Learning the best algorithm for max-cut, clustering and other partitioning problems

Vaishnavh Nagarajan

Joint work with

Maria-Florina Balcan, Ellen Vitercik and Colin White

Learning the "best" algorithm

Real-world **applications:** NP hard problems such as clustering, max-cut

→

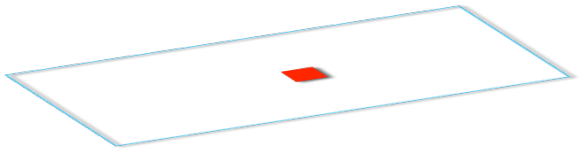
Algorithms, Heuristics

$\{alg_1, alg_2, alg_3, alg_4 ....\}$

→

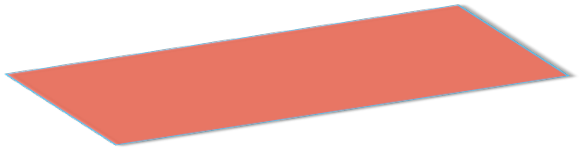
Select "best" alg according to approximation factor, running time etc.,

Worst-case

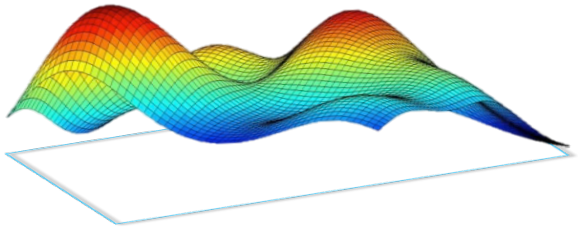


A worst case problem in the universe of problem instances

Average case



A uniform distribution over all problem instances

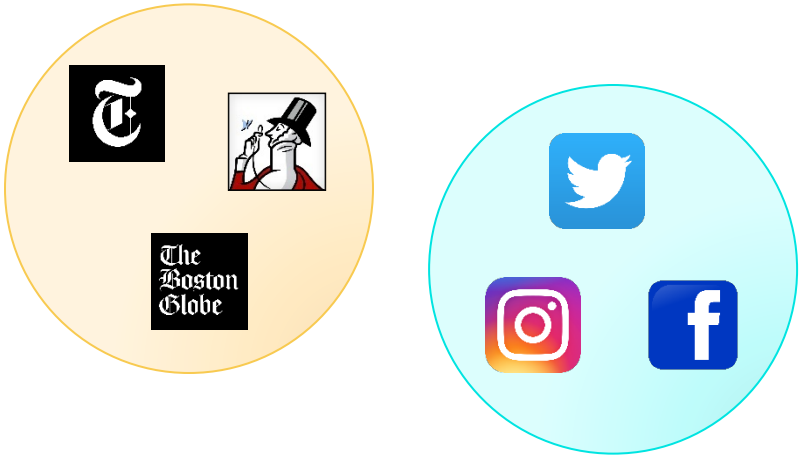


An **unknown application-specific** distribution over problem instances

Efficient approaches with theoretical guarantees to learn the "best" algorithm from a rich family of algorithms.

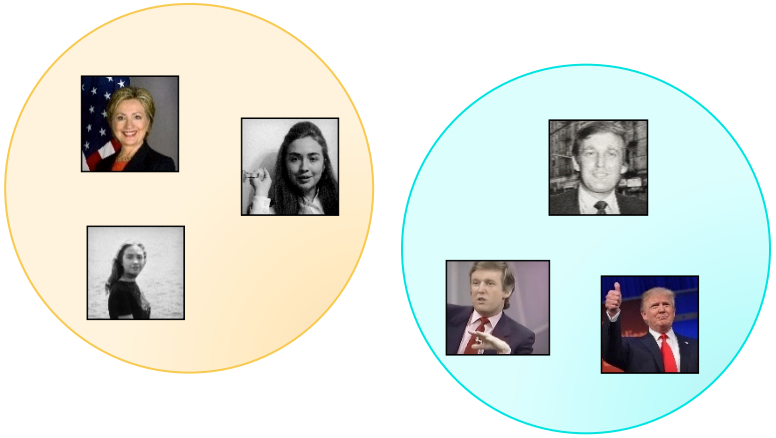
Example: Clustering

Clustering webpages by topic



A problem instance

Clustering images by subject

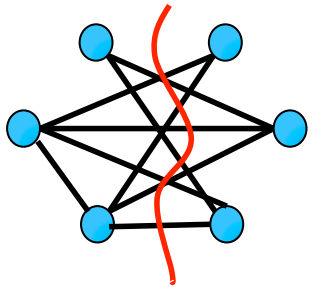


A problem instance

- A problem faced in many different domains.
- Many approximation and heuristic algorithms
- No technique is best across all applications.

Our goal: choose the best algorithm for a given application domain.

Example: Integer Quadratic Programming



- Abstract problem with diverse applications:
  - finding the max-cut in a graph
  - SAT
  - correlation clustering

- Relax to a semidefinite program: many ways to “round”

Our goal: choose the best algorithm for a given application domain.

Background Work

- Long history of application-specific algorithm selection in artificial intelligence research.
  - automated algorithm configuration
  - algorithm portfolio selection
- 2016: New learning-theoretic model by Gupta and Roughgarden

Very little theoretical analysis of application-specific algorithm selection.

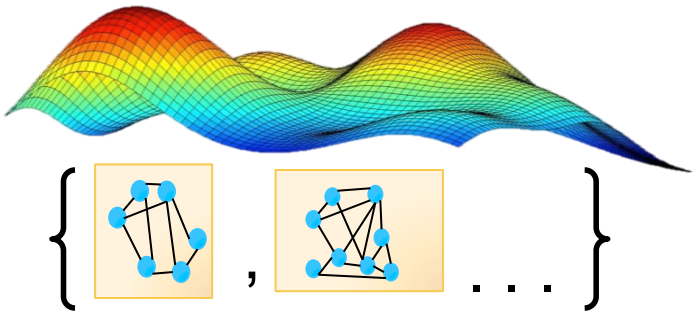
Outline

- Introduction
- **Goal and approach**
- Clustering
- Max-cut

Algorithm Selection Model

1. Fix a family of algorithms

$\mathcal{A}=\{alg_1, alg_2, alg_3, alg_4 ....\}$



3. Unknown application-specific distribution over set of all problem instances

2. Fix a performance metric

$\text{COST}(alg, I) =$ 

Ideal goal: Pick alg from $\mathcal{A}$ with best expected performance.

$$\mathbf{E}[\text{COST}(alg, I)]$$

But we don't know the distribution over problem instances!

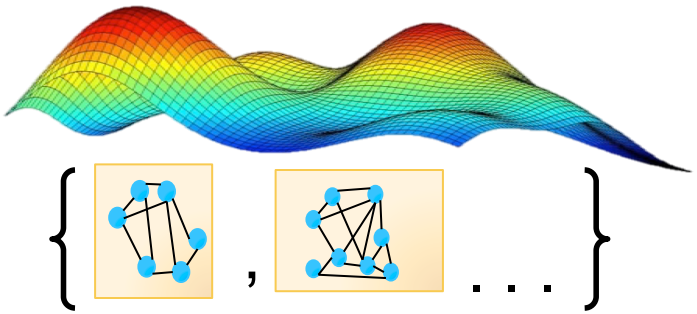
Algorithm Selection Model

1. Fix a family of algorithms

$\mathcal{A}=\{alg_1, alg_2, alg_3, alg_4 ....\}$

2. Fix a performance metric

COST(alg , I) = 



problem instance

another problem instance

Sample some training problem instances

$S = \{I_1, I_2, I_3\}$

Pick empirically best alg from $\mathcal{A}$

We hope alg is near-optimal in expectation over unknown distribution

3. Unknown application-specific distribution over set of all problem instances

Question 1: How do we ensure near-optimality of empirically best algorithm? That is, how many samples S are needed?

Question 2: How do we find the empirically best algorithm from $\mathcal{A}$ in polytime?

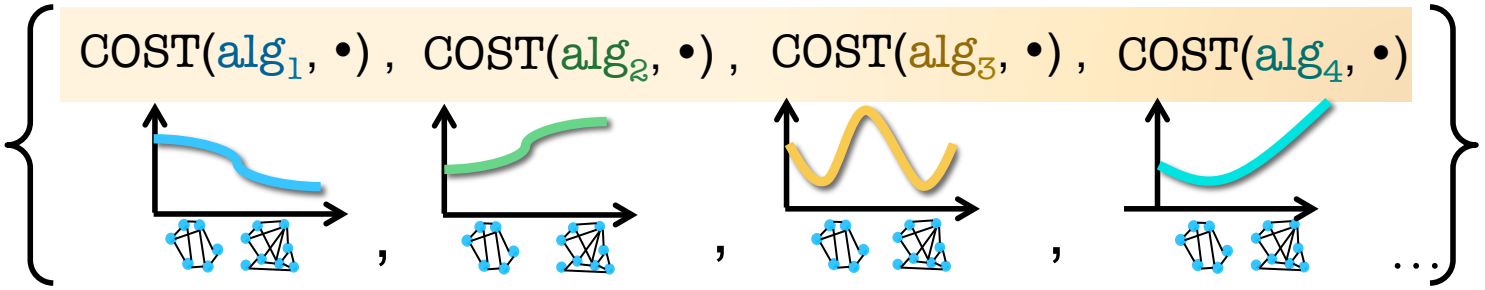
Algorithm Selection Model

1. Fix a family of algorithms

   $\mathcal{A}=\{\text{alg}_1, \text{alg}_2, \text{alg}_3, \text{alg}_4 \ldots.\}$

2. Fix a performance metric

   $\text{COST}(\text{alg}, I) =$ 

$\text{COST}_{\mathcal{A}} =$ 

x-axis: Problem instance space
y-axis: COST of an algorithm

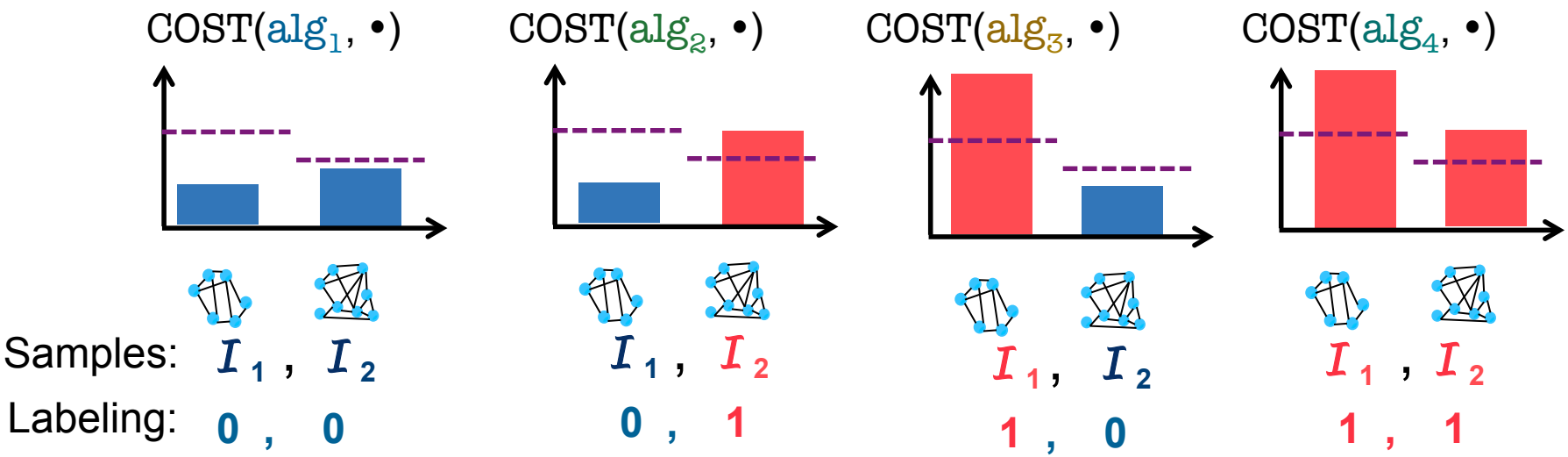
Sample complexity proportional to "intrinsic complexity" of $\text{COST}_{\mathcal{A}}$

Question 1: How do we ensure near-optimality of empirically best algorithm? That is, how many samples S are needed?

Answer: $$|\mathcal{S}| = \tilde{\mathcal{O}}\left(\frac{\text{Pdim}(\text{COST}_{\mathcal{A}})}{\epsilon^2}\right)$$

Pseudodimension of $\mathrm{COST}_{\mathcal{A}}$

$$\mathrm{COST}_{\mathcal{A}} = \{\mathrm{COST}(\mathrm{alg}, \bullet) \mid \mathrm{alg} \text{ in } \mathcal{A}\}$$



Size of the largest set of problem instance samples such that there are $2^{|S|}$ algorithms in $\mathcal{A}$ each inducing a different COST "labeling" of samples S w.r.t some thresholds r_i.

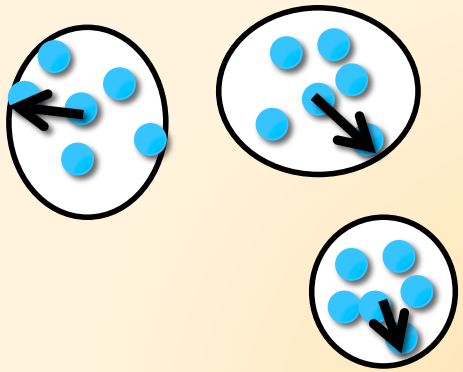
Outline

- Introduction
- Our goal and approach
- **Clustering**
- Max-cut

Clustering

A problem instance

- a set $\mathcal{X}$ of n points
- pairwise distances between them



any *k*-clustering algorithm

A partition of $\mathcal{X}$ into *k* sets

minimize COST

e.g., maximum radius of clusters, average radius of clusters.

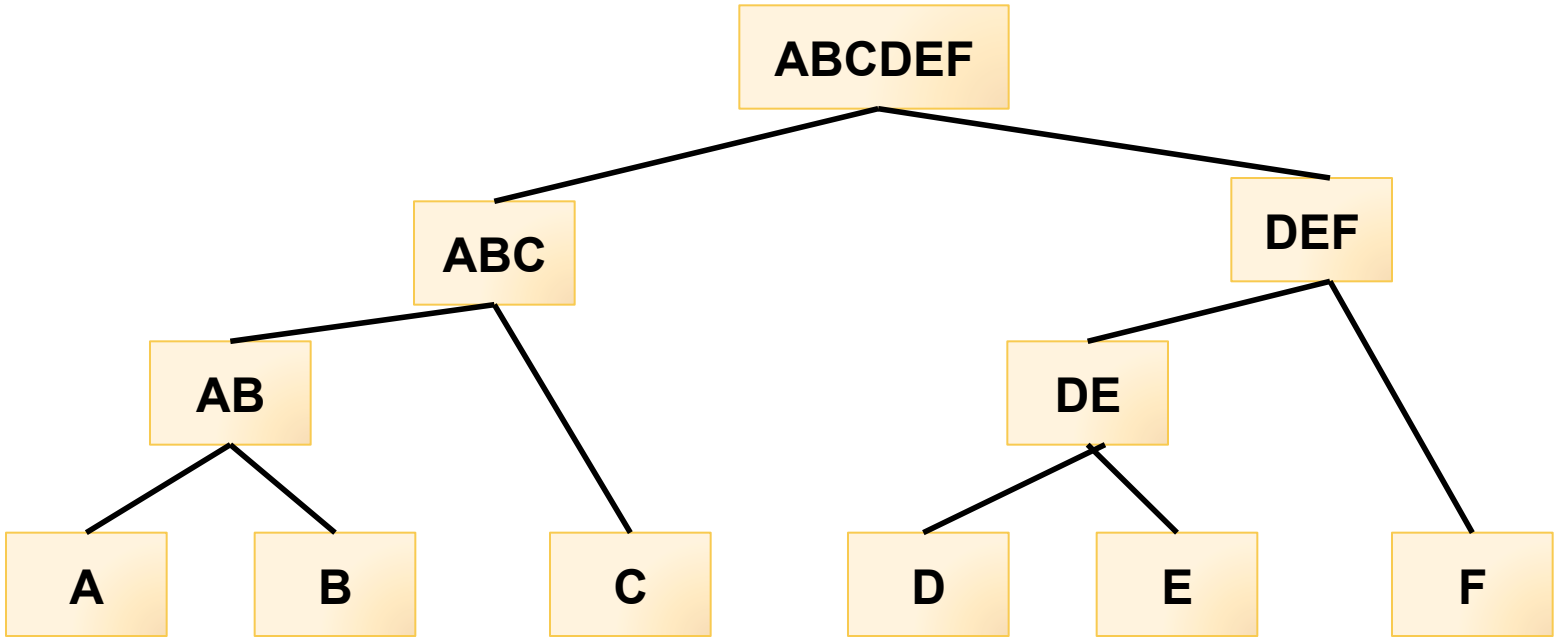
We consider an arbitrary cost.

A rich class of clustering algorithms

A problem instance:
set of n points
and
pairwise distances
between them

→

Build a cluster
tree bottom-up:
iteratively merge
two "closest"
clusters



A rich class of clustering algorithms

A problem instance:
set of n points
and
pairwise distances
between them

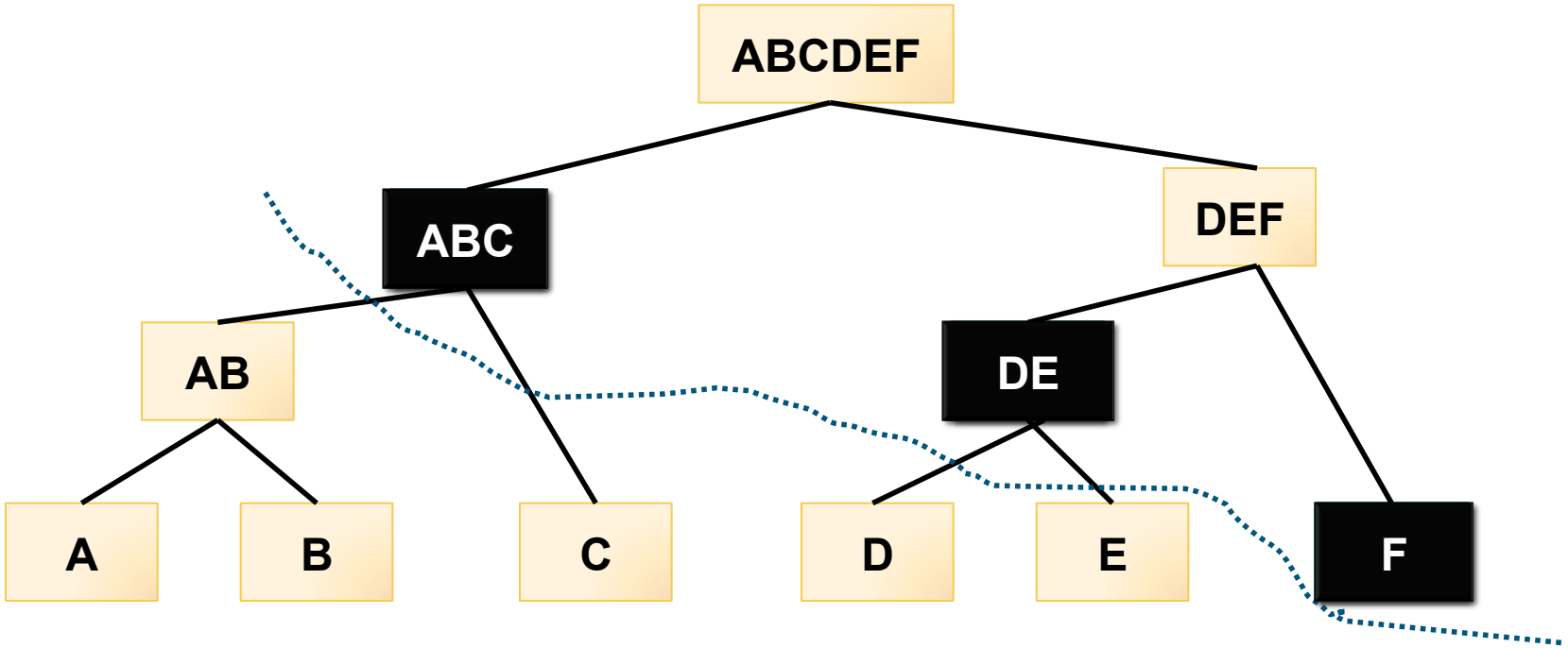
→

Build a cluster tree bottom-up: iteratively merge two "closest" clusters

→

Perform dynamic programming on the tree to identify best of all prunings.

→ *k*-Clustering



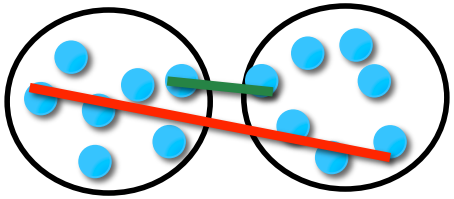
A rich class of clustering algorithms

A problem instance:
set of n points
and
pairwise distances
between them

Single linkage
:
α-linkage
:
Complete linkage

Perform dynamic programming on the tree to identify best of all prunings.

→ *k*-Clustering



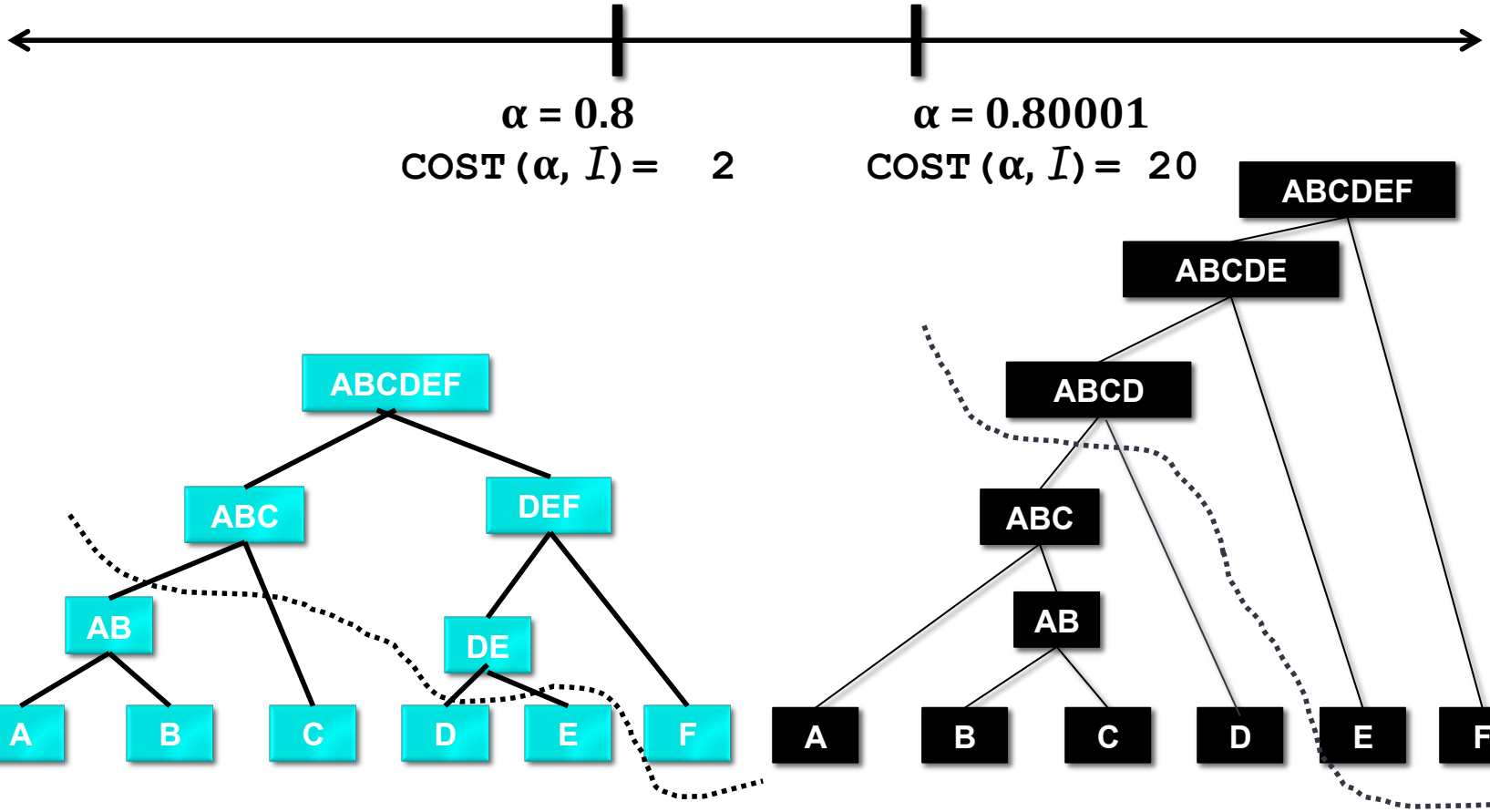
$$\alpha \cdot \min_{p \in N_i, q \in N_j} d(p,q) + (1-\alpha) \cdot \max_{p \in N_i, q \in N_j} d(p,q)$$

Linear interpolation between single- and complete-linkage both of which enjoy strong worst-case guarantees in various settings

**Each α is a different path/algorithm:
which is the best for an application?**

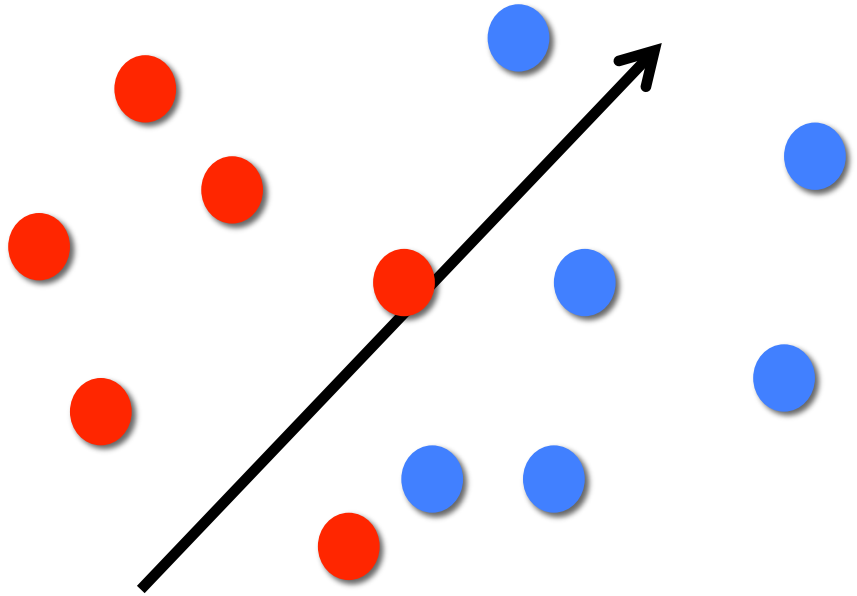
Key Challenge

Nearby values of α could result in very different cluster trees and costs.

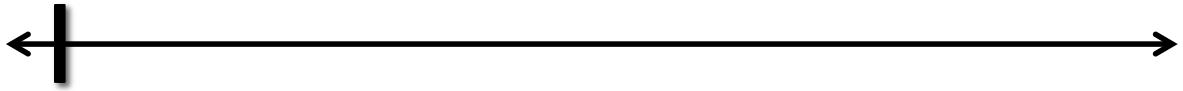


Key Challenge

Function:



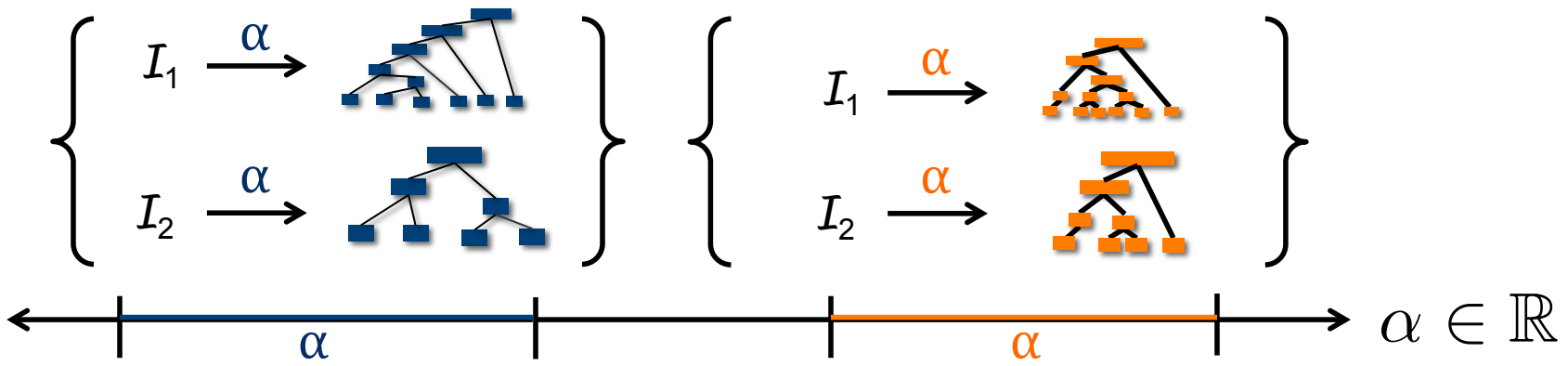
Parameter :



Changing parameters of a function in machine learning: smooth change in behavior.

Key Idea

For a given set S of problem instances (each of at most n points), we can break the line into $O(|S|n^8)$ of intervals:
➔ α values from the same interval result in a fixed set of trees



Key Idea

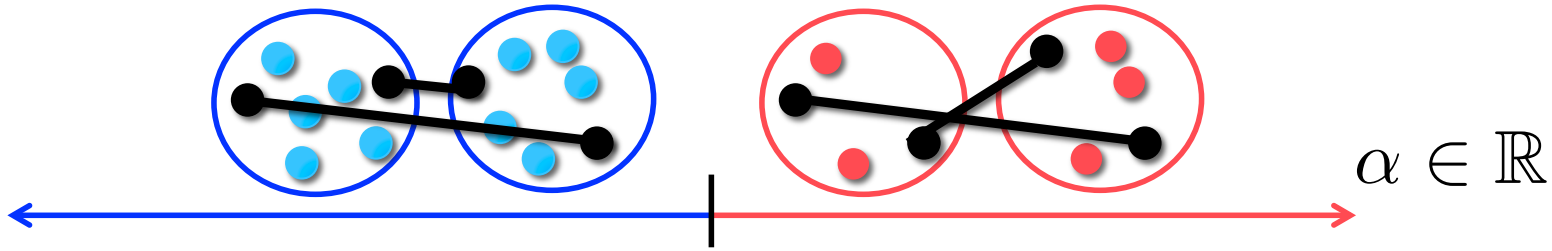
For a given set S of problem instances (each of at most n points), we can break the line into $O(|S|n^8)$ of intervals:
➔ α values from the same interval result in a fixed set of trees

Proof Idea

Decision to merge cluster nodes **A, B** before **P,Q** flips at only one **α** when:

$$d(A,B) = d(P,Q)$$

$$\alpha\, d(a_{min},b_{min}) + (1-\alpha)\, d(a_{max},b_{max}) = \alpha\, d(p_{min},q_{min}) + (1-\alpha)\, d(p_{max},q_{max})$$



For an I, only $O(n^8)$ such comparisons → partitions **α** line into $O(n^8)$ intervals

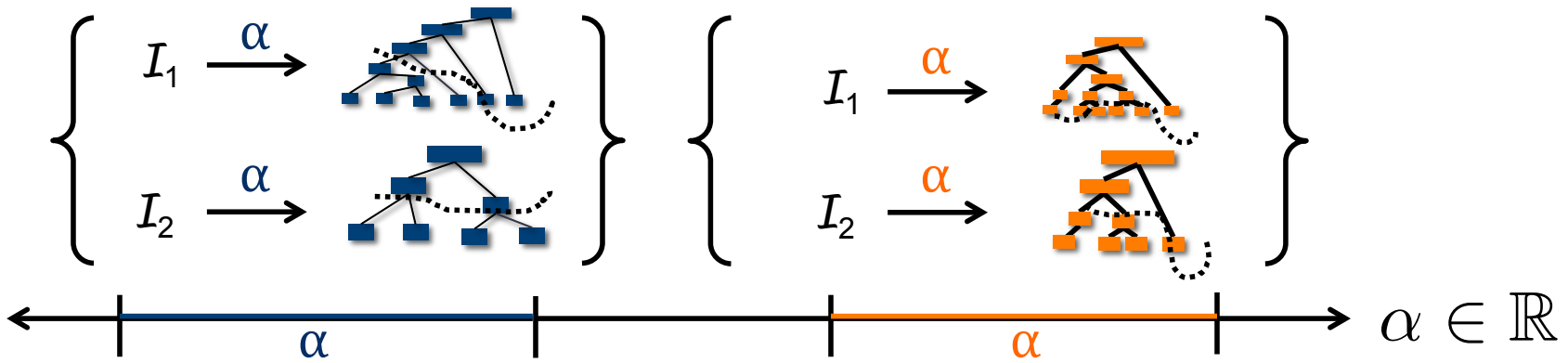
Key Idea

For a given set S of problem instances (each of at most n points), we can break the line into $O(|S|n^8)$ of intervals:

➔ α values from the same interval result in a fixed set of trees

➔fixed set of pruned solutions

➔fixed set of costs



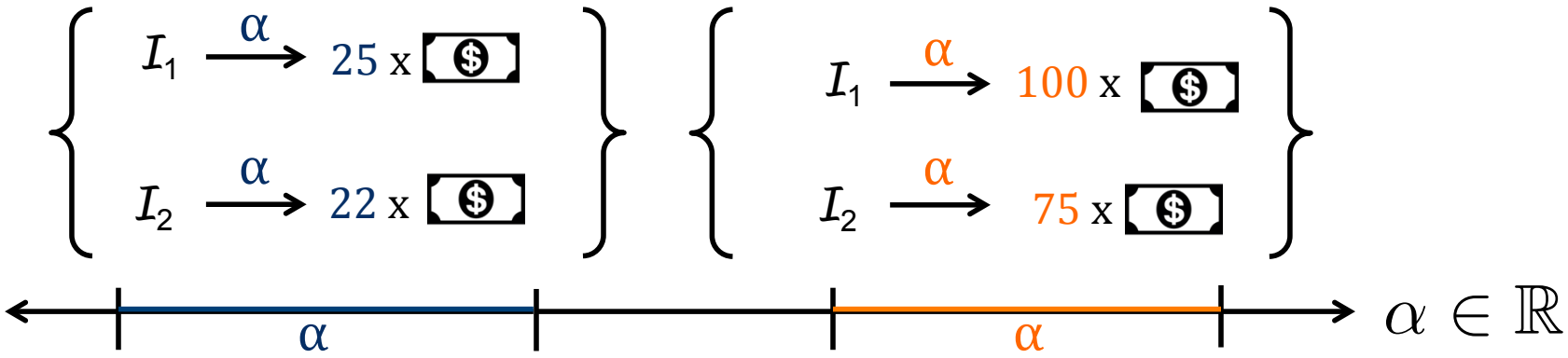
Pseudodimension bounds

Theorem: For any abstract COST, for the class of α-linkage rule based clustering algorithms: $\mathbf{Pdim}(\mathrm{COST}_{\mathcal{A}}) = \Theta(\log n)$

Upper bound: If $|S| = \mathbf{Pdim}(\mathrm{COST}_{\mathcal{A}})$, then:

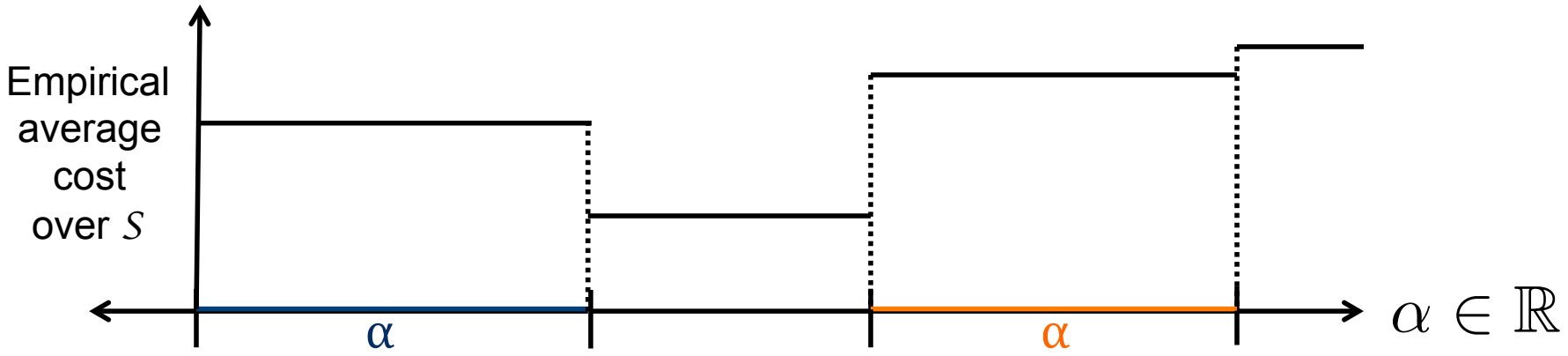
$$2^{|S|} < O(|S|n^8)$$

Lower bound: Carefully **construct** $\Omega(\log n)$ clustering instances



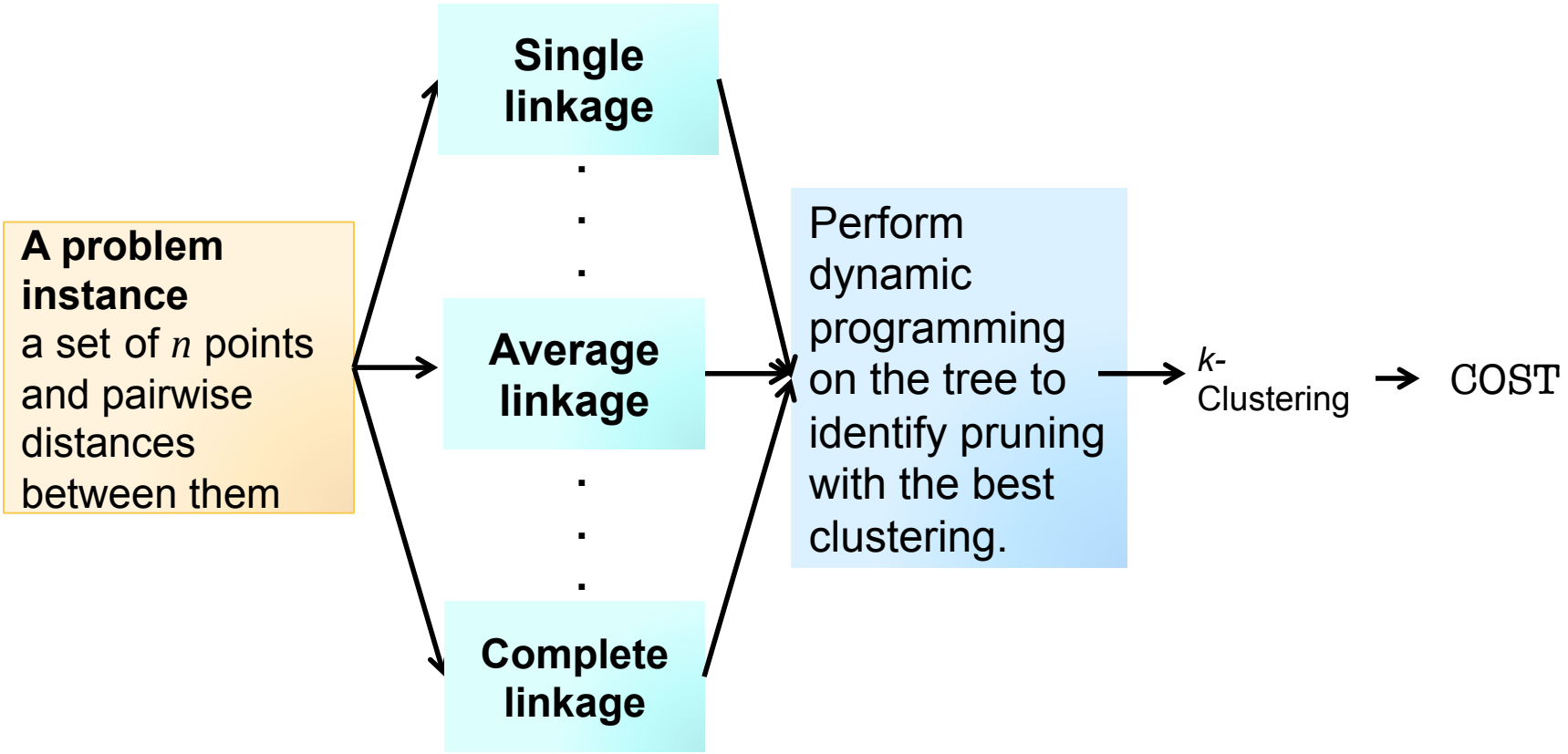
Computationally Efficient Algorithm Selection

- Draw $O(\log n)$ samples S
- Solve for the $O(|S|n^8)$ intervals.
- Run the algorithm for **only one α per interval.**
- Find the empirically best interval.



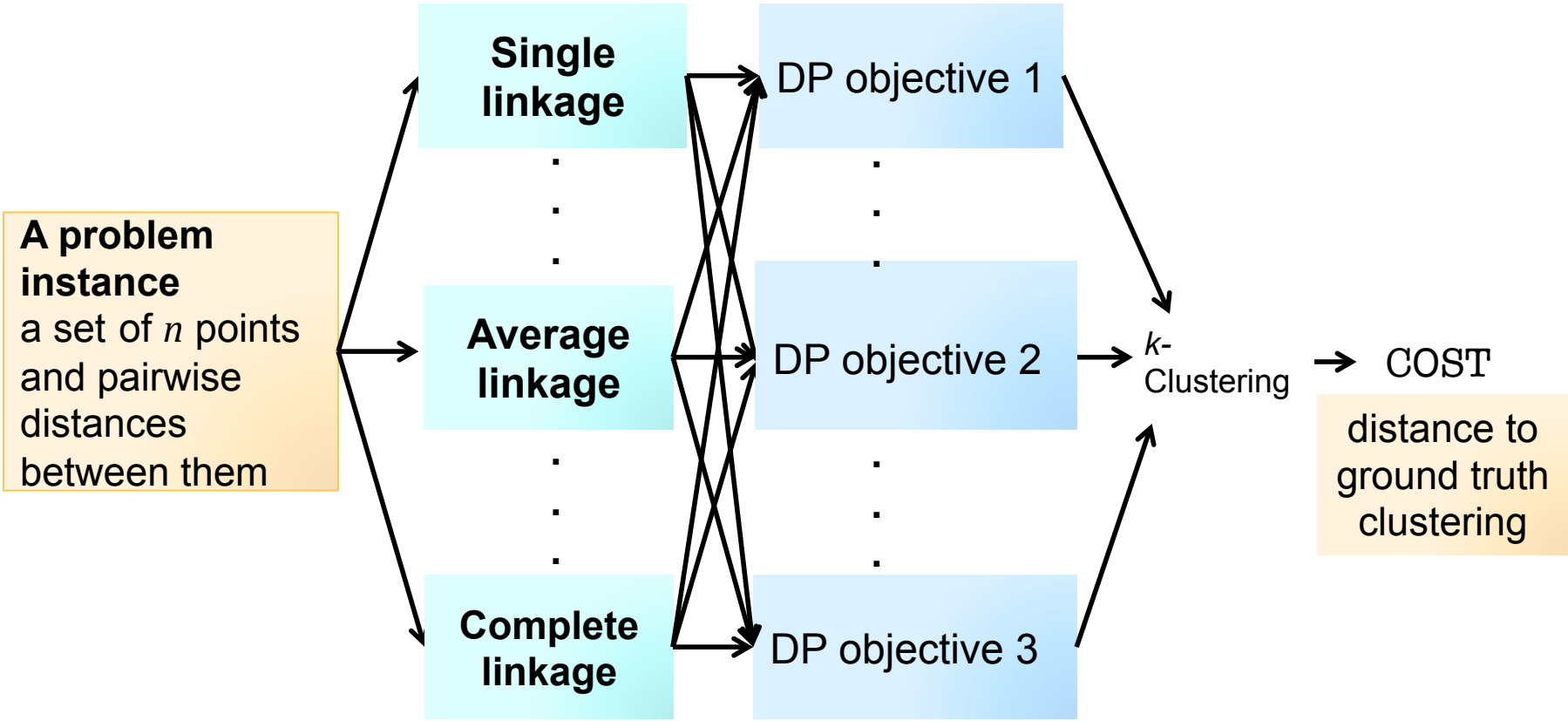
More general results

α-linkage corresponding to a non-linear interpolation:
$\mathrm{Pdim}(\mathrm{COST}_{\mathcal{A}}) = \Theta(n)$



More general results

Another layer of richness in the dynamic programming step.

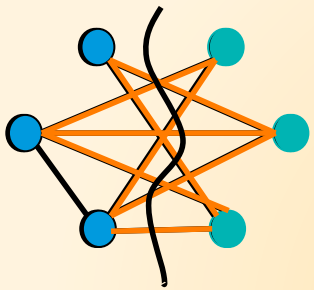


Outline

- Introduction
- Our goal and approach
- Clustering
- **Max-cut**

Maximum Cut

A problem instance
A graph G of n vertices V with edge weights w_{ij}



any max-cut algorithm →

A partition of V into two sets:

$$v_i \in \{-1, +1\}$$

↓ maximize a quadratic OBJ

total edge weight connecting the two sets

$$\text{OBJ} = \sum w_{ij} \left(\frac{1 - v_i v_j}{2} \right)$$

Our results apply to more general integer quadratic programs besides max-cut.

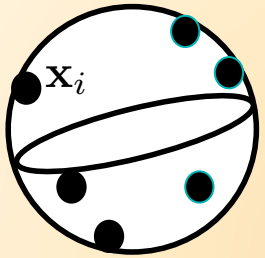
Standard approximation approach

Max cut OBJ:
Integer-quadratic programming formulation

$v_i \in \{-1, +1\}$

2. Semidefinite programming (SDP) relaxation

$\mathbf{x}_i \in \mathbb{R}^n$



3. Round SDP embedding

3. Round SDP embedding

3. Round SDP embedding

A graph cut solution

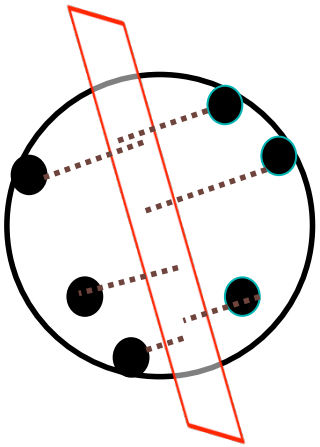
$v_i \in \{-1, +1\}$

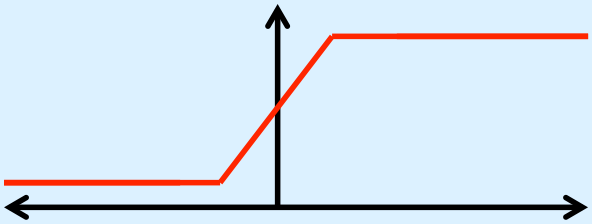
Each rounding → different algorithm.
Best algorithm for an application?

A family of SDP rounding techniques

Find the SDP embedding → Choose a random hyperplane **h** → (Randomized Rounding) → **Output:** For each vertex, assign +1/-1 according to some probability

[FL06]





$$\Phi_s(x) = \begin{cases} -1 & \text{if } y < -s \\ y/s & \text{if } -s <= y <= s \\ +1 & \text{if } y > s \end{cases}$$

Rounding function

A family of s-linear rounding functions

Each value of s → expected OBJ

Key Idea

- Want to bound Pdim($\mathrm{OBJ}_{\mathcal{A}}$):

$\mathrm{OBJ}_{\mathcal{A}}$ = {$\mathrm{OBJ}(\mathbf{s}, \bullet)$: problem instance space → [0, 1] | $\mathbf{s} > 0$ }

where: $\mathrm{OBJ}(\mathbf{s}, I) = \mathbb{E}_{\text{hyperplane}}[\mathrm{OBJ}^*(\mathbf{s}, I, \text{hyperplane})]$

has a closed form expression

- Consider:

$\mathrm{OBJ}^*_{\mathcal{A}}$ = {$\mathrm{OBJ}^*(\mathbf{s}, \bullet, \bullet)$: problem instance space x $\mathbf{R}^n$ → [0, 1] | $\mathbf{s} > 0$ }

- Pdim($\mathrm{OBJ}_{\mathcal{A}}$) <= Pdim($\mathrm{OBJ}^*_{\mathcal{A}}$) → bound Pdim($\mathrm{OBJ}^*_{\mathcal{A}}$) for sample complexity.

Pseudodimension bounds

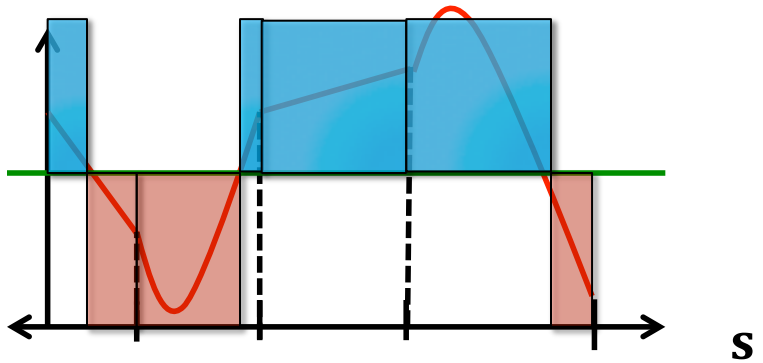
Theorem: For the class of s-linear based randomized rounding approaches $\mathcal{A}$:

$$\text{Pdim}(\text{OBJ}^*_{\mathcal{A}}) = \Theta(\log n)$$

Upper bound follows from:

$$\text{OBJ}^*(\mathbf{s}, \mathcal{I}, \mathbf{h}) = \sum_{i,j} \Phi_s(\mathbf{x}_i \bullet \mathbf{h})\, \Phi_s(\mathbf{x}_j \bullet \mathbf{h})$$

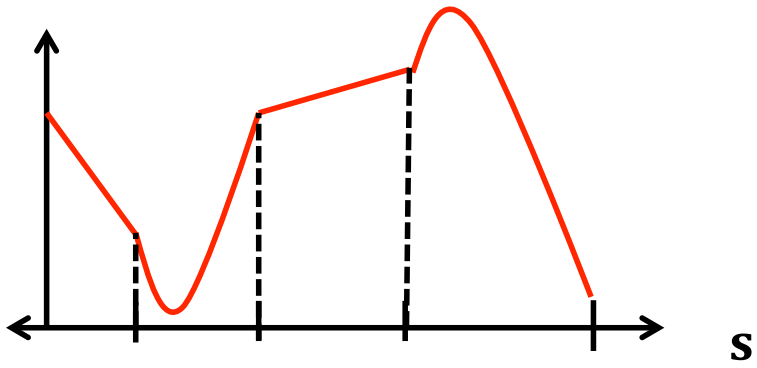
- $\text{OBJ}^*(s, \mathcal{I}, h)$ at most n+1 quadratic/linear pieces
- A threshold → $O(n)$ intervals
- $2^{|S|} < O(|S|n)$



Lower bound: Carefully construct $\Omega(\log n)$ (max-cut problem instance, hyperplane) pairs.

Computationally Efficient Algorithm Selection

- Draw $O(\log n)$ problem instance samples S.
- For each, draw a random hyperplane.
- Empirical average of OBJ*(s, $\mathcal{I}_i$, h_i) → $O(|S|n)$ pieces.

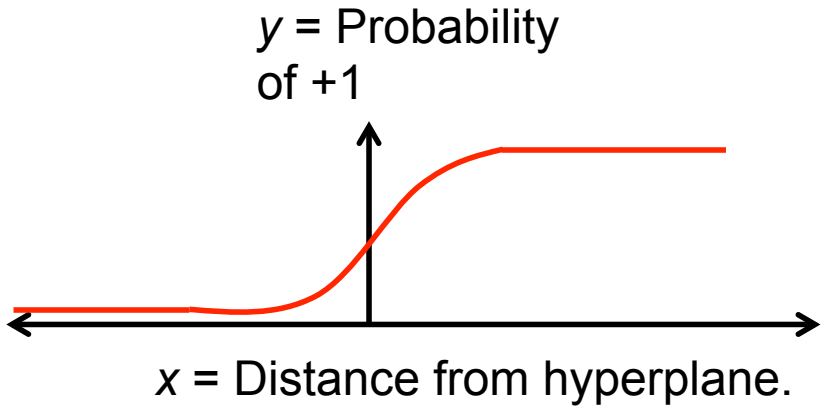


- Find best s efficiently.

More general result

Theorem: For any class $\mathcal{A}$ based on sigmoid-like parametrized rounding functions:

$$\text{Pdim}(\text{OBJ}_{\mathcal{A}}) = \Theta(\log n)$$



Sigmoid-like rounding function

Summary

- Design and analysis of algorithms and learning theory.
- Multi-stage, randomized procedures.
- Tight bounds on the intrinsic complexity
- Surprisingly, superconstant bounds despite only $O(1)$ parameters.

Future directions:

- Generalize analysis to other rounding functions and other problems that can be relaxed to SDP?
- Algorithm families with too slow algorithms?