathcal{S}} \left(\sum_{i \in S'} x_i + a|S \cap S'| \right)$ *is the maximum of at most* $k+1$ *linear functions. Thus, it is convex and piecewise linear with at most k breakpoints.*

Proof The objective is a (restricted) *parametric* version of the original optimization problem [17, 14, 15, 8, 10]. In a solution with respect to a parameter a, the weight of each entry is replaced by the linear function $x_i + a$ when $i \in S$ and is fixed to x_i otherwise. The value of a particular output S' with respect to S and a is $\sum_{i \in S'} x_i + a|S \cap S'|$. The optimal solution as a function of a is the maximum over all $S' \in \mathcal{S}$. These are linear functions with at most $k+1$ different slopes, corresponding to the intersection size $|S \cap S'|$. For each intersection size $\{0, \ldots, k\}$, there is one dominant linear function. The maximum of linear functions with slopes $\{0, \ldots, k\}$ is such that each function dominates in at most one piece and the slopes are increasing. ■

If we explicitly compute a representation of the objective function, we can obtain a Δ-stable optimum, by identifying a corresponding value of the parameter a: The breakpoints of the function correspond to decreasing changeout (increasing slopes) from changeout k (slope 0) to changeout 0 (slope k). The marginal gain in fitness from each unit increase in changeout is the respective value of a at the breakpoint. Hence:

Corollary 4.2 *When the problem has additive fitness, additive and uniform changeout, and fixed-size output, the tradeoff is concave.*

There are at most $k+1$ different α-stable solutions (eliminating duplications) and thus *all* can be specified in $O(k^2)$ space. When the tradeoff is output-monotone, then this description requires only $O(k)$ space. This is because there are at most $2k$ entries involved in total, and each has a single exit or entry point as we sweep the parameter. We next explore three basic problems: Top-k, MST, and assignment. We will see that output-monotonicity holds for Top-k and MST but not for assignment.

4.1 Stable top-k

The top-k problem has input domain $\mathcal{X} \subset R^n$ so that x_i is the 'weight' of i. The permitted outputs $\mathcal{S}$ are all subsets $S \subset [n]$ of size k, and the goal is to maximize ϕ, the sum of weights of the selected subset. The problem satisfies the conditions of Theorem 4.1 and therefore the α-stable top-k with respect to input x and current set S is α-top-$k(S,x,a) \equiv$ top-$k(y)$, where $y_i = x_i + I_{i \in S} a$.

Lemma 4.3 *The stability-fit tradeoff for stable top-k is output-monotone and can be found in* $O(n + k \log k)$ *time.*

Proof Consider the sequence of swaps made by α-top-k(S,x,a) while sweeping a. For sufficiently small $a \geq 0$, α-top-k$(S,x,a) = S$. For sufficiently large a, α-top-k$(S,x,a) =$ top-k(x). The total number of swaps is the set difference $d(S,$top-k$(x))$. The swaps are between the lightest $i \in S$ and the heaviest $i \notin S$.

To compute all tradeoff points, we sort the entries in S by increasing x_i to obtain the order $(i_1, \ldots, i_k)$, and sort the top-k entries $i \notin S$ in decreasing order $(j_1, \ldots, j_k)$. This obtains the stated time bounds. Let $H = \arg\max_h x_{i_h} < x_{j_h}$. For $h \leq H$, the most beneficial swap of at most h items, swaps-in items $j_1, \ldots, j_h$ and swaps-out items $i_1, \ldots, i_h$. The α-stable swap uses the maximum $h \leq H$ such that $x_{j_h} - x_{i_h} \geq a$. The tradeoff is clearly output-monotone, as each entry is swapped in or out at one particular value. ■

Incremental algorithm: The problem also satisfies the conditions of Theorem 4.2 and therefore to obtain an incremental algorithm, we outline a simple dynamic algorithm for maintaining a top-k set. We use two heap data structures. The first heap is a min-heap of the k cached entries (we call it the "in-heap"). The second heap is a max-heap of the remaining $n-k$ entries ("out-heap"). Weight updates are performed as updates to the weights of the corresponding elements in the heap. Swaps cause the deletion of the minimum item in the in-heap and the maximum item in the out-heap, and their reinsertion in the other heap. Swaps are performed when $x_i - x_j \geq \alpha$, where i is the maximum item in the out-heap and j is the minimum item in the in-heap. In this algorithm each update has an overhead of $O(\log n)$ — note that an update of a single value can result in at most a single swap.

Example: Consider an input vector $x = \langle 1,4,7,5 \rangle$ and $k = 2$. The top-2 set contains the entries $\{3,4\}$. Suppose now that the current output is the first two entries $S = \{1,2\}$. The best swap is $1 \leftrightarrow 3$ (swapping the smallest cached value with the largest uncached value) and the second most beneficial swap is $2 \leftrightarrow 4$ (swapping 2nd smallest cached with 2nd largest uncached). The optimal tradeoff has 3 points: performing no swap, only the first swap, or both swaps. Looking at α-stable solution, the first swap has utility gain $7-1=6$ and the second swap has utility gain $5-4=1$. Thus the first swap is performed when $a \leq 6$ and both when $a \leq 1$. ■

A simple but potentially useful observation is that our results extend to fitness function of the more general form $\phi(S,x) = \sum_{i \in S} \psi_i(x_i)$, where ψ_i are some functions we apply the the input. The extension is obtained by simply treating $\psi(x)$ as the input. One usage example is to measure fitness by L_p^p: $\sum_{i \in S} |x|^p$. We next demonstrate how the choice of p may affect the result when we process a sequence of inputs using the same a value: Augmenting our example, consider the α-stable outputs for two inputs: $x = \langle 1,4,7,5 \rangle$ and another vector $z = \langle 2,3,8,4 \rangle$. In both cases, the current output is the first two entries $\{1,2\}$. The vector z has the same tradeoff curve as x when using the L_1 norm. If we want to use the L_2 norm for fitness, we use $\psi(x) = x^2$. In this case, the two most valuable swaps have utility gains of $7^2 - 1^2 = 48$ and $5^2 - 4^2 = 9$ on x and $8^2 - 2^2 = 60$ and $4^2 - 3^2 = 7$ on z. Therefore, on the same a, and looking at both vectors together, it is possible that only the first swap is performed on the x ($60 \geq a > 48$), only the first swaps are performed on both ($48 \geq a > 9$), the two swaps on x and only the first swap on z ($9 \geq a > 7$), or all four swaps ($a \leq 7$).

4.2 Stable MST

In the stable MST problem, the inputs are a set of edge weights and the outputs are spanning trees. The fit of a tree is its negated weight. The change cost between two trees is the set difference of included edges. The problem clearly satisfies the conditions of Theorem 4.1, hence, an α-stable MST can be obtained by adding a to the edge weights of edges in the current output tree and computing an MST. Applying Theorem 4.2, dynamic α-stable MST can be handled using off-the-shelf dynamic MST algorithms [12].

Lemma 4.4 *The tradeoff for stable MST is output monotone.*

Proof Consider starting from an optimal MST and slowly increasing a, obtaining a sequence of α-stable MSTs. An edge is removed when it is the heaviest edge in some cycle. This can only happen if this edge is not a member of the output S (because weights of all members of S decrease by the same amount and all weights of other edges stay the same). Once this happen, weights of other edges in the cycle either keep decreasing or stay the same, so the edge remains the heaviest in that cycle and out of the MST. Similarly, an edge is inserted into the α-stable MST when it has smallest weight in some cut. Such an edge must be a member of S and its weight remains the smallest in its cut as a increases. Thus the tradeoff is output

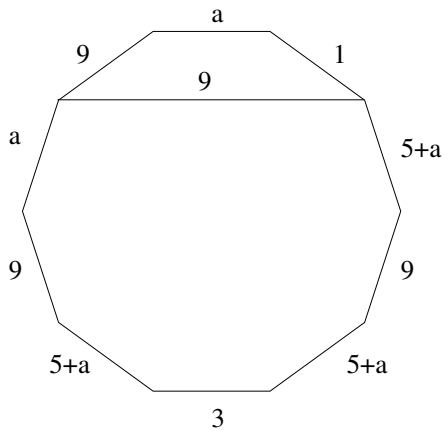

Figure 1: Example demonstrating non-monotonicity of assignment

monotone. ■

The tradeoff can be computed using a parametric MST algorithm [1], but the computation can be made more efficient for this special case.

4.3 Stable assignment

In the stable assignment problem, the inputs are weights of edges in a complete bipartite graph and the outputs are maximum matchings. The problem satisfies the conditions of Theorem 4.1, and hence, an α-stable solution can be computed by adding a to all the weights of edges present in the current matching and then finding a maximum-weight assignment (maximum weight bipartite matching). We also know there are at most $n+1$ distinct stable optimal assignments (assuming a single optimal assignment is used). From Theorem 4.2, dynamic α-stable assignment reduces to dynamic maximum bipartite matching [18]. However, unlike the previous problems, the tradeoff is not output monotone. Specifically, Figure 1 shows a simple example of an assignment problem which is not output-monotone in its inclusion of edges. It shows a 10 node bipartite weighted graphs (where missing edges are assumed to have weight zero). A matching from the previous timestep causes five edges to have their (current) scores increased by a. We can find the tradeoff by considering solutions which include different numbers of the previous edges, as per Lemma 4.1. The solution that picks none of the previous edges has weight 31. The solution that picks one of the previous edges picks the top edge, and has weight $30+a$. The solution that picks three previous edges has weight $24+3a$, and picks the three edges with weight $5+a$. Lastly, the solution that picks all five previous edges has weight $15+5a$. The cases of two and four previous edges are dominated by these solutions. The α-OPT objective is the maximum of the four linear equations: $y=31$, $y=30+1a$, $y=24+3a$, $y=15+5a$, and each dominates the other three for a different range of a. Further, note that the top a edge is included for $a \in (1,3)$ and not for $a \in (0,1)$ and $(3,4.5)$. Therefore, the tradeoff is not output monotone.

5 Stable PPS sample

In this section we consider the stable extension of PPS sampling. The input domain $\mathcal{X}$ consists of real non-negative vectors $w \in R_+^n$ and the output domain $\mathcal{S}$ contains probability distributions specified by all vectors $p \in [0,1]^n$ such that $\sum_{i=1}^n p_i = k$. Together with the distribution, we maintain a random sample where entry i is sampled with probability p_i and different entries are sampled independently.

Fitness function: When the ith entry is sampled with probability p_i, the unbiased non-negative estimator

with minimum variance on the weight w_i of the entry is the Horvitz-Thompson estimator [13], i.e. $\frac{w_i}{p_i}$ when the entry is sampled and 0 otherwise. The variance is $w_i^2(1/p_i - 1)$.

We measure the quality of a sample distribution p for input w by the negated sum of variances $\sum_i w_i^2(1/p_i - 1)$. We can omit the additive term $-\sum_i w_i^2$ (which does not depend on p) and obtain the fitness function

$$\phi(p,w) = -\sum_i \frac{w_i^2}{p_i} \ .$$

The best-fit for a sample of expected size k is the distribution with minimum sum of variances. This distribution is the PPS sampling probabilities, $p = \text{pps}(k,w)$, obtained by solving for τ the following equation, and using $p_i = \min\{1, w_i/\tau\}$:

$$\sum_i \min\{1, w_i/\tau\} = k \ . \tag{3}$$

Changeout cost: The random sample is a set of entries and our distance function measures the *expected* set difference between the samples. The distance therefore depends on the particular *subsampling procedure* we employ to move from a sample S drawn from probability distribution p to a new one drawn from q. Clearly, the distance must be at least $\|p-q\|_1 = \sum_i |p_i - q_i|$. We can use a SUBSAMPLE procedure (pseudo-code in Algorithm 1), which has distance *exactly* $d(p,q) = \|p-q\|_1$ and is therefore minimal. We note that we can implement SUBSAMPLE using VarOpt (see details in [5]), which can ensure that the number of inclusions and exclusions are each between the floor and ceiling of their expectation. Alternatively, we can associate a permanent random number (PRN) [4] $u_i \in U[0,1]$ so that $i \in S \iff u_i \leq p_i$.

Example 5.1 *Consider sampling a set of expected size* 2 *from* 6 *entries. Suppose we have a current sample S, drawn according to a uniform distribution p, where $p_i = 1/3$ for $i \in [6]$. This distribution has the best fit when all entries in the input have the same weight.*

Suppose that the new input weights are $w = (2,4,1,5,6,0)$. The PPS sampling probabilities for a sample of size 2*, obtained with threshold $\tau = 9$, are*

$$pps(2,w) = \left(\frac{2}{9}, \frac{4}{9}, \frac{1}{9}, \frac{5}{9}, \frac{2}{3}, 0\right) . \tag{4}$$

The expected changeout cost of modifying the sample from one drawn according to the uniform p to one drawn according to $pps(2,w)$ is $\|p - pps(2,w)\|_1 = \frac{4}{3}$.

The actual changeout to the sampling distribution $pps(2,w)$ depends on our initial sample S and the randomization when subsampling. Suppose $S = \{3,4\}$, that is, contains the third and fourth entries. Entries $\{1,6\}$ are not in S and have reduced sampling probabilities and therefore will not be included in the new sample. Entry 4 *is in S and has increased inclusion probability and therefore will be included in the new sample. Entries $\{2,5\}$ which are not in S but for which inclusion probabilities increased, need to be sampled for inclusion with respective probabilities $\{1/6, 1/2\}$. Entry* 3 *which is in S but has a reduced inclusion probability is sampled out with probability $2/3$. Therefore, the new sample will include entry* 4 *and will not include entries $\{1,6\}$ but may include any subset of entries $\{2,3,5\}$.*

The stability-fit tradeoff includes a sequence of distributions that are between p (higher variance, no changeout cost) and $pps(2,w)$ (high changeout cost, minimum variance). In the sequel, we will see how the tradeoff is computed for this example. ∎

Algorithm 1 SUBSAMPLE procedure

```
procedure SUBSAMPLE(S, p, q)
    for i ∈ [n] do
        if i ∉ S and q_i > p_i then
            if RAND() < (q_i − p_i)/(1 − p_i) then
                S ← S ∪ {i}                ▷ i is sampled-in with probability (q_i − p_i)/(1 − p_i)
        if i ∈ S and q_i < p_i then
            if RAND() > q_i/p_i then
                S ← S \ {i}                ▷ i is sampled-out with probability 1 − q_i/p_i
return S
```

6 Computing the stability/fit tradeoff for stable PPS

In this section we establish the following:

Theorem 6.1 *An α-stable PPS distribution, Δ-stable PPS distribution, and a description of the tradeoff mapping parameter values to fitness and vice versa, can be computed in $O(n \log n)$ time. Moreover, the tradeoff is concave and output-monotone.*

Let p be the initial output and w the current weights. The Δ-stable distribution q is the solution of the following (nonlinear) optimization problem:

$$\begin{aligned} \text{minimize} \sum_i \frac{w_i^2}{q_i} \\ \forall i,\ 1 \geq q_i \geq 0 \\ \sum_i q_i = k \\ \sum_i |q_i - p_i| \leq D \end{aligned} \tag{5}$$

We are interested in efficiently solving the problem for a particular value of the changeout D and also in computing a compact representation of all solutions.

We reduce (5) to two simpler convex optimization problems with linear constraints. BEST-INCREASE computes, for any value x, the most beneficial total increase of x to current sampling probabilities, i.e., that would bring the biggest reduction in the sum of variances. Similarly, BEST-DECREASE computes the least detrimental total decrease of x, i.e., the decrease that results in the minimum increase of the sum of variances. We show that descriptions of both functions can be found in time $O(n \log n)$. Last, we show how to fully describe the Δ-stable PPS distributions and α-stable distributions in time $O(n \log n)$ by using these functions.

6.1 The most beneficial increase (BEST-INCREASE).

The most beneficial increase for $x \in (0, \sum_i (1 - p_i)]$ is δ which solves the following convex program.

$$\begin{aligned} \text{minimize} \sum_i \frac{w_i^2}{p_i + \delta_i} \\ \forall i,\ 1 - p_i \geq \delta_i \geq 0 \\ \sum_i \delta_i = x \end{aligned} \tag{6}$$

Lemma 6.1 *For any increase $x \in (0, \sum_i (1-p_i)]$ there is a threshold value $y \equiv \underline{\tau}(x)$ such that the solution of (6) is $\delta_i = \min\{1, \max\{p_i, w_i/y\}\} - p_i$. The function $\underline{\tau}(x)$ and its inverse $\Delta^+(y)$ are continuous and decreasing and piecewise with at most $2n$ breakpoints, where each piece is a hyperbola. Moreover, a representation of both functions can be computed in $O(n \log n)$ time.*

Proof From convexity, the solution is unique and the Karush-Kuhn-Tucker conditions [16] specify its form: The partial derivatives $\frac{\partial \frac{w_i^2}{p_i+\delta_i}}{\partial \delta_i} = -\frac{w_i^2}{(p_i+\delta_i)^2}$ are equal for all i such that $1 - p_i > \delta_i > 0$. We denote this common value of $\frac{w_i}{p_i+\delta_i}$ by $\underline{\tau}(x)$. For i where $\delta_i = 0$, we have $\frac{w_i}{p_i} \leq \underline{\tau}(x)$ and when $0 < \delta_i = 1 - p_i$, we have $w_i \geq \underline{\tau}(x)$.

Using this, we can now consider the solution as a function of x. We first sort all items with $0 < p_i < 1$ in decreasing order of w_i/p_i and place items with $p_i = 0$ and $w_i > 0$ at the head of the sorted list. The list is processed in order: we first increase δ_1 until $\frac{w_1}{p_1+\delta_1} = \frac{w_2}{p_2}$. We then increase both δ_1 and δ_2 so that the ratios are all equal until $\frac{w_1}{p_1+\delta_1} = \frac{w_2}{p_2+\delta_2} = \frac{w_3}{p_3}$ and so on (but we stop increasing a probability once it is equal to 1). At any point, the ratios $\frac{w_i}{p_i+\delta_i}$ on the prefix $1, \ldots, h$ we have processed are all equal to some value y except for items where p_i reached 1. The threshold $y \equiv \underline{\tau}(x)$ can be interpreted as the rate of decrease of the sum of variances per unit increase in expected sample size after implementing optimally a total increase of x. It is a decreasing function of x and satisfies $y \in (w_{h+1}/p_{h+1}, w_h/p_h]$, where h is the last processed entry on our sorted list. The new sampling probabilities $q = p + \delta$ satisfy $q_i(y) = \min\{1, \max\{p_i, w_i/y\}\}$. Observe that the probabilities $q(y)$ are PPS of size $\sum_{i=1}^h p_i + x$ of the prefix of the list

$$(q_1, \ldots, q_h) \leftarrow \text{pps}(\sum_{i=1}^h p_i + x, (w_1, \ldots, w_h)) \ ,$$

and y is the threshold value of these PPS probabilities.

Consider the (at most $2n$) points defined by the ratios w_i/p_i (when $p_i < 1$ and $w_i > 0$) and the weights w_i. Consider the relation between x and $y = \underline{\tau}(x)$. The function is a hyperbola $x = a + b/y$ for some constants a, b between consecutive breakpoints.

The inverse function of $\underline{\tau}(x)$, which we call $\Delta^+(y)$, maps threshold values to probability increases, and satisfies

$$x = \Delta^+(y) = \sum_i q_i(y) - p_i = \sum_i \min\{1, \max\{p_i, w_i/y\}\} - p_i \ .$$

It is continuous and decreasing with range $[\min_{i|p_i<1} w_i, R)$, where $R = \infty$ if $\exists i$ with $w_i > 0$ and $p_i = 0$ and $R = \max_{i|p_i<1} \frac{w_i}{p_i}$ otherwise. It is piecewise where each piece is a hyperbola.

Algorithm 2 computes a representation of the functions $\Delta^+(y)$ and $\underline{\tau}(x)$ each as a sorted list of pieces of the form $(I, f(y))$ where I are intervals that partition the domain and f is the function for values on that interval. For $\Delta^+(y)$, the pieces have the form $f(y) = a + b/y$. The inverse function $\underline{\tau}(x)$ is simultaneously produced by the algorithm by inverting each piece $(I, a+b/y)$ to obtain a respective interval where end point v of I is mapped to end point $f(v) = a + b/v$ (left end points are mapped to right end points and vice versa). The inverse function on the corresponding interval is $\underline{\tau}(x) = f^{-1}(x) = \frac{b}{x-a}$.

For each i with $w_i > 0$ and $p_i < 1$, Algorithm 2 creates an *entry point* with value $\frac{w_i}{p_i}$ and for each item with positive w_i, creates an exit point with value w_i. These (at most $2n$) points are the breakpoints of $\Delta^+(y)$. The breakpoints are processed in order to compute the functions $\Delta^+(y)$ and $\underline{\tau}$. The computation is performed in linear time after sorting the breakpoints. ∎

The following example shows how BEST-INCREASE is computed for the instance in Example 5.1:

Algorithm 2 The function $\Delta^+(y)$ and its inverse $\underline{\tau}(x)$ for w and p

procedure BEST-INCREASE(w, p)
  $W \leftarrow 0$ ▷ weight of active items
  $D \leftarrow 0$ ▷ Contribution to difference of items that exited and negated sum of initial probabilities of active items
  $L \leftarrow \emptyset$ ▷ Initialize L as an empty max-heap
  for $i \in [n]$ **do**
    if $(w_i > 0)$ **then**
      if $(p_i < 1)$ **then**
        $(A.v, A.t, A.i) \leftarrow (\frac{w_i}{p_i}, \text{"entry"}, i)$ ▷ value, breakpoint type, item
        $L \leftarrow L \cup A$ ▷ add entry record to L
      else if $p_i = 0$ **then**
        $W \leftarrow W + w_i$
      $(A.v, A.t, A.i) \leftarrow (w_i, \text{"exit"}, i)$ ▷ value, breakpoint type, item
      $L \leftarrow L \cup A$ ▷ add exit record to L
  $A \leftarrow \text{GETMAX}(L)$ ▷ Pull Record with largest $A.v$ in L
  if $W > 0$ **then**
    $\Delta^+ \leftarrow \{([A.v, \infty), \frac{W}{y})\}$ ▷ "Rightmost" piece of Function $\Delta^+(y)$
    $\underline{\tau} \leftarrow \{((0, \frac{W}{A.v}], \frac{W}{x})\}$ ▷ "Leftmost" piece of Function $\underline{\tau}(x)$
  else
    $\Delta^+ \leftarrow \emptyset$, $\underline{\tau} \leftarrow \emptyset$
  $r \leftarrow A.v$ ▷ definition boundary so far of Δ^+
  $\ell \leftarrow \frac{W}{A.v}$ ▷ definition boundary so far of $\underline{\tau}$
  while $L \neq \emptyset$ **do**
    $A \leftarrow \text{GETMAX}(L)$
    $i \leftarrow A.i$
    if $A.t =$ "entry" **then**
      $W \leftarrow W + w_i$
      $D \leftarrow D - p_i$
    else ▷ A is an exit point
      $W \leftarrow W - w_i$
      $D \leftarrow D + 1$
    $\Delta^+ \leftarrow \Delta^+ \cup ([A.v, r], D + \frac{W}{y})$ ▷ Function Δ^+ from $A.v$ to next breakpoint
    $r \leftarrow A.v$
    $\underline{\tau} \leftarrow \underline{\tau} \cup \{((\ell, D + \frac{W}{A.v}], \frac{W}{x-D})\}$ ▷ Function $\underline{\tau}$ from previous to current breakpoint
    $\ell \leftarrow D + \frac{W}{A.v}$
return Δ^+, $\underline{\tau}$

Example 6.2 *Returning to Example 5.1, we consider the best increase for w and the uniform p. Since probabilities are uniform, Algorithm 2 processes entries in order of decreasing weight. The first entry on our list is entry* 5*, which has weight of* 6*. The best increase is applied only to this entry until the ratio matches the ratio of the second entry on the list, which happens when* $6/(1/3+\delta) = 5/(1/3)$*, that is, when* $\delta = 1/15$*. So the first piece of our function is for the interval* $x \in (0, 1/15]$*. The corresponding distribution has entry* 5 *with probability* $1/3+x$ *(and all other entries with the original* $1/3$*). The algorithm has active weight* $W = 6$ *and thus produces the threshold function* $\underline{\tau} = 6/(1/3+x)$*. We can verify that the inclusion probability for changeout x is* $q_5 = 6/\underline{\tau}(x) = 1/3+x$*. The second piece of the function associates increases for both entry* 5*, and the second entry on our list, which is entry* 4*. They are increased until the ratios are equal to the third entry on our list, which is entry* 2*. This happens when the increase to entry* 5 *is* $1/6$*, according to* $6/(1/3+\delta) = 4/(1/3)$ *and when the increase to entry* 4 *is* $1/12$ *according to* $5/(1/3+\delta) = 4/(1/3)$*. The total increase from entries* $\{4,5\}$ *is* $1/6+1/12 = 1/4$ *and we obtain that the second piece of the function is for* $x \in (1/15, 1/4)$*, and both entries* $\{4,5\}$ *are increased. Algorithm 2 computes the active weight* $W = 6+5$ *and the threshold* $\underline{\tau}(x) = 11/(x+2/3)$*. Thus for an increase of x, entry* 5 *is sampled with probability* $6/\underline{\tau}(x) = (6/11)(x+2/3)$ *and entry* 4 *with probability* $5/\underline{\tau}(x) = (6/11)(x+2/3)$*. The third piece would have* 3 *active entries* $\{2,4,5\}$ *(the three largest entries.). The active weight is* $W = 15$ *and the threshold is accordingly* $\underline{\tau}(x) = 15/(x+1)$*. The interval for this third piece is* $(1/4, 2/3]$*.* ∎

6.2 The least detrimental decrease (BEST-DECREASE)

We now discuss the complementary BEST-DECREASE problem: Computing the best decrease to the probabilities. For $x \in [0, \sum_i p_i]$, it is the solution of the optimization problem

$$\begin{aligned} \text{minimize} \sum_{i|w_i>0} \frac{w_i^2}{p_i - \delta_i} \\ \forall i,\ p_i \geq \delta_i \geq 0 \\ \sum_i \delta_i = x \end{aligned} \tag{7}$$

We first observe that any total decrease $x \leq D_0$, where $D_0 = \sum_{i|w_i=0} p_i$ can be arbitrarily applied to probabilities of items with zero weight and does not increase the sum of variances.

Lemma 6.3 *For any decrease* $x \in (D_0, \sum_i p_i]$*, there is a threshold* $y \equiv \overline{\tau}(x)$ *such that the solution of* (7) *has the form* $\delta_i = p_i - \min\{p_i, w_i/y\}$*. The function* $\overline{\tau}(x)$ *and its inverse* $\Delta^-(y)$ *are increasing, continuous, and piecewise with at most n breakpoints where each piece is a hyperbola. Moreover, a representation of both functions can be computed in* $O(n \log n)$ *time.*

Proof For any decrease $x \in (D_0, \sum_i p_i]$, the solution is unique and specified by the Karush-Kuhn-Tucker conditions [16]. For all i where $w_i = 0$ we must have $\delta_i = p_i$. For all i such that $\delta_i \in (0, p_i)$, the partial derivatives

$$\frac{\partial \frac{w_i^2}{p_i - \delta_i}}{\partial \delta_i} = -\frac{w_i^2}{(p_i - \delta_i)^2}$$

are equal and we denote the common value of $\frac{w_i}{p_i - \delta_i}$ by $\overline{\tau}(x)$. When $\delta_i = 0$, we must have $\frac{w_i}{p_i} \geq \overline{\tau}(x)$.

The solution thus has the form

$$\delta_i = p_i - \min\{p_i, w_i/y\} = \max\{0, p_i - w_i/y\}\ .$$

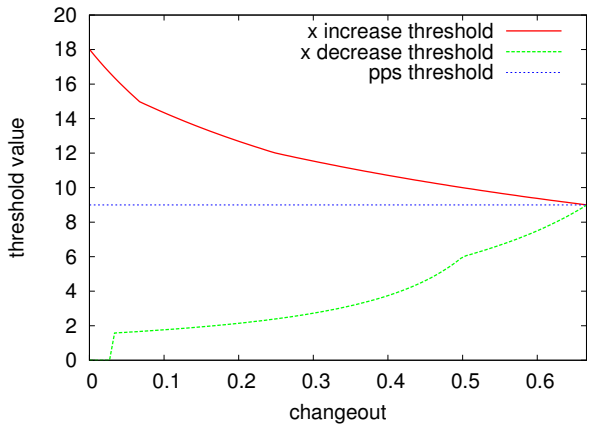

Figure 2: The thresholds $\underline{\tau}$ and $\overline{\tau}$ as a function of changeout x for Example 5.1. PPS threshold with maximum changeout $(4/3)$ is provided as a reference.

We obtain the relation

$$x = \sum_i \delta_i = \sum_{i|w_i/y<p_i} (p_i - w_i/y) \ ,$$

which defines the functions $y \equiv \overline{\tau}(x)$ and its inverse $x \equiv \Delta^-(y)$. Both $\overline{\tau}(x)$ and its inverse $\Delta^-(y)$ are continuous and decreasing. The range of $\overline{\tau}(x)$ (domain of $\Delta^-(y)$) is $[\min_{i|p_i>0} w_i/p_i, \infty)$. Both are piecewise with at most n breakpoints which correspond to the values of the ratios w_i/p_i. Algorithm 3 computes a representation of these two functions. It first orders the items as in Algorithm 2 and the proof of Lemma 6.1, according to the ratios w_i/p_i and processes this list in reverse order, generating the pieces of the two functions. Each function is represented as a list of consecutive pieces of the form $(I, f(y))$, where I is an interval. For convenience, we define $\overline{\tau}(x) = 0$ for $x \leq D_0$. For $\Delta^-(y)$, $f(y)$ has the form $f(y) = a - b/y$. The algorithm inverts $f(y)$ to obtain the corresponding piece of the function $\overline{\tau}(x)$: an end point v of I is mapped to end point $a - b/v$ (left end points are mapped to right end points and vice versa) and the function is $b/(a-x)$.

The probabilities $q \leftarrow p - \delta$ computed for a given x are PPS with threshold is y:

$$(q_t, \ldots, q_n) \leftarrow \text{pps}(\sum_{i=t}^{n} p_i - x, (w_t, \ldots, w_n)) \ .$$

■

Example 6.4 *We compute the* Best-decrease *function for Example 5.1 using Algorithm 3. We obtain that for $x \in (0, 1/3]$ we only decrease the probability for entry* 6*, which has weight of* 0*. We then decrease only the second smallest entry, which is entry* 3 *with weight* 1*. The interval of the second piece is accordingly $x \in (1/3, 1/2]$, we have active weight* 1*, $\overline{\tau}(x) = 1/(2/3 - x)$, and $q_6 = 0$ and $q_2 = 1/\overline{\tau}(x) = 2/3 - x$. The third piece $x \in (1/2, 2/3]$ involves entries $\{1, 3\}$. The active weight is $W = 3$ and $\overline{\tau}(x) = \frac{3}{1-x}$. Accordingly, the probabilities are $q_6 = 0$, $q_2 = 2/\overline{\tau}(x)$, and $q_1 = 1/\overline{\tau}(x)$.* ■

Algorithm 3 The function $\Delta^-(y) : (0,\infty)$ and its (extended) inverse $\overline{\tau} : (0, \sum_i p_i)$ for w and p

procedure BEST-DECREASE(w, p)
  $W \leftarrow 0$ ▷ Initial active weight W
  $D \leftarrow 0$
  $\Delta^- \leftarrow \emptyset$, $\overline{\tau} \leftarrow \emptyset$
  for $i \in [n]$ **do**
    if $(w_i > 0)$ **then**
      $(A.v, A.i) \leftarrow (\frac{w_i}{p_i}, i)$ ▷ value, item
      $L \leftarrow L \cup A$ ▷ add entry record to L
    else, $D \leftarrow D + p_i$ ▷ $w_i = 0$
  $\overline{\tau} \leftarrow ((0, D], 0)$ ▷ Threshold function is 0 for $y \leq D$
  $\ell \leftarrow D$ ▷ Left boundary of current definition of $\overline{\tau}$
  $r \leftarrow \infty$ ▷ Right boundary of current definition of Δ^{-1}
  while $L \neq \emptyset$ **do**
    $A \leftarrow$ GETMIN(L) ▷ Pull out record with minimum $A.v$
    $i \leftarrow A.i$
    $W \leftarrow W + w_i$
    $D \leftarrow D + p_i$
    $\Delta^- \leftarrow \Delta^- \cup ([A.v, r), D - \frac{W}{y})$
    $r \leftarrow D - \frac{W}{A.v}$
    $\overline{\tau} \leftarrow \overline{\tau} \cup ((\ell, D - \frac{W}{A.v}], \frac{W}{D-x})$
    $\ell \leftarrow D - \frac{W}{A.v}$
return Δ^-, $\overline{\tau}$

6.3 Computing a Δ-stable PPS distribution.

We consider computing Δ-OPT(w, p, D), which is the distribution with maximum fitness (minimum variance) $\phi(q, w)$, amongst q that are within distance D from p ($\|p - q\|_1 \leq D$). The applicable values of D are in the interval $[0, \|\text{pps}(k, w) - p\|_1]$ and clearly Δ-OPT$(w, p, 0) = p$ and Δ-OPT$(w, p, \|\text{pps}(k, w) - p\|_1) = \text{pps}(k, w)$. For general D, we obtain the distribution by combining the best increase of $D/2$ and the best decrease of $D/2$. The two are disjoint for $D \leq \|\text{pps}(k, w) - p\|_1$. Pseudocode is provided in Algorithm 4.

Lemma 6.5 *A representation of* Δ-OPT(w, p, D) *can be computed in* $O(n \log n)$ *time. The representation supports computing the output distribution for a particular value D in time linear in the number of entries with* $p_i \neq q_i$.

Proof Algorithm 4 computes Δ-OPT(w, p, D) in $O(n)$ time after computing the thresholds $\overline{\tau}(x)$ and $\underline{\tau}(x)$ for $x = D/2$, using Algorithms 2 and 3. If we are interested in a particular D, the algorithms can be stopped once the piece of the function containing $x = D/2$ is computed. This takes $O(n \log n)$ time from Algorithms 2 and 3. The complete representation is the combination of the representations of $\underline{\tau}(x)$ and $\overline{\tau}(x)$. If it is precomputed, Algorithm 4 returns the distribution in time linear in the number of modified entries. ■

We now show how to compute the Δ-stable distribution for the instance in Example 5.1.

Example 6.6 *On Example 5.1, the optimal PPS has changeout* $4/3$ *and thus we are interested in the* Δ*-stable distribution for* $D < 4/3$. *We do this by combining the best increase of* $D/2$, *which affects a prefix of the entries* $\{5, 4, 2\}$ *with the best decrease of* $D/2$, *which affects a prefix of the entries* $\{6, 3, 1\}$. *If* $D = 1$,

Algorithm 4 Compute the distribution Δ-OPT(w,p,D)

function OPT(w,p,D)
  $x \leftarrow D/2$
  $\underline{T} \leftarrow \underline{\tau}(x)$
  $\overline{T} \leftarrow \overline{\tau}(x)$
  if $\overline{T} = 0$ **then** $R \leftarrow x$
  for $i \in [n]$ **do**
    if $p_i \le \frac{w_i}{\underline{T}}$ **then**
      $q_i \leftarrow \frac{w_i}{\underline{T}}$
    else if $\overline{T} = 0$ **then**
      if $R > 0$ and $w_i = 0$ **then**
        $q_i \leftarrow \max\{0, p_i - R\}$
        $R \leftarrow R - p_i$
    else if $p_i < \frac{w_i}{\overline{T}}$ **then** ▷ $\overline{T} > 0$ and $p_i \in [\frac{w_i}{\overline{T}}, \frac{w_i}{\underline{T}})$
      $q_i \leftarrow \frac{w_i}{\overline{T}}$
    else ▷ $\overline{T} > 0$, $p_i \in [\frac{w_i}{\overline{T}}, \frac{w_i}{\underline{T}})$
      $q_i \leftarrow p_i$
return q

the best increase of $x = 0.5$ involves all three entries. We have $\underline{\tau} = 7.5$ and accordingly the entries $\{5,4,2\}$, which have weights $\{6,5,4\}$, have $q_5 = 6/7.5 = 0.8$, $q_4 = 5/7.5 = 2/3$, and $q_2 = 4/7.5 = 8/15$. The best decrease of $x = 0.5$ has $\overline{\tau} = 6$. Accordingly, $q_6 = 0$, $q_3 = 1/6$, and $q_1 = 2/6 = 1/3$. Therefore, the Δ-stable distribution for $D = 1$ is

$$q = \left(\frac{1}{3}, \frac{8}{15}, \frac{1}{6}, \frac{2}{3}, \frac{4}{5}, 0\right).$$

Figure 2 illustrates the thresholds $\overline{\tau}$ and $\underline{\tau}$ as a function of the changeout x. When the changeout is $D = 4/3$, the increase and decrease are $D/2 = 2/3$ and both thresholds meet the PPS threshold, reflecting that the Δ-stable solution is the best-fit PPS (4). ■

6.4 Computing an α-stable PPS distribution.

For a given stable solution, we can consider its changeout D from the starting distribution and the Lagrange multiplier a, which is the rate of decrease of the sum of variances per unit change after implementing a total change of D in sampling probabilities. The relation between the two is $\alpha(D) \equiv a = \underline{\tau}(D/2)^2 - \overline{\tau}(D/2)^2$.

Lemma 6.7 *A description of the function $\alpha(D)$, and a solution for an equation $\alpha(D) = a$, can be computed in $O(n \log n)$ time.*

Proof The function $\alpha(D)$ is a piecewise rational expression which is decreasing with D. Its breakpoints are the union of the breakpoints of $\underline{\tau}$ and $\overline{\tau}$. For each piece, we can compute the respective range of D values and of a values. To compute the α-stable distribution for a given a, we locate the piece in which a is in the range and solve for x the equation $a = \underline{\tau}(x)^2 - \overline{\tau}(x)^2$. Within a specified piece, this yields a quartic equation (finding the root of a polynomial of degree 4), and we take the root x that lies in the domain of the piece.

Once x is computed, we have the changeout $D = 2x$ which corresponds to a and the thresholds $\underline{\tau}(x)$ and $\overline{\tau}(x)$. We can substitute the thresholds in the initialization of Algorithm 4 and apply it to compute the

α-stable distribution. ∎

Combining these pieces provides the result claimed by Theorem 6.1.

7 Incremental stable PPS

We show how we can efficiently maintain an α-stable distribution when modifications to the weight vector are made one entry at a time.

Theorem 7.1 *An α-stable PPS distribution, and a corresponding sample, can be maintained in amortized* $\mathrm{poly}\log(n)$ *time per modification.*

Consider an α-stable distribution and the set of ratios w_i/p_i. Following the notation we used in the batch case, let $\underline{\tau}$ be the largest ratio with $p_i < 1$; and let $\overline{\tau}$ be 0 if there is an entry with zero weight and positive probability and otherwise the smallest ratio with $w_i > 0$. The difference of the squares is at most a: $\underline{\tau}^2 - \overline{\tau}^2 \leq a$. When the weight of an entry i is modified, its ratio changes. If this results in a maximum difference of squares exceeding a, we need to recompute the α-stable distribution. This event can happen only when either $p_i < 1$ and $w_i/p_i > \underline{\tau}$ or $p_i = 0$, $w_i > 0$ and $\overline{\tau} > 0$ or $w_i/p_i < \overline{\tau}$. The modification that involves the modified entry itself must be at one end of the ordered ratios list and a sequence of entries from the other end of the ordered ratios. We increase $\overline{\tau}$ to simultaneously compensate for decreasing $\underline{\tau}$ or vice versa, but noticing we never need to exceed the original threshold value at the affected end. The total number of modified probabilities, however, can be large, even when we modify the weight of a single entry. On the other hand, the expectation of the change in the sample (the L_1 distance between initial and final α-stable distributions) is at most 1 and we can ensure (as we outlined in the subsampling discussion) that there is at most one insertion and one deletion from the sample.

Our treatment of the incremental problem introduces a representation of the α-stable distribution, which allows for efficient tracking and modifications of sampling probabilities of sets of entries and also efficient maintenance of a corresponding sample. This is combined together with amortized analysis which charges the modifications to the representation to previous modifications in the input.

We now provide full details of the data structures we use. To gain intuition, we first consider the special case ($a = 0$) of maintaining a PPS distribution. Conceptually, entries are maintained in a sorted order according to their weight and we also track the respective τ value. We use a structure, such as Fenwick tree [7], that supports prefix sums, lookups, insertions and deletions in $O(\log n)$ time. When an entry is modified, it is deleted from its previous position and reinserted with its new weight. If the weight decreased (increased), the threshold can only decrease (respectively, increase). The respective new τ (new solution of (3)) can be computed with a logarithmic number of lookups. An actual sample can be maintained using permanent random numbers (PRN). This is facilitated by maintaining another sorted order according to the ratios w_i/u_i (where u_i is the PRN of entry i). The sample includes all entries with $w_i/u_i \geq \tau$. For this, we use a data structure that supports insertions, deletions, and lookups (of the smallest ratio that is larger than a value) in $O(\log n)$ time such as a self adjusting binary tree [22].

In the case of maintaining a stable sample for general a, probabilities in the stable sample only need to be modified when the ratio w_i/p_i of a modified input entry is outside the current $(\underline{\tau}, \overline{\tau})$ range. In the batch algorithm we worked with the ordered list of ratios w_i/p_i. The prefix and suffix of this list formed PPS samples, and thus, within the suffix and prefix of the ordered ratios list, the ratio order corresponds to the weights order. But in the middle range, for ratios in $(\underline{\tau}, \overline{\tau})$, the order of ratios may not correspond to the order of weights.

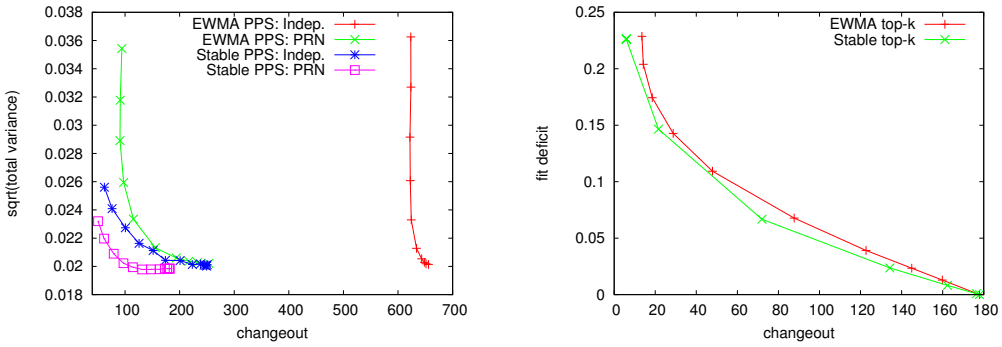

Figure 3: Left: Stable PPS uniformly and noticeably outperforms EWMA PPS. Permanent Random Numbers (PRN) improve both methods. Right: Similar performance for Stable and EWMA top-k, although computational aspects favor Stable top-k.

A critical observation for our analysis is that once two entries are "processed together" in a prefix or suffix of the sorted ratios list, and thus the heavier one precedes the lighter one, their relative position remains the same in the sorted ratios list, even if they are no longer members of the prefix or suffix (have ratios below $\underline{\tau}$ or above $\overline{\tau}$). This situation only changes, i.e. $w_i > w_j$ and $w_i/p_i < w_j/p_j$ when the weight of at least one of the entries is modified. Intuitively, in our analysis, we charge the cost of "correcting" this order when an entry works its way back into a prefix or suffix to the modifications in the input.

The main structure that we use to support this analysis is what we call a *stretch*, which maintains a subset of entries. The stretch supports the following operations in $O(\log n)$ time: deletion of an entry, splitting a stretch into two stretches that include all entries with weights that are respectively below and above some value, merging two stretches with non-overlapping weight ranges, performing a lookup of the smallest/largest weight above some value, and returning the weight of all entries that are below or above a certain value. Consequently, two stretches with overlapping weight ranges can be merged in time proportional to $\log(n)$ times the number of interleaved segments in their joint sorted order.

Instead of treating entries as single entities, they are grouped to stretches. All entries currently in the same stretch share the same ratio $\tau = w_i/p_i$ (which we refer to as the threshold of the stretch) or else have $w_i > \tau$ and $p_i = 1$. We thus assign a ratio to a stretch as the range that spans its threshold and its largest weight. All stretches are maintained in a structure which supports insertion, deletion, and accessing stretches with maximum or minimum threshold in $O(\log n)$ time. This structure corresponds to the "sorted list of ratios" in the batch algorithm.

When an entry is modified, it is deleted from its current stretch and inserted as a single-entry stretch. To recompute the distribution in response to the modification, we apply an extension of our batch algorithm that can manipulate stretches: If the update produces a higher maximum ratio or lower minimum ratio and α-stability is violated, we obtain new values for $\underline{\tau}$ and $\overline{\tau}$ while processing stretches by decreasing and increasing ratios, as done (for single entries) by the batch algorithm.

As the prefix (and suffix) of the "sorted" ratios are processed in decreasing ratio order, stretches are merged into a single prefix (or a single suffix) stretch. We stop when the difference of squares is exactly a. The cost of merging stretches depends on the number of interleaved sorted segments of stretches, and each segment can be "charged" to a modification of an item.

8 Illustration and Comparison with Time Decay

We provide a brief illustrative performance comparison of Stable PPS and Stable top-k against their time-decayed counterparts. We can think of an application where the input is a demand or load vector of a system. A dashboard is shown to a user monitoring the system. We refer to the content of the dashboard, which is a set of contributors (aiming for top-k or a representative sample), as a *summary*. A top-k set shows the heaviest contributors whereas a PPS sample shows representative contributors.

We would like the dashboard to show an accurate view of the current "state" (good fit) while minimizing fluctuations (changeout). We may want to be able to optimize a combination of both (α-stability) or simply limit changeout in a time interval (Δ-stability).

Our algorithms for the respective stable problems compute a precise optimal tradeoff of fit and stability of the output. We compare with another method: Computing a best fit for an Exponentially Weighted Moving Average (EWMA) of the input. The EWMA method achieves some tradeoff indirectly, by smoothing the input. The tradeoff with EWMA is obtained by controlling the geometric decay parameter. A slower decay makes for a smoother input which is less similar to the current input state, but we expect to have lower changeout cost. The disadvantages of EWMA is that it provides no guarantees on quality and does not necessarily produce an optimal or even a monotone tradeoff. Moreover, to be able to control changeout we may need to work with a range of decay parameters at the same time, requiring us to store and maintain a large state.

Experimental Set-up. The data comprises IP flow records aggregated over the 144 ten-minute periods over the span of a day, obtained from a large ISP. On average, there were $61,292$ (anonymized) keys in each period t, from a total of $2,516,594$ unique keys. The results shown are for a summary size of $k = 1,000$ in each case.

For the time decayed version we constructed for each key i an EWMA $\tilde{x}_{i,t}$ of the process $x_{i,t}$. In EWMA PPS, at each time period t, we perform PPS selection over the keys present but using EWMA weights, i.e., the weight set $\{\tilde{x}_{i,t} : x_{i,t} > 0\}$, and formed the Horvitz-Thompson estimate of the true weight as an auxiliary variable, i.e., $x_{i,t}/\tilde{p}_{i,t}$ for selected items, where $\tilde{p}_{i,t}$ is the selection probability. In EWMA Top-k we selected at each time t the set $\tilde{S}_t$ of k items of highest EWMA weight $\tilde{x}_{i,t}$, and as measure of fit computed the corresponding sum of original weights $\sum_{i\in\tilde{S}_t} x_{i,t}$. By controlling the geometric decay parameter of EWMA we achieve some tradeoff of fit and changeout. With faster decay, the quantities $\tilde{x}_{i,t}$ are closer to $x_{i,t}$ and we expect better fit but also higher changeout cost, since faster decay also makes $\tilde{x}_{i,t}$ less similar to $\tilde{x}_{i,t'}$ for $t' < t$.

PPS Sampling: Fit is measured as the square root of sum of weight estimation variances. Fit and changeout costs are averaged over the 144 periods. For EWMA PPS we swept the mean m of the geometric decay from 1 (no history) to 64 periods. We computed the Δ-stable PPS distributions sweeping target maximal changeout cost D between 0 and the target sample size $k = 1,000$. We evaluate independent (pseudo)random number generation and coordinated randomization, using permanent random numbers (PRN) [4], for each key. These methods determine how the sample changes when the probabilities change. With PRN, we use subsampling (Algorithm 1) to make the samples as similar as possible when probabilities change (This is the stable PPS algorithm we presented). With independent sampling, the sample is redrawn. Independent sampling is thus expected to yield worse tradeoffs and is presented for comparison and to evaluate the benefits of subsampling.

Figure 3 shows the stability-fit tradeoffs for Stable PPS and EWMA PPS. We can see that EWMA PPS with independent random numbers has much larger changeout cost (in the range 620–650) than the other methods. Using PRN with EWMA PPS much reduces the change cost. Stable PPS performs noticeably

better, achieving very small changeout (around 50 for the PRN variant) with relatively little growth in error as changeout is reduced (about 15% growth compared with about 70% growth for EWMA). PRN has little impact on variance because, for a given input set, marginal variance is unchanged.

Top-k: Figure 3 shows stability-fit tradeoffs for Stable top-k and EWMA top-k, where fit is represented by the deficit between the total weight of the top-k estimate, and the true total top-k weight. Fit and changeout are again averaged over the 144 periods. The performance of both are similar, although the stable version does obtain a better tradeoff. Nevertheless, we note that stable methods are preferable to EWMA methods for both the theoretical guarantees of an optimal tradeoff and for computation reasons, when we would like the ability to control the tradeoff at each iteration. With the stable algorithms, the stability or flexibility parameters can be modified directly at each iteration, whereas with time-decay these parameters can be controlled only indirectly through the decay parameters. Changing decay parameters, however, requires extensive historical data to be retained, and rescanned for each change.

9 Stable Extensions of Optimization Problems

In general, we can consider the stable extension of almost any optimization problem. Theorem 4.1 shows that for a natural class of problems, α-stability reduces to a single modified instance of the original optimization problem. However, there are many other problems which do not admit such a reduction. We considered the case of PPS sampling, and showed procedures which find the optimal solution. Also of interest are those problems where the original optimization is known to be hard: then we seek approximation algorithms for the stable extension (making use of Theorem 4.1 if possible).

In this section, we provide further examples of problems that naturally admit stable extensions:

In the *0/1 knapsack problem*, items have values x_i and weights w_i, and we have a parameter W. The goal is to find a set of items of maximum value, subject to an upper bound of W on the sum of their weights, $\sum_{i \in S} w_i \leq W$. Under additive changeout cost, this fits the requirement of Theorem 4.1, and so any approximation algorithm for 0/1 knapsack provides the corresponding approximation for the α-stable version of the problem.

In *shortest-paths routing* the output is a tree routing to a single destination. The fitness is the (weighted) sum, over nodes, of the path length to the root. The changeout cost between two trees is the set difference (viewing trees as directed to the destination), which corresponds to the number of modifications to the routing table for the destination.

The *k-set cover problem* is, given a fixed collection of sets, to pick at most k to maximize coverage of x. The fitness is the sum of weights of covered items, and we consider the uniform changeout cost function. We show that the α-stable k-cover is hard by outlining a reduction to the optimum k-cover on a modified input. Our modified instance contains all the original items and sets. It also contains a new item that is unique for each set in the current output S. The new items have weights a. We argue that a cover is α-stable for a if and only if it is an optimal k-cover of the modified input.

9.1 k-center Clustering

It is natural to consider the stable extension of various *clustering* problems. Here, we wish to minimize the cost of the new clustering plus the cost of switching out some of the cluster centers. In this section, we show how to find a constant factor approximation for the k-center objective in this setting.

Given the stable extension of any optimization problem, an α-optimal solution can always be computed by exhaustive search on possible outputs. A method with smaller search space is to first construct an algorithm for an extended version of the optimization problem which allows a part of the output to be fixed. This

extension is a special case of the stable extension. We can then perform a smaller search over all subsets of the current output, and apply the (restricted) extended algorithm to each instance. Some problems naturally yield to such extensions: for covering problems, we remove items covered by the fixed component and apply the (approximate) covering algorithm to what remains.

A less straightforward case arises in the context of clustering. Here, we want to extend a given subset of cluster centers with some additional centers to minimize a particular cluster objective. For the k-center objective, the existing 2-approximation algorithm can be extended to handle fixed facilities (this "restricted" extension is also interesting in itself). More formally, for points in a metric space m, we have:

Lemma 9.1 *Given a fixed set of centers F and a set of n points P, we can find a set of clusters C of size s that guarantees to 2-approximate the k-center objective, i.e.*

$$\max_{p \in P} \min_{c \in C \cup F} m(p,c) \leq 2 \min_{C:|C|=s} \max_{p \in P} \min_{c \in C \cup F} m(p,c)$$

Proof Here, we adapt the 2-approximation algorithm due to Gonzalez [9]. Starting with $C = \emptyset$, we pick the point from the input that is furthest from the current centers in $C \cup F$, i.e.

$$C \leftarrow C \cup \{\arg\max_{p \in P} \min_{c \in C \cup F} m(p,c)\}$$

and iterate until $|C| = s$. This requires $O((|F|+s)n)$ distance computations.

To see the approximation guarantee, consider the next point q that would be added to S if we continued the algorithm for one more iteration. This is some distance d from its closest center in $C \cup F$. Therefore, we have that all points in $C \cup \{q\}$ are separated by d, and further that every point in $C \cup \{q\}$ is at least d from every point in F. This can be seen by contradiction: if any point in C were closer than d, then it would not have been picked as the furthest point ahead of q.

This provides a witness for the cost of the clustering. There are two cases: (i) some point in $C \cup \{q\}$ is assigned to some center in F under the optimal clustering; or (ii) some pair of points in $C \cup \{q\}$ are assigned to the same center under the optimal clustering. If (i) holds, then immediately we have that the cost of the optimal clustering is at least d. If (ii) holds, then we have two points separated by distance d in the same cluster. Then the cost of the optimal clustering is at least $d/2$, which happens when their cluster center is distance $d/2$ from each (it cannot be closer to both without violating the triangle inequality).

On the other hand, since q is the furthest from its closest center in $C \cup F$, it follows that *all* points are within d of their closest center, and therefore the cost of the clustering found by the algorithm is d, which is at most twice the optimal cost. ∎

This indicates that we can find a 2-approximation for the stable extension by considering all subsets of the current clustering, and applying the above lemma. However, for subsets larger than $k/2$, we can simply work with the (approximate) solution that switches out all k centers for the optimal k-center clustering on the current set: this at most doubles the changeout cost, while improving the clustering cost. Thus, we only need to consider subsets of size at most $k/2$.

10 Concluding Remarks

We formalized a model of stability and fit, where the benefits of an improved solution have to be traded off against the costs of changing from the current status quo. In particular, our model was inspired by the need to improve the performance of a production application management system, making changing

performance and usage data presented in a dashboard monitored by people. From this motivation, we provide stable extensions of PPS sampling and algorithms for some basic optimization problems (top-k, MST, and assignment). Our solutions are efficient and also apply to a dynamic version where we wish to efficiently maintain a tradeoff when the input is only slightly modified at each step.

References

[1] P. K. Agarwal, D. Eppstein, L. J. Guibas, and M. R. Henzinger. Parametric and kinetic minimum spanning trees. In *FOCS*, 1998.

[2] S. Babu, P. Bizarro, and D. J. DeWitt. Proactive re-optimization. In *ACM SIGMOD*, 2005.

[3] A. Borodin, N. Linial, and M. E. Saks. An optimal on-line algorithm for metrical task system. *J. ACM*, 39(4):745–763, 1992.

[4] K. R. W. Brewer, L. J. Early, and S. F. Joyce. Selecting several samples from a single population. *Australian Journal of Statistics*, 14(3):231–239, 1972.

[5] E. Cohen, N. Duffield, C. Lund, M. Thorup, and H. Kaplan. Efficient stream sampling for variance-optimal estimation of subset sums. *SIAM J. Comput.*, 40(5), 2011.

[6] E. Cohen and M. Strauss. Maintaining time-decaying stream aggregates. *J. Algorithms*, 59:19–36, 2006.

[7] P. M. Fenwick. A new data structure for cumulative frequency tables. *Softw. Pract. Exper.*, 24(3):327–336, 1994.

[8] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.*, 18:30–55, 1989.

[9] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38(2-3):293–306, 1985.

[10] D. Gusfield. Parametric combinatorial computing and a problem of program module distribution. *J. Assoc. Comput. Mach.*, 30:551–563, 1983.

[11] J. Hájek. Asymptotic theory of rejective sampling with varying probabilities from a finite population. *The Annals of Mathematical Statistics*, 35(4):1491–1523, 1964.

[12] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4), 2001.

[13] D. G. Horvitz and D. J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47(260):663–685, 1952.

[14] R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

[15] R. M. Karp and J. B. Orlin. Parametric shortest path algorithms with an application to cyclic staffing. *Discrete Applied Mathematics*, 3:37–45, 1981.

[16] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proc. of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 481–492, Berkeley, Calif., 1951. University of California Press.

[17] N. Megiddo. Combinatorial optimization with rational objective functions. *Math. of Oper. Res.*, 4:414–424, 1979.

[18] P. Sankowski. Faster dynamic matchings and vertex connectivity. In *SODA*, pages 118–126, 2007.

[19] C-E. Särndal, B. Swensson, and J. Wretman. *Model Assisted Survey Sampling*. Springer, 1992.

[20] H. Shachnai, G. Tamir, and T. Tamir. A theory and algorithms for combinatorial reoptimization. In *LATIN*, 2012.

[21] R. Singh and N. S. Mangat. *Elements of survey sampling*. Springer-Verlag, New York, 1996.

[22] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, July 1985.