Approximation Guarantees for Data-Driven Algorithm Design

Avrim Blum

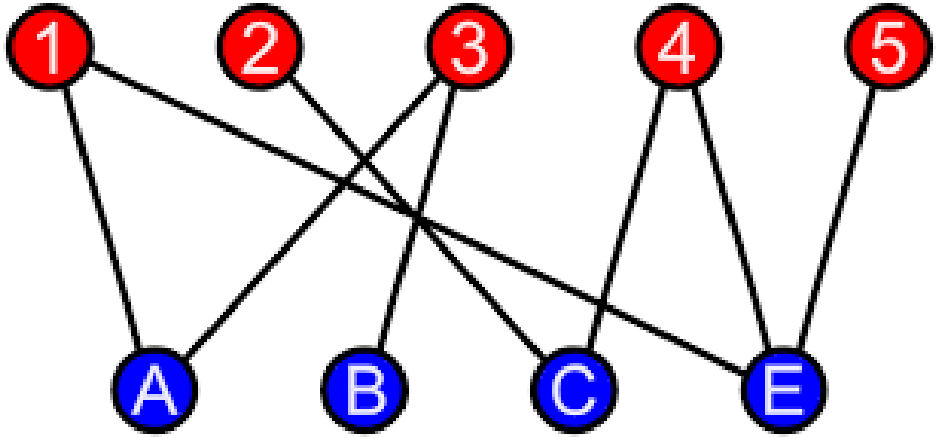
TTIC

Work based on “Competitive Strategies to use Warm Start Algorithms with Predictions” (SODA 2025) joint with Vaidehi Srinivas, plus thoughts based on a course co-taught with Dravy Sharma

Data-Driven Algorithm Design

If you want an algorithm to do particularly well on problem instances coming from some domain X, why not train/tune it on typical instances from that domain?

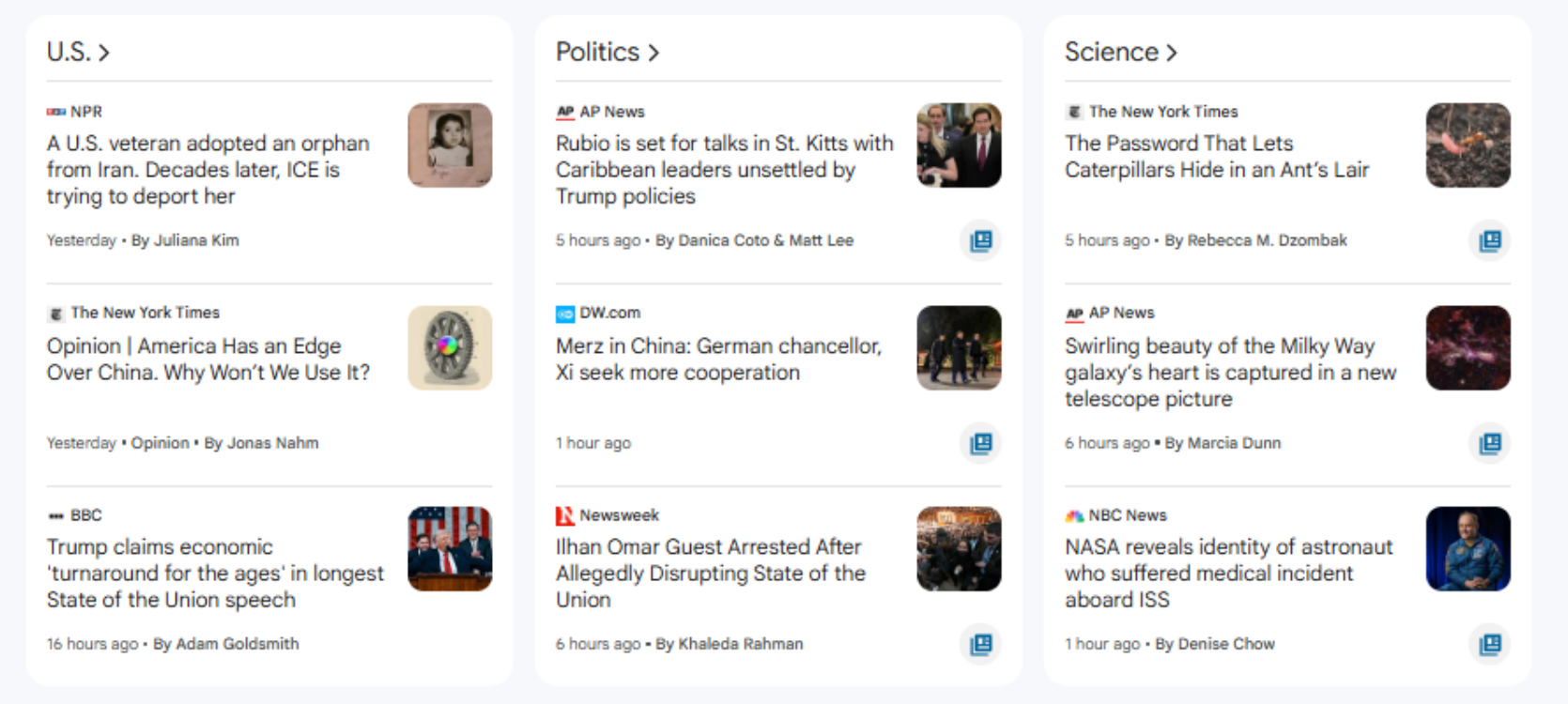
➢Solving matching problems coming up in a particular resource-allocation domain



Data-Driven Algorithm Design

If you want an algorithm to do particularly well on problem instances coming from some domain X, why not train/tune it on typical instances from that domain?

- Solving matching problems coming up in a particular resource-allocation domain
- Clustering news articles by topic



Data-Driven Algorithm Design

If you want an algorithm to do particularly well on problem instances coming from some domain X, why not train/tune it on typical instances from that domain?

- Solving matching problems coming up in a particular resource-allocation domain
- Clustering news articles by topic
- Etc.

Data-Driven Algorithm Design

If you want an algorithm to do particularly well on problem instances coming from some domain X, why not train/tune it on typical instances from that domain?

- Solving matching problems coming up in a particular resource-allocation domain
- Clustering news articles by topic
- Etc.

In fact, this is what companies implementing algorithms do. Is there something we can say about this theoretically?

Yes. Recent direction with a lot of exciting work, interesting mathematical and algorithmic techniques.

Theoretical Guarantees for Data-Driven Algorithm Design

High-level idea:

- Define a parameterized family of algorithms, typically motivated by one or more algorithms with good worst-case or good empirical performance.
- Given a sample of typical problem instances (iid draws from some distribution), aim to learn a good setting of the parameters that will do well on future instances.
  - ➢ Challenging statistical/generalization questions, especially when algorithm behavior can be a highly discontinuous function of parameters. Brings in a lot of interesting math.
  - ➢ Challenging algorithmic questions.
- Can also aim for online regret guarantees (with again challenging analytical and algorithmic questions).

To learn more, I taught a course on it with Dravy Sharma: https://home.ttic.edu/~avrim/MLforAD25/. Also, Nina Balcan has an excellent book chapter on the topic.

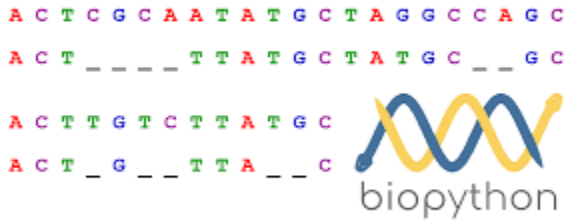
Theoretical Guarantees for Data-Driven Algorithm Design

A non-exhaustive list of some particularly nice papers:

- Maria-Florina Balcan, Vaishnavh Nagarajan, Ellen Vitercik, Colin White: Learning-Theoretic Foundations of Algorithm Configuration for Combinatorial Partitioning Problems. COLT 2017.
- Maria-Florina Balcan, Travis Dick, Colin White: Data-Driven Clustering via Parameterized Lloyd's Families. NeurIPS 2018.
- Maria-Florina Balcan, Travis Dick, Ellen Vitercik: Dispersion for Data-Driven Algorithm Design, Online Learning, and Private Optimization. FOCS 2018.
- Maria-Florina Balcan, Dan F. DeBlasio, Travis Dick, Carl Kingsford, Tuomas Sandholm, Ellen Vitercik: How much data is sufficient to learn high-performing algorithms? generalization guarantees for data-driven algorithm design. STOC 2021, JACM 2024.
- Maria-Florina Balcan, Dravyansh Sharma: Learning Accurate and Interpretable Decision Trees. UAI 2024. (Best student paper award)

Example: Biological sequence alignment (DNA, protein, etc)

[Balcan et al, "How much data is sufficient to learn high-performing algorithms?", JACM 2024]



- Sequences typically aligned using Dynamic Programming, with different costs for mismatches, indels, number of gaps, etc.
- DDAD problem: given a collection of pairs, with known "ground truth" alignment for them, find costs such that DP with those costs will produce correct matches on new pairs.
  - ➢ Statistical/generalization question: how much training data do you need to guarantee that good performance on the training data will whp translate to good performance on new pairs?
  - ➢ Algorithmic question: how can you efficiently find the best setting of costs on the training data?

In this case, "good performance" corresponds to solution quality. In other cases (like what I'll focus on today), "good performance" corresponds to running time.

Approximation Algs for Data-Driven Algorithm Design

Idea: use ideas from approximation algorithms (and competitive analysis) to simplify the algorithmic challenge, and perhaps also make it easier to compare to stronger benchmarks, by willingly giving up some constant (or logarithmic or polynomial) factors.

- Focus on "warm start" initialization / meta-learning, which is a particularly well-behaved setting.
- We'll even see a cameo appearance of the k-server problem.

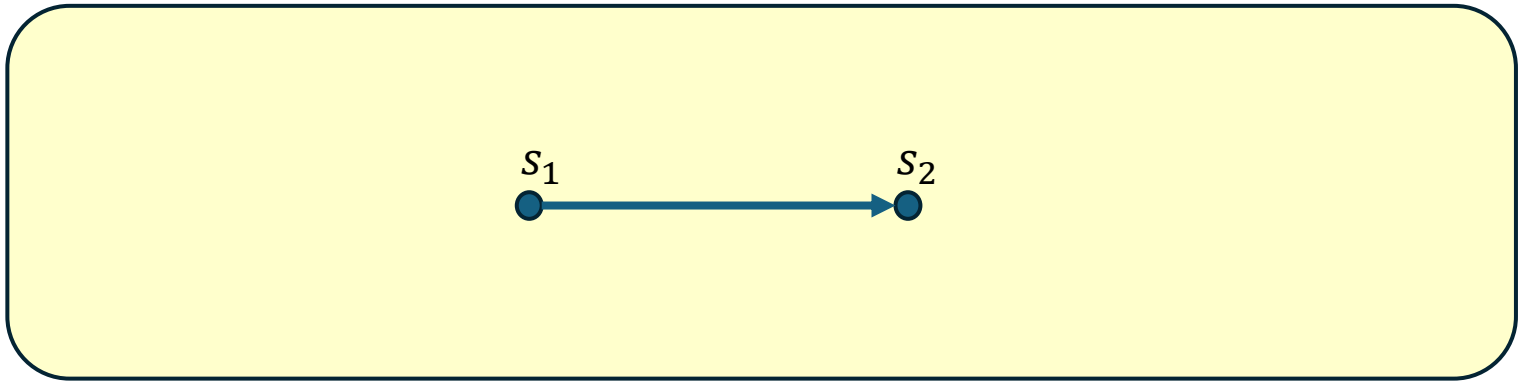
- Note: we'll be losing multiplicative factors with respect to our benchmark. So, the perspective/hope is that the thing we're competing with (e.g., the optimal initialization strategy in hindsight) is asymptotically much better than a generic start.

Warm-start initialization setup

Many algorithms run faster if you initialize them with a solution that is "close" to the right answer.

- E.g., Ford-Fulkerson max-flow algorithm
  Sami Davies, Benjamin Moseley, Sergei Vassilvitskii, and Yuyan Wang. “Predictive flows for faster Ford-Fulkerson.” (ICML 2023)
- Hungarian algorithm for min-weight perfect matching
  Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. “Faster matchings via learned duals.” (NeurIPS 2021)
- Gradient descent

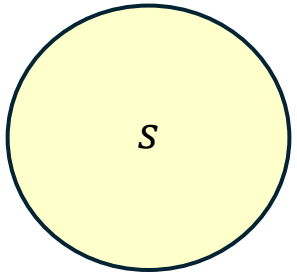
Can think of this as having a metric space on solutions, where $d(s_1, s_2)$ represents the cost if you start from s_1 and solve a problem instance I whose solution is s_2.



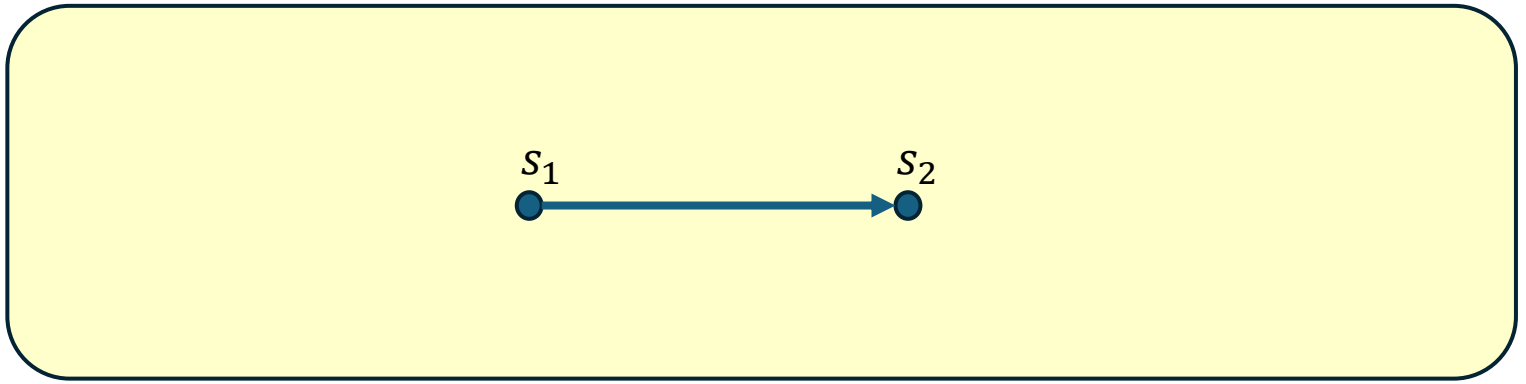
Warm-start initialization setup

Note: In reality, it takes time to check a correct solution even if one is handed to you, so can allow $d(s, s) > 0$. It doesn't affect anything. But we will assume symmetric.

Running the algorithm from a starting point s is like growing a ball around s.

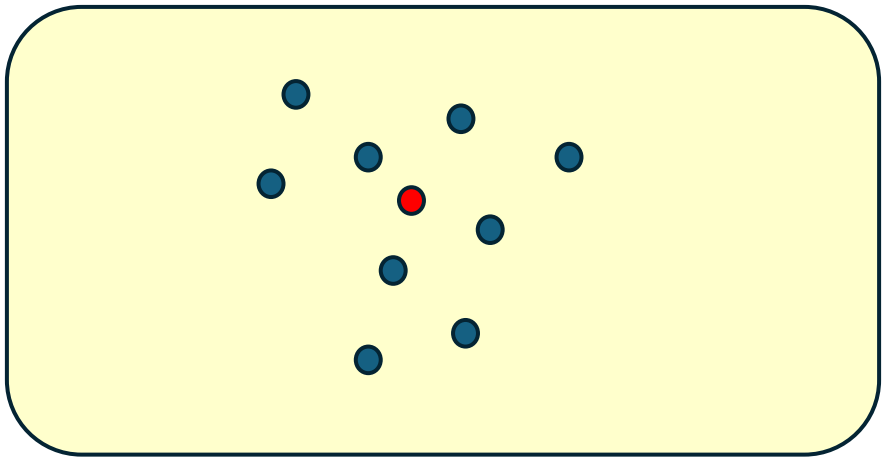


Can think of this as having a metric space on solutions, where $d(s_1, s_2)$ represents the cost if you start from s_1 and solve a problem instance I whose solution is s_2.



What's known? (In terms of theoretical guarantees)

- Recent work on learning a good initialization for a distribution of problem instances.
  - Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. "Faster matchings via learned duals." (NeurIPS 2021)
  - Sami Davies, Benjamin Moseley, Sergei Vassilvitskii, and Yuyan Wang. "Predictive flows for faster Ford-Fulkerson." (ICML 2023)
- And on competing with best single initialization for an adversarial sequence.
  - Mikhail Khodak, Maria-Florina Balcan, Ameet Talwalkar, and Sergei Vassilvitskii. "Learning predictions for algorithms with predictions." (NeurIPS 2022) [Also improves the sample-complexity bounds for distributional case]



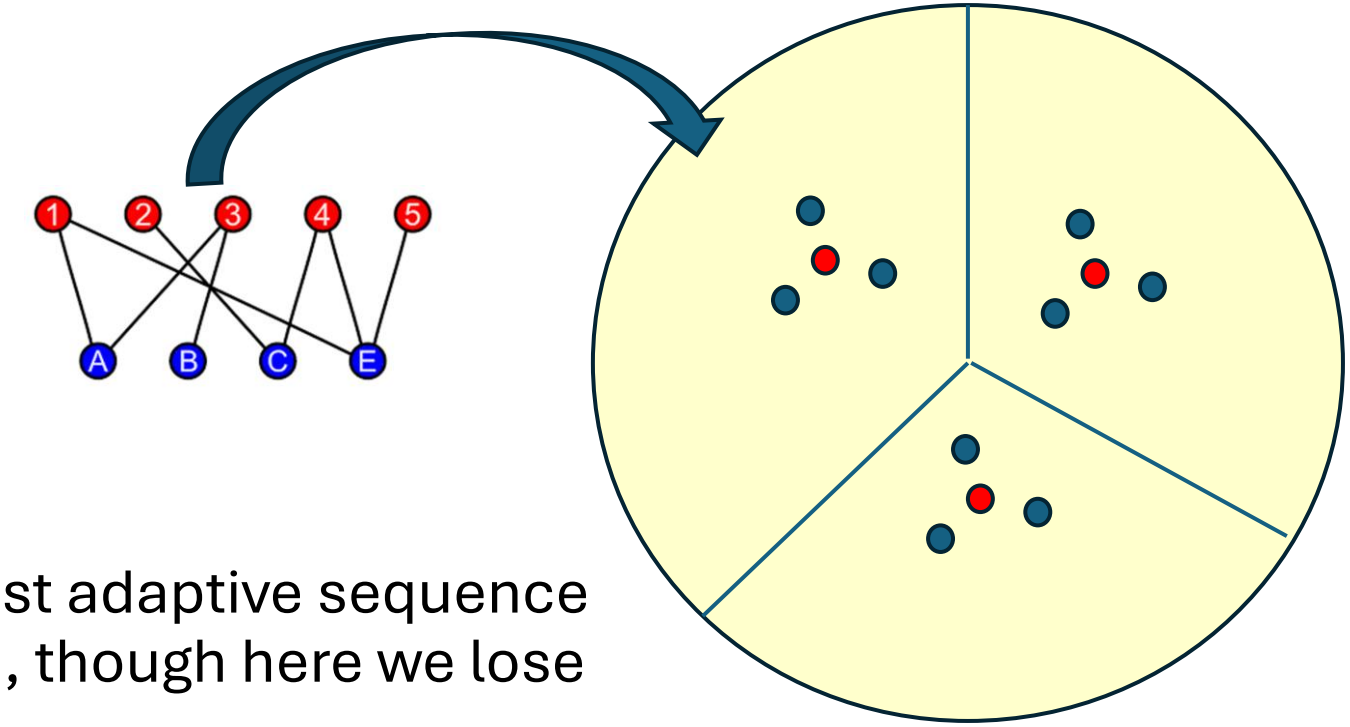
What's known? (In terms of theoretical guarantees)

- Recent work on learning a good initialization for a distribution of problem instances.
  - Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. "Faster matchings via learned duals." (NeurIPS 2021)
  - Sami Davies, Benjamin Moseley, Sergei Vassilvitskii, and Yuyan Wang. "Predictive flows for faster Ford-Fulkerson." (ICML 2023)
- And on competing with best single initialization for an adversarial sequence.
  - Mikhail Khodak, Maria-Florina Balcan, Ameet Talwalkar, and Sergei Vassilvitskii. "Learning predictions for algorithms with predictions." (NeurIPS 2022) [Also improves the sample-complexity bounds for distributional case]

But what if the best single initialization is not good enough? Maybe if we're willing to lose some multiplicative factors we can compete against stronger benchmarks.

Two kinds of results in this direction

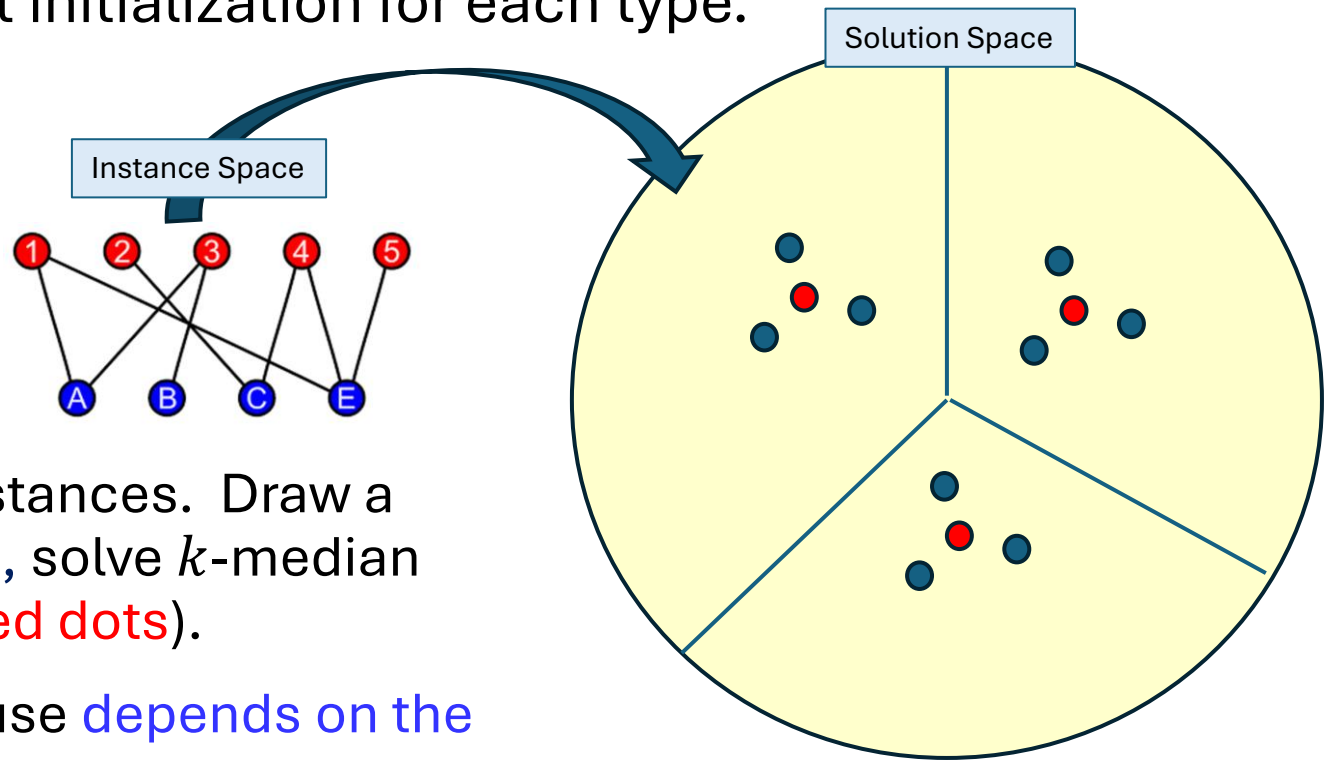
1. Aim to learn a mapping $h \in H$ from features of the instance to a "type" in $\{1, \ldots, k\}$ where we will use a different initialization for each type.



2. Online algorithm competing with best adaptive sequence warm starts (or set of k warm starts), though here we lose polynomial factors in k.

Learning a mapping

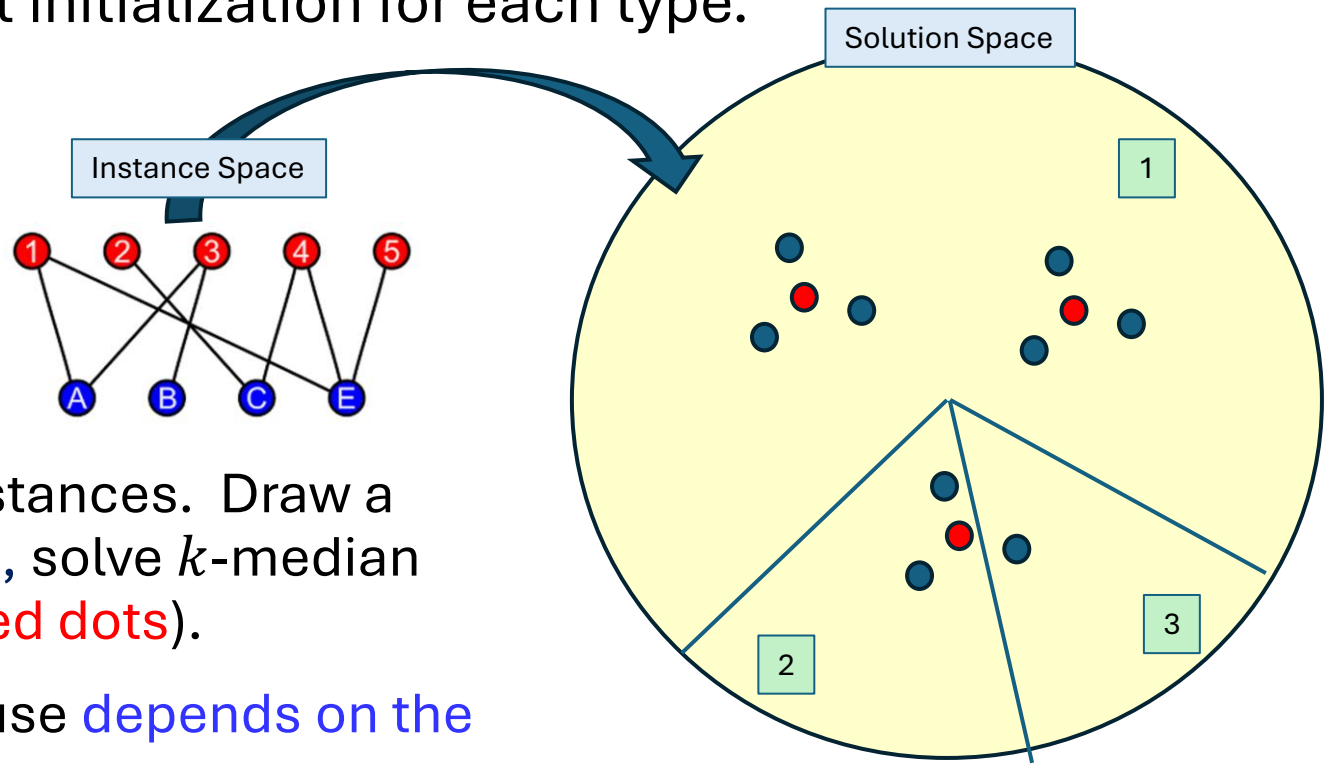
1. Aim to learn a mapping $h \in H$ from features of the instance to a "type" in $\{1, \ldots, k\}$ where we will use a different initialization for each type.



- Assume we have a distribution over instances. Draw a sample, find their solutions (blue dots), solve k-median problem to find best k initializations (red dots).
- Issue: the best set of initializations to use depends on the mapping h.

Learning a mapping

1. Aim to learn a mapping $h \in H$ from features of the instance to a "type" in $\{1, \ldots, k\}$ where we will use a different initialization for each type.

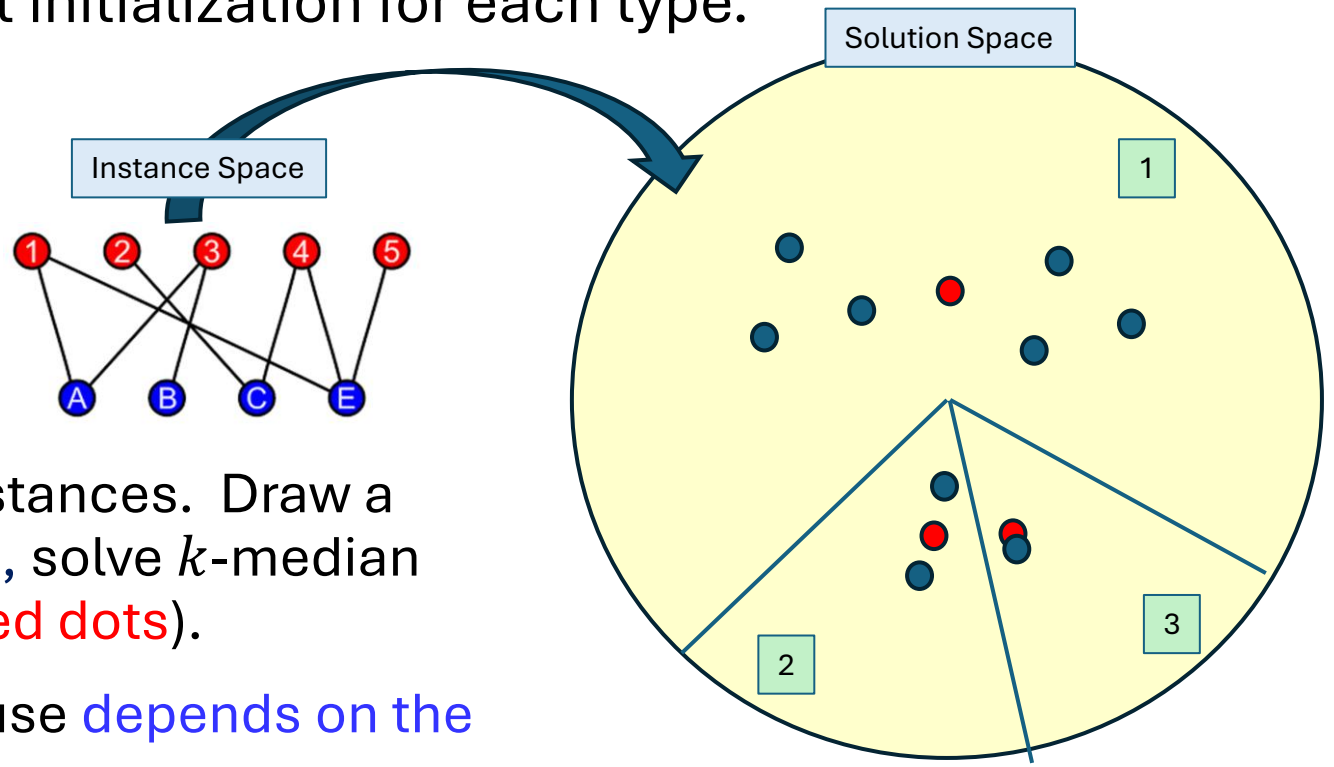


- Assume we have a distribution over instances. Draw a sample, find their solutions (blue dots), solve k-median problem to find best k initializations (red dots).
- Issue: the best set of initializations to use depends on the mapping h.

Of course, the first mapping is better, but might not exist in H. Mapping needs to be simple (much easier to compute than solving for a solution), so we don't expect a perfect h to exist.

Learning a mapping

1. Aim to learn a mapping $h \in H$ from features of the instance to a "type" in $\{1, \ldots, k\}$ where we will use a different initialization for each type.

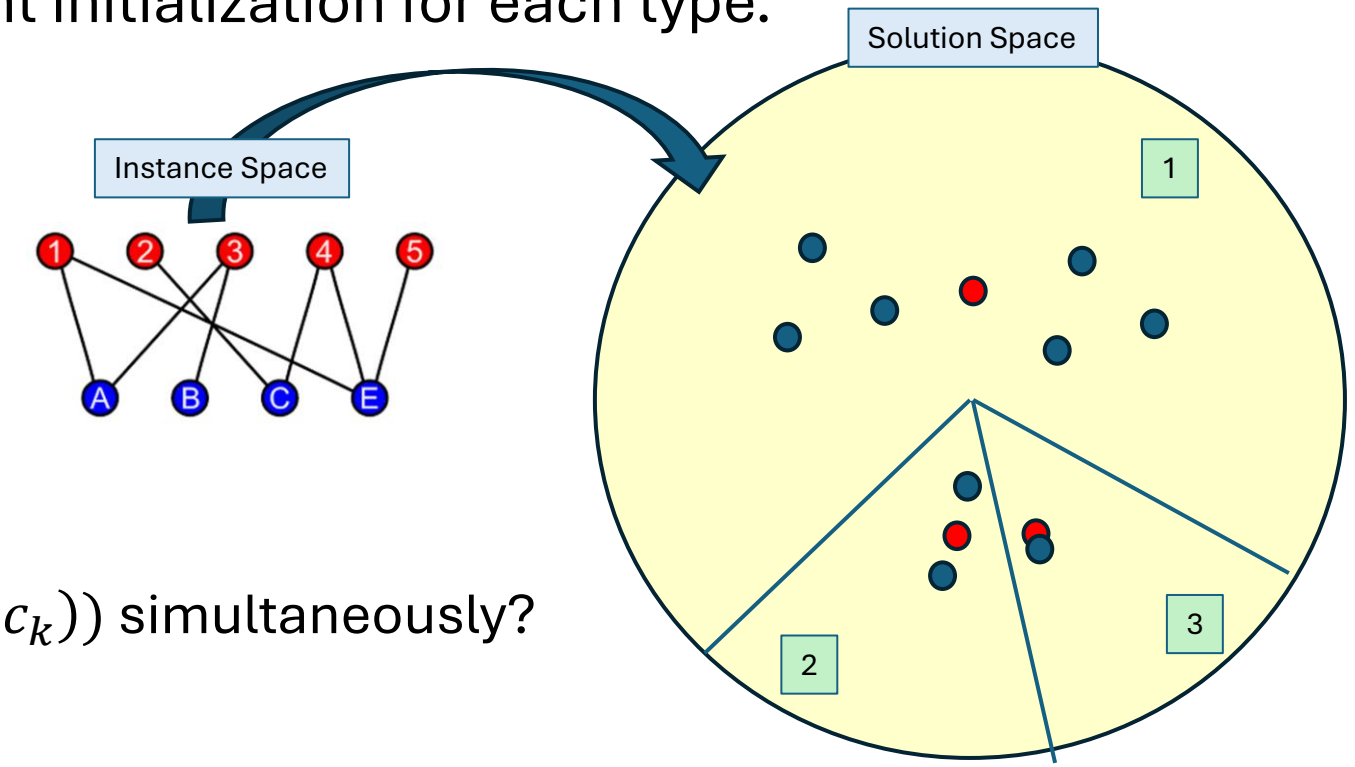


- Assume we have a distribution over instances. Draw a sample, find their solutions (blue dots), solve k-median problem to find best k initializations (red dots).
- Issue: the best set of initializations to use depends on the mapping h.

Of course, the first mapping is better, but might not exist in H. Mapping needs to be simple (much easier to compute than solving for a solution), so we don't expect a perfect h to exist.

Learning a mapping

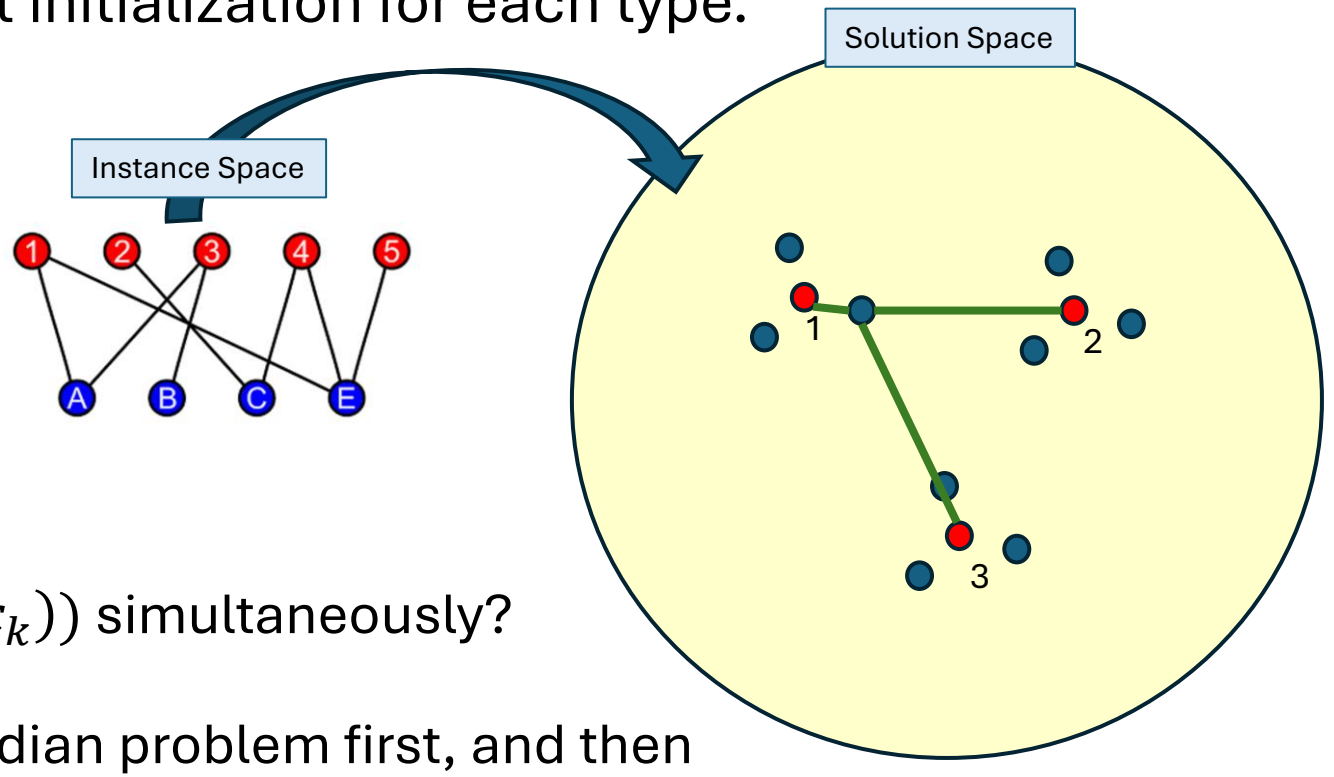
1. Aim to learn a mapping $h \in H$ from features of the instance to a "type" in $\{1, \ldots, k\}$ where we will use a different initialization for each type.



Q: Do we need to solve for a pair $(h, (c_1, \ldots, c_k))$ simultaneously?

Learning a mapping

1. Aim to learn a mapping $h \in H$ from features of the instance to a "type" in $\{1, \dots, k\}$ where we will use a different initialization for each type.



Q: Do we need to solve for a pair $(h, (c_1, \dots, c_k))$ simultaneously?

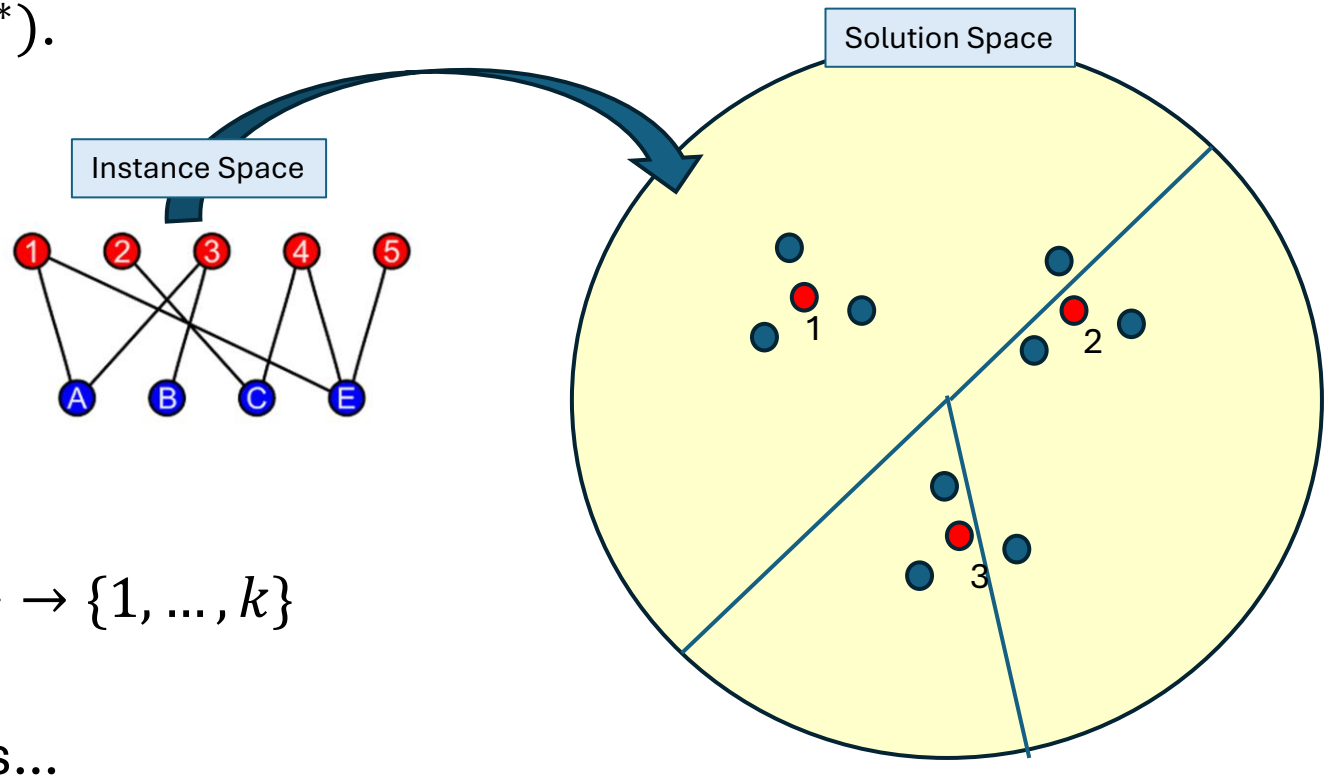
Or can we just solve (or apx solve) the k-median problem first, and then aim to find the mapping $h \in H$ of lowest loss wrt those centers?

Notation: sample $S = \{(I_j, s_j)\}$.

Learning a mapping (approximation algorithm)

Claim: If we optimistically solve the k-median problem first, getting centers $c_1, \dots, c_k$, and then find the mapping $h \in H$ of lowest loss wrt these centers $\sum_{I_j \in S} d(c_{h(I_j)}, s_j)$, then this will be within a factor of 3 of the best pair (h^*, c^*).

(and then we can argue generalization using existing sample-complexity arguments)



Technical assumption:

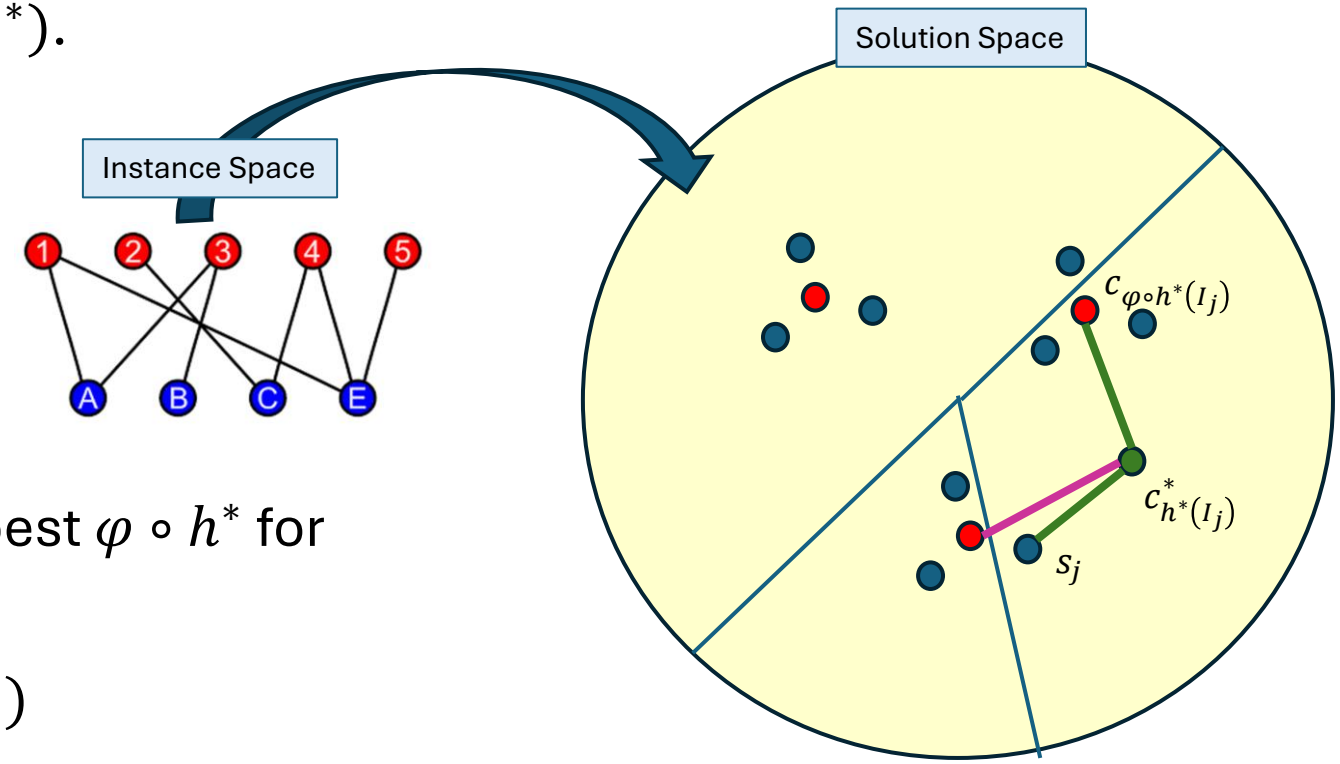
- If $h \in H$ then $\varphi \circ h \in H$ for all $\varphi: \{1, \dots, k\} \to \{1, \dots, k\}$

Analysis idea: several triangle inequalities…

Learning a mapping (approximation algorithm)

Claim: If we optimistically solve the k-median problem first, getting centers $c_1, \ldots, c_k$, and then find the mapping $h \in H$ of lowest loss wrt these centers $\sum_{I_j \in \mathcal{S}} d(c_{h(I_j)}, s_j)$, then this will be within a factor of 3 of the best pair (h^*, c^*).

(and then we can argue generalization using existing sample-complexity arguments)

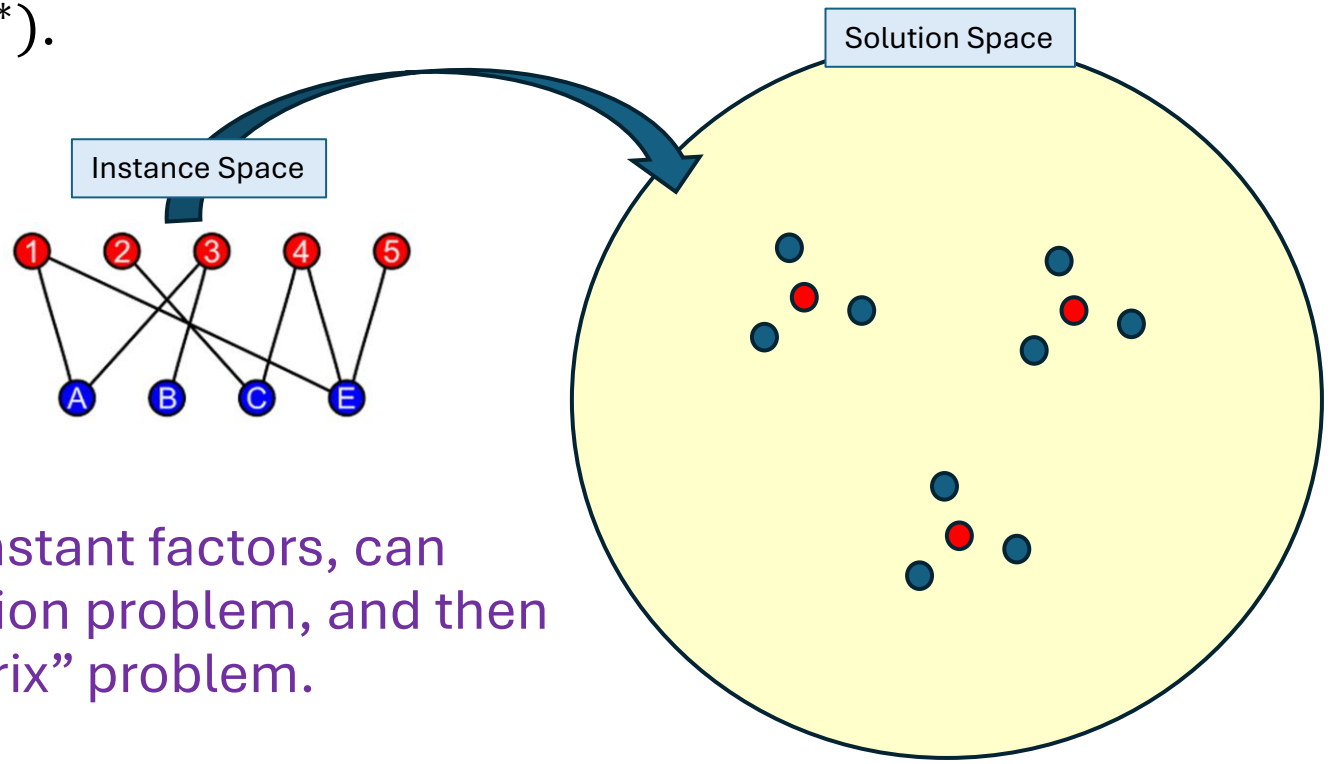


Analysis idea:

- Cost of h for centers $c_1, \ldots, c_k \leq$ cost of best $\varphi \circ h^*$ for centers $c_1, \ldots, c_k$.
- $\leq \sum_{I_j \in \mathcal{S}} d\left(c_{\varphi \circ h^*(I_j)}, c^*_{h^*(I_j)}\right) + d(c^*_{h^*(I_j)}, s_j)$
- $\leq$ (cost of unconstrained optimum) $+ 2 \cdot$ (cost of (h^*, c^*))
- $\leq 3 \cdot$ (cost of (h^*, c^*)).

Learning a mapping (approximation algorithm)

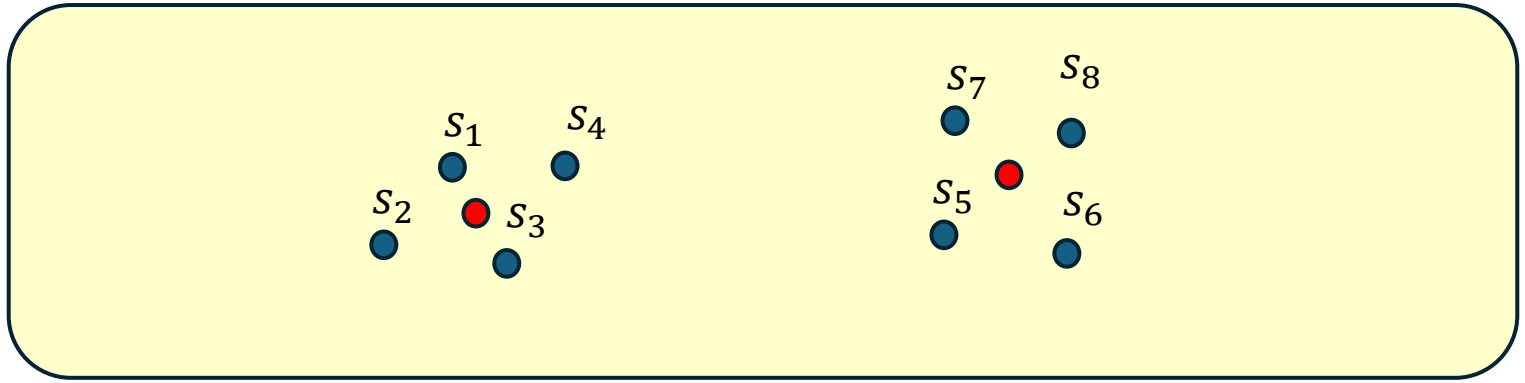
Claim: If we optimistically solve the k-median problem first, getting centers $c_1, \ldots, c_k$, and then find the mapping $h \in H$ of lowest loss wrt these centers $\sum_{I_j \in S} d(c_{h(I_j)}, s_j)$, then this will be within a factor of 3 of the best pair (h^*, c^*).



Summary: If we're willing to lose some constant factors, can first solve (or apx solve) a classic optimization problem, and then reduce to a standard "ERM with a cost matrix" problem.

Online adaptive initialization

What if our problem instances are not iid? Can we compete with best adaptive initialization (or set of k initializations)?

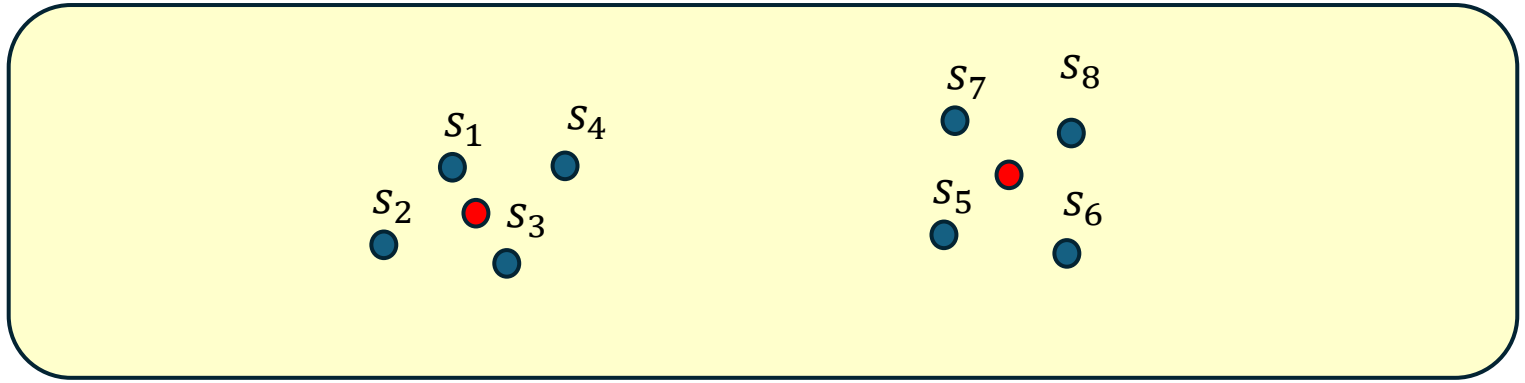


For this to make sense, we need to charge the "offline competitor" for movement. Otherwise "correctly guessing the solution to the next problem" has 0 cost.

E.g., for $k = 1$, comparing to optimal $d(c_1, s_1) + d(c_1, c_2) + d(c_2, s_2) + d(c_2, c_3) + d(c_3, s_3) + \cdots$

Online adaptive initialization

What if our problem instances are not iid? Can we compete with best adaptive initialization (or set of k initializations)?

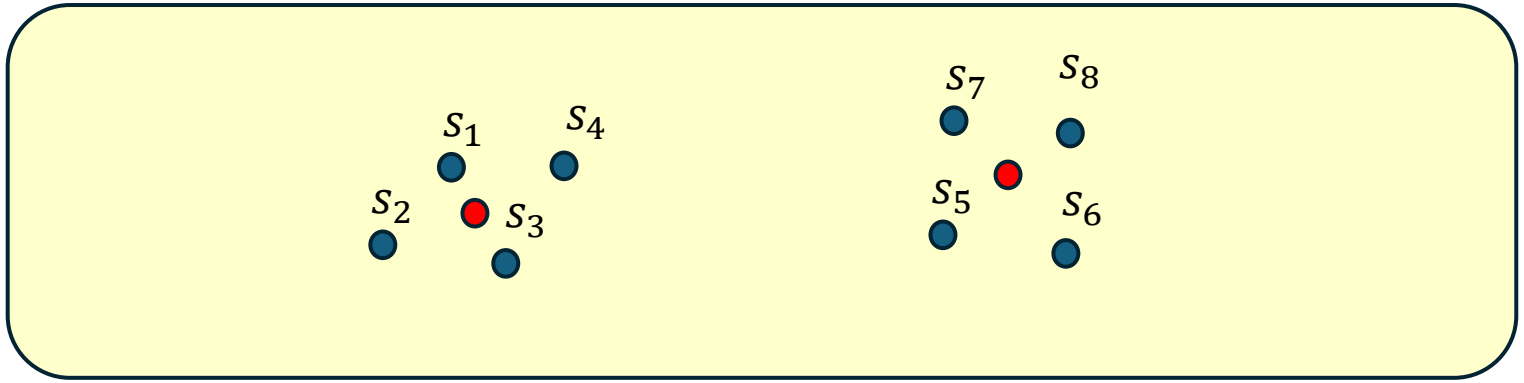


For this to make sense, we need to charge the "offline competitor" for movement. Otherwise "correctly guessing the solution to the next problem" has 0 cost.

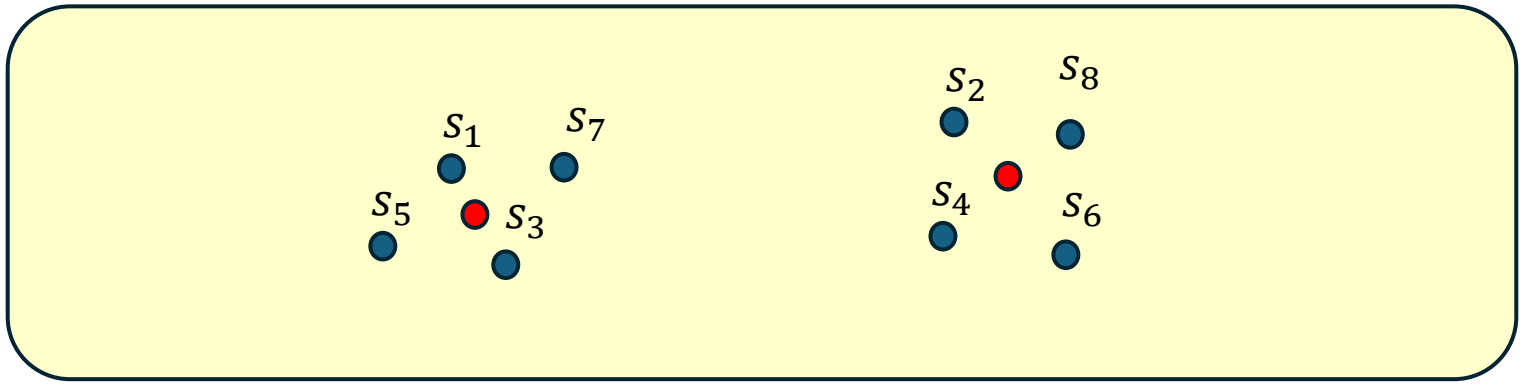
We're going to consider the case where there is no feature info (but would be interesting to extend).

Some examples

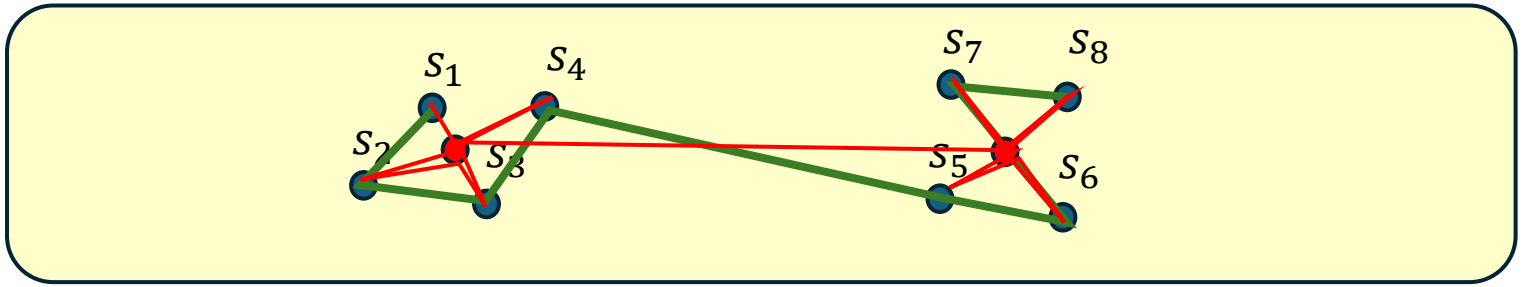
This scenario is not bad for $k = 1$:



This scenario is bad for $k = 1$ but not bad for $k = 2$:



Claim: Simple factor-2 approximation for $k = 1$



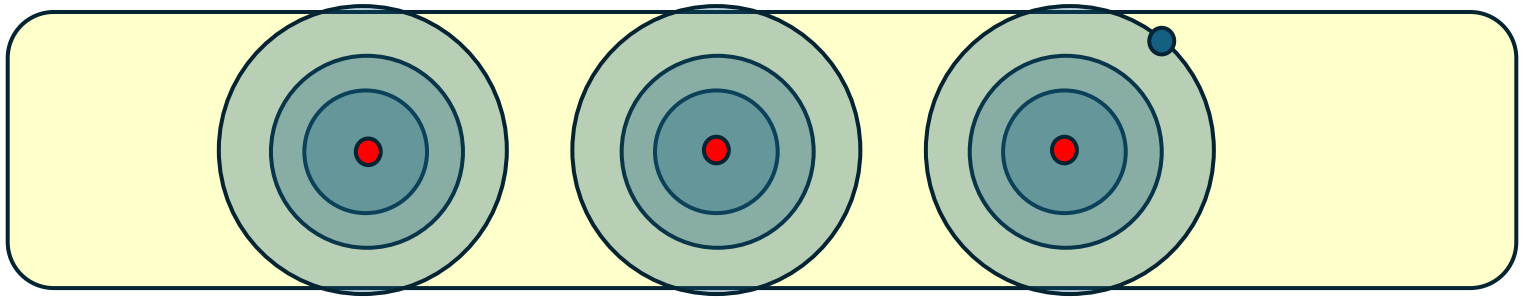
Algorithm: just use solution s_j as warm start for problem instance I_{j+1}.

Analysis: compare to best sequence $c_1, c_2, \ldots, c_T$.

Say s_1 is generic start, I_2 is first real instance

- Our cost: $d(s_1, s_2) + d(s_2, s_3) + d(s_3, s_4) + d(s_4, s_5) \ldots$
- $\leq d(s_1, c_1) + d(c_1, c_2) + d(c_2, s_2) + d(s_2, c_2) + d(c_2, c_3) + d(c_3, s_3) + d(s_3, c_3) + \cdots$
- $\leq 2 \cdot [d(s_1, c_1) + d(c_1, c_2) + d(c_2, s_2) + d(c_2, c_3) + d(c_3, s_3) + \cdots] = 2 \cdot$(cost of opt seq)

What about approximating the best set of k (adaptive) initializations?



Idea #1: If we knew that each new problem-instance was likely to have a solution near one of k possible warm-starts $c_1, \ldots, c_k$, then we could run k copies of the algo in parallel.

- Would be within a factor k of the best of these k warm-starts.

- Of course, we don't know the locations of the $c_1, \ldots, c_k$ we are competing with, and they're changing over time.

What about approximating the best set of k (adaptive) initializations?

Idea #2: With only constant-factor loss, we can assume the warm-starts we are competing with are solutions of previously-seen instances (by triangle inequality similar to $k = 1$ case)

- Of course, we don't know which ones are the currently-active ones…

- This is a lot like the classic "k-server problem" except:
  - (a) it's harder because we don't know the location of the new point we're trying to "serve" (we can only grow spheres), but
  - (b) It's also easier because we aren't limited to using exactly k warm-starts ourselves.
  - (c) And it's different in that we'd ideally like to get a simultaneous guarantee for all k: I.e.,

$$\text{our cost} \leq \min_k [g(k) \cdot (\text{cost of best set of } k \text{ adaptive initializations})]$$

What about approximating the best set of k (adaptive) initializations?

Idea #3: Let's start at all previously-seen solutions, and run in parallel from them.

- Wait, that's not good. Our performance would degrade (apx ratio would increase) with # instances seen.

Idea #4: What if we favor more recent solutions? On instance t, spend $\frac{1}{f(i)}$ fraction of time using solution s_{t-i} as warm start, for $f(i) = O(i^2)$ or $f(i) = O(i \log^2 i)$.

- Rationale: The optimal trajectory must come from a fairly recent solution most of the time, since the solution used gets updated to the current solution.

What about approximating the best set of k (adaptive) initializations?

Idea #3: Let's start at all previously-seen solutions, and run in parallel from them.

- Wait, that's not good. Our performance would degrade (apx ratio would increase) with # instances seen.

Idea #4: What if we favor more recent solutions? On instance t, spend $\frac{1}{f(i)}$ fraction of time using solution s_{t-i} as warm start, for $f(i) = O(i^2)$ or $f(i) = O(i \log^2 i)$.

- Rationale: The optimal trajectory must come from a fairly recent solution most of the time, since the solution used gets updated to the current solution.

- Unfortunately, that's still not quite enough to get a poly(k) bound.

What about approximating the best set of k (adaptive) initializations?

Idea #5: On instance t, spend $\frac{1}{f(i)}$ fraction of time using solution s_{t-i} as warm start, but also once one ball engulfs another, kill the process for the one that got subsumed.

- This then gives a poly(k) approximation (competitive ratio).

What about approximating the best set of k (adaptive) initializations?

Final result:

- Gives an $\tilde{O}(k^4)$ or $\tilde{O}(k^3)$ approximation depending on exact model for counting costs.
- Can improve to $\tilde{O}(k^2)$ if we only care about a specific k, using a different algorithm.
- Best lower bound we know is $\Omega(k)$.
- To go below this would need to use feature info.

More broadly

- This was just one kind of Data-Driven Algorithm Design (using past instances to determine warm-starts).

- Data-Driven Algorithm Design more generally considers a family of algorithms defined by various parameters (in this case, the family is defined by the initialization).

- Can the approximation algorithms perspective give us more provably-efficient ways to ***find*** near-optimal algorithms in the family (possibly given access to natural ERM oracles), or do nearly as well as the best algorithm in the family?