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, \ldots\}

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

Efficient approaches with theoretical guarantees to learn the “best” algorithm from a rich family of algorithms.
Example: Clustering

Clustering webpages by topic

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

2. Fix a performance metric
\[ \text{COST}(\text{alg}, I) = \] $\$ \\
Ideal goal: Pick \text{alg} from \mathcal{A} with best expected performance.

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

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

But we don’t know the distribution over problem instances!
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
   \[ \text{COST}(\text{alg}, I) = \] $\hspace{1cm}$

Sample some training problem instances
   \( S = \{I_1, I_2, I_3\} \)

Pick empirically best \( \text{alg} \) from \( \mathcal{A} \)

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

We hope \( \text{alg} \) is near-optimal in expectation over unknown distribution

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

\[
\text{COST}_\mathcal{A} = \left\{ \text{COST(\text{alg}_1, \cdot)}, \text{COST(\text{alg}_2, \cdot)}, \text{COST(\text{alg}_3, \cdot)}, \text{COST(\text{alg}_4, \cdot)}, \ldots \right\}
\]

- **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**: 
\[
|S| = \tilde{O} \left( \frac{\text{Pdim} (\text{COST}_\mathcal{A})}{\epsilon^2} \right)
\]
Pseudodimension of \( \text{COST}_\mathcal{A} \)

\[
\text{COST}_\mathcal{A} = \{ \text{COST}(\text{alg}, \cdot) \mid \text{alg in } \mathcal{A} \}
\]

**Size of the largest set of problem instance samples**

such that there are \( 2^{|S|} \) algorithms in \( \mathcal{A} \) each inducing a different \( \text{COST} \) “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

A partition of $\mathcal{X}$ into $k$ sets
any $k$-clustering algorithm
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
: 
\[ \alpha \cdot \min_{p \in N_i, q \in N_j} d(p, q) + (1 - \alpha) \cdot \max_{p \in N_i, q \in N_j} d(p, q) \]

$\alpha$-linkage
: 

Complete linkage

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

$k$-Clustering

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

Each $\alpha$ is a different path/algorithm: which is the best for an application?
Key Challenge

Nearby values of \( \alpha \) could result in very different cluster trees and costs.

\[
\begin{align*}
\alpha &= 0.8 \\
\text{COST}(\alpha, I) &= 2 \\
\alpha &= 0.80001 \\
\text{COST}(\alpha, I) &= 20
\end{align*}
\]
Key Challenge

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:

$\Rightarrow \alpha$ values from the same interval result in a fixed set of trees

\[ \begin{align*}
    I_1 &\xrightarrow{\alpha} \ \text{trees}\ \\
    I_2 &\xrightarrow{\alpha} \\
\end{align*} \]
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:

$\Rightarrow \alpha$ 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 $\alpha$ when:

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

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

For an $I$, only $O(n^8)$ such comparisons $\Rightarrow$ partitions $\alpha$ 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:

- $\alpha$ values from the same interval result in a fixed set of trees
- fixed set of pruned solutions
- fixed set of costs

\[
\begin{align*}
\{ I_1 \} & \xrightarrow{\alpha} \{ \text{trees} \} \\
\{ I_2 \} & \xrightarrow{\alpha} \{ \text{trees} \}
\end{align*}
\]

\[
\begin{align*}
\{ I_1 \} & \xrightarrow{\alpha} \{ \text{trees} \} \\
\{ I_2 \} & \xrightarrow{\alpha} \{ \text{trees} \}
\end{align*}
\]
Pseudodimension bounds

**Theorem**: For any abstract COST, for the class of $\alpha$-linkage rule based clustering algorithms: $\text{Pdim}(\text{COST}_A) = \Theta(\log n)$

**Upper bound**: If $|S| = \text{Pdim}(\text{COST}_A)$, then:

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

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

\[
\begin{align*}
I_1 &\xrightarrow{\alpha} 25 \times \$ \\
I_2 &\xrightarrow{\alpha} 22 \times \$
\end{align*}
\] \begin{align*}
I_1 &\xrightarrow{\alpha} 100 \times \$ \\
I_2 &\xrightarrow{\alpha} 75 \times \$
\end{align*}

$\alpha \in \mathbb{R}$
Computationally Efficient Algorithm Selection

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

\( \alpha \)-linkage corresponding to a non-linear interpolation:
\[ P_{\text{dim}}(COST_{\mathcal{A}}) = \Theta(n) \]

A problem instance
a set of \( n \) points
and pairwise distances
between them

Single linkage
\[ \cdots \]

Average linkage
\[ \cdots \]

Complete linkage
\[ \cdots \]

Perform dynamic programming on the tree to identify pruning with the best clustering.

\( k \)-Clustering \( \rightarrow \) COST
More general results

Another layer of richness in the dynamic programming step.

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

DP objective 1
DP objective 2
DP objective 3

Cost:
Distance to ground truth clustering
Outline

- Introduction
- Our goal and approach
- Clustering
- Max-cut
Maximum Cut

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

A max-cut algorithm
Any partition of $V$ into two sets:
$v_i \in \{-1, +1\}$

maximize a quadratic $\text{OBJ}$
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
\( x_i \in \mathbb{R}^n \)

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

[FL06]

Find the SDP embedding → Choose a random hyperplane $h$ → Randomized Rounding

Output: For each vertex, assign $+1/-1$ according to some probability

$y = \text{Probability of } +1 \text{ assignment}$

$x = \text{Distance from hyperplane}$

Rounding function

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

Each value of $s \rightarrow$ expected OBJ
Key Idea

- Want to bound $\text{Pdim}(\text{OBJ}_A)$:

$\text{OBJ}_A = \{\text{OBJ}(s, \bullet) : \text{problem instance space} \to [0, 1] \mid s > 0 \}$

where: $\text{OBJ}(s, I) = E_{\text{hyperplane}}[\text{OBJ}^*(s, I, \text{hyperplane})]$ has a closed form expression

- Consider:

$\text{OBJ}^*_A = \{\text{OBJ}^*(s, \bullet, \bullet) : \text{problem instance space} \times \mathbb{R}^n \to [0, 1] \mid s > 0 \}$

- $\text{Pdim}(\text{OBJ}_A) \leq \text{Pdim}(\text{OBJ}^*_A) \to$ bound $\text{Pdim}(\text{OBJ}^*_A)$ 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:

$$\text{OBJ}^*(s, I, h) = \sum_{i,j} \Phi_s(x_i \cdot h) \Phi_s(x_j \cdot h)$$

- $\text{OBJ}^*(s, I, h)$ at most $n+1$ quadratic/linear pieces
- A threshold $\rightarrow O(n)$ 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 $OBJ^*(s, I_i, h_i) \to O(|S|n)$ pieces.

- Find best $s$ efficiently.
Theorem: For any class $\mathcal{A}$ based on sigmoid-like parametrized rounding functions:

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

More general result

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?