Algorithms for Finding the Maxima of a Set of Vectors

- Problem Definition
- Main Theorem
- Lower Bound
- Dimension Sweep Algorithm
- Divide and Conquer Algorithm
- Outreach of KLP Algorithms
Learning Goals

1. Learn efficient algorithm for finding maxima in small or moderate sized sets of vectors based on:
   1. Dimension-sweep techniques
   2. Divide and Conquer techniques

2. Learn, review techniques to obtain results in computational complexity analysis

3. What are efficiency limits in obtaining Pareto fronts for finite sets?
Problem statement

Subset selection problem: Given a set of $n$ vectors $V$. Find all (Pareto) maximal vectors, i.e.:

Maximal Set: Find all $s \in V$ for $V \subset \mathbb{R}^d, |V| = n$, such that not exists $v \in V : v \succ s$

Question: What is the most efficient algorithm to solve this problem in terms of computational time complexity?

Computational complexity: $C_d(n) = \min_A \max_V c_d(A, V)$

Symbols:

$V$: Finite sets of size $n$
$A$: Algorithms (in standard computational model)
$d$: Dimension of vectors or number of objectives
$n$: Number of vectors
$c_d(A, V)$: Number of comparisons needed to compute set of maxima for a particular $A$ and $V$
Big O notation or Landau* Symbols

Let \( g : \mathbb{N} \to \mathbb{R} \) denote a function.

\[
O(g) = \{ f : \mathbb{N} \to \mathbb{R} \mid \exists C > 0, n_0 < \infty \text{ with } \forall n \geq n_0 : |f(n)| \leq Cg(n) \}
\]

\[
\Omega(g) = \{ f : \mathbb{N} \to \mathbb{R} \mid \exists C > 0, n_0 < \infty \text{ with } \forall n \geq n_0 : |f(n)| \geq Cg(n) \}
\]

\[
\Theta(g) = \bigcap \{ f : \mathbb{N} \to \mathbb{R} \mid \exists C > 0, n_0 < \infty \text{ with } \forall n \geq n_0 : |f(n)| \leq Cg(n) \cap |f(n)| \geq Cg(n) \}
\]

\[
f(n) \in O(g(n)) \iff \lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty
\]

\[
f(n) \in \Omega(g(n)) \iff \lim_{n \to \infty} \frac{f(n)}{g(n)} > 0
\]

\[
f(n) \in \Theta(g(n)) \iff 0 < \lim_{n \to \infty} \frac{f(n)}{g(n)} < \infty
\]

*E. Landau (1877 – 1938) German Jewish mathematician
KLP’s Complexity Bounds


\[ C_d(n) \in O(n \log n) \text{ for } d = 2, 3 \]

\[ C_d(n) \in O(n(\log n)^{d-2}) \text{ for } d \geq 4, \text{ and} \]

\[ C_d(n) \in \Omega(\lceil \log(n!) \rceil) \]

Note: \( \log(n!) = \log(1) + \log(2) + \ldots + \log(n) \leq n \log(n) \).

\[ \log(n!) > \log(n/2) + \ldots + \log(n) = n/2(\log(n) − 1). \]

\[ \log(n!) \in O(n \log(n)) \cap \Omega(n \log(n)) = \Theta(n \log(n)). \]
Lower bound

In order to establish a lower bound let us review a classical result by Knuth:

There are $n!$ ordered sequences, say $a_1, \ldots, a_n$ with $a_1 < \ldots < a_n$.

A comparison can be viewed as a branch in a binary decision tree.

The leaves of the decision tree are the ordered sequences, and hence there are $n!$ leaves.

To obtain a single ordering by binary decisions (comparisons) requires $\lceil \log_2(n!) \rceil$ decisions, which is the height of a maximally balanced search tree.
Lower bounds

We consider a decision tree, where each leaf is associated with a direct graph of an ordering and there is an arc from \( a_i \) to \( a_j \) if and only if \( a_i < a_j \) was the result on the path from the root to the leaf.

A sequence of vectors \((a_1, b_1), \ldots, (a_n, b_n)\) contains only maximal elements, iff \( \forall i, j \in \{1, \ldots, n\} : a_i < a_j \Leftrightarrow b_i > b_j \)

For each pair \((a_i, b_i)\) the algorithm must decide (from the results of the comparisons) that for any other vector \((a_j, b_j)\) either \( a_j < a_i \) or \( b_j < b_i \).

In order to decide whether or not all \((a_i, b_i)\) are maximal, this requires the transitive closure of the directed graph to have an arc between every pair of nodes.

This information is sufficient to determine an ordering on the nodes, and to acquire this information must take time \( \log([n!]) \).
Upper complexity bounds

This algorithm finds the maximal set for every pre-ordered set \((V, \succ)\) of size \(n\) with \(T(n) \in O(n^2)\) pairwise comparisons:

1. INPUT Preordered set: \((V, \succ)\)

2. FOR ALL \(i \in \{1, \ldots, n\}\)

3. \(t_i = true\)

4. FOR ALL \(j \in \{1, \ldots, n\}\)

5. IF \(v_j \succ v_i\) THEN \(t_i = false; \) BREAK

6. IF \(t_i = true\) OUTPUT \(v_i\) (is maximal)
Complexity bounds for partially ordered sets

It can be shown that $\Theta(n^2)$ is the time complexity for finding the minimal (or maximal) set of a partial ordered set of a general partially ordered set (no additional structure is given).

Firstly, the naïve algorithm (previous slide) shows that the problem is in $O(n^2)$.

To prove a lower bound consider that it needs to decide whether or not all elements are maximal.

To show this we need to check all pairs of points, say $v_i$ and $v_j$.

Note, that if we leave out some pair, we do not know whether $a_i$ and $a_j$ are in a domination relation and if $a_i$ or $a_j$ are not dominated so far, this information is needed.
Efficient algorithm for 2-D

For sets of 2 – D vectors we can find Pareto maximal points with this algorithm in \( T(n) \in O(n \log(n)) \) comparisons:

1. INPUT Sequence \((x_1, y_1), \ldots, (x_n, y_n)\) sorted by the x-coordinate in descending order.

2. SET \( i \leftarrow 1; T_0 = \emptyset; y_0^* \leftarrow -\infty. \)

3. IF \( y_i < y_i^* \) SET \( T_i \leftarrow T_{i-1}, y_i^* \leftarrow y_{i-1}^* \)

4. ELSE SET \( T_i = T_{i-1} \cup \{v_i\}; y_i^* \leftarrow y_i \)

5. IF \( i = n \) RETURN \( T_i \) ELSE \( i = i + 1 \) GOTO 3

Cost of algorithm is governed by bound in \( O(n \log(n)) \) on sorting \( V \). Hence, \( c_2(A, V) \in O(n + n \log(n)) = O(n \log n) \) and, using lower bound, \( C_2(n) \in \Theta(n \log(n)). \)
2-D Maximal Set Algorithm Animation
Efficient algorithm for 3-D

The following definition is needed to generalize the algorithm to 3-D:

For a vector \( v = (v_y, v_z)^T \) and a set of vectors \( YZ \) we say \( v \prec YZ \), iff \( \exists (y, z) \in YZ : (y, z)^T \succ (v_y, v_z)^T \).

Alternatively, using cones, we can define:

\[
(v_x, v_y)^T \prec YZ, \text{ iff } (v_y, v_z)^T \in \bigcup_{(y, z)^T \in YZ} ((y, z)^T \ominus (\mathbb{R}^d \setminus 0)) .
\]
3-D KLP: Algorithm design

The deeper, the darker ...
3-D KLP: Algorithm design – three cases when adding the next lower point

Point is non-dominated by boundary.

2-D boundary dominates point.

... and dominates other point(s).
AVL Tree to find dominated points in YZ

$\begin{pmatrix} -\infty & \infty \end{pmatrix}^T$

$\begin{pmatrix} \infty & -\infty \end{pmatrix}^T$
3-D KLP algorithm animation

- maximal vectors in YZ
- maximal vectors not in YZ
- dominated vectors
Efficient algorithm for 3-D

For $3 - D$ vectors we can generalize this algorithm:

1. INPUT Sequence $(x_1, y_1, z_1), \ldots, (x_n, y_n, z_n)$ sorted by the $x$-coordinate in descending order.

2. SET $i \leftarrow 1$; $T_0 = \emptyset$; $YZ_0^* \leftarrow \emptyset$.

3. IF $(y_i, z_i) < YZ_{i-1}^*$ SET $T_i \leftarrow T_{i-1}$, $YZ_i^* \leftarrow YZ_{i-1}^*$

4. ELSE SET $T_i = T_{i-1} \cup \{v_i\}$;

5. $YZ_i^* \leftarrow \text{Maxima}(YZ_{i-1}^* \cup \{(y_i, z_i)\})$

6. IF $i = n$ RETURN $T_i$ ELSE $i \leftarrow i + 1$; GOTO 3
Algorithm discussion

Balanced search tree can maintain $y$ for $YZ_i^*$, that is the non-dominated points in the $yz$-plane.

Step 5: Might take time in $O(n \log(n))$.

However, nodes can only enter and leave $YZ$ once.

Therefore total effort over all iterations for step 5 is $O(n \log(n))$.

Finding neighbors of new point in AVL tree, inserting and deleting single elements from AVL tree takes time in $O(\log(n))$.

In summary: The total effort of the algorithm is in $O(n \log(n))$. Therefore, the algorithm can be considered as efficient (or asymptotically optimal).
Efficient algorithm for N-D

For $d \geq 4$ KLP suggested a divide and conquer algorithm.

1. FUNCTION $M = \text{Maxima}(V, d)$

2. INPUT Sequence of $d$ dimensional vectors $V = (v_1, ..., v_n)$ sorted by the first coordinate.

3. Partition $V$ into subsets $R = (v_1, ..., v_{n/2})$ and $S = (v_{n/2+1}, ..., v_n)$.

4. $\hat{R} = \text{Maxima}(R)$ and $\hat{S} = \text{Maxima}(S)$ and $T$ is the subset of vectors in $\hat{S}$ that is not dominated by vectors in $\hat{R}$.

5. $M \leftarrow \hat{R} \cup T$
Subproblem: Efficiently determine elements of a set $S$ that are not dominated by a set $R$

1. Arrange elements of $R$ as $(u_1, \ldots, u_r)$ and the elements of $S$ as $(v_1, \ldots, v_s)$ such that $\#1(u_1) > \ldots > \#1(u_r)$ and $\#1(v_1) > \ldots > \#1(v_s)$

2. SET $\#1(u_0) \leftarrow \infty$; $\#1(u_{n+1}) \leftarrow -\infty$

3. FIND $k$ such that $\#1(u_k) \geq \#1(v_{s/2}) > \#1(u_{k+1})$.

4. SET $R_1 \leftarrow (u_1, \ldots, u_k)$ AND $R_2 \leftarrow (u_{k+1}, \ldots, u_r)$

5. SET $S_1 \leftarrow (v_1, \ldots, v_{s/2})$ AND $S_2 = (v_{s/2+1}, \ldots, v_s)$

6. $T \leftarrow \begin{bmatrix} R_1 \\ S_1 \end{bmatrix} \cap \begin{bmatrix} R_2 \\ S_1 \end{bmatrix} \cup \begin{bmatrix} R_1 \\ S_2 \end{bmatrix} \cap \begin{bmatrix} R_2 \\ S_2 \end{bmatrix}$.

Here $\begin{bmatrix} R \\ S \end{bmatrix}$ denotes the elements of $S$ that are not dominated by any element in $R$, and $\#1(v)$ is the first component of vector $v$. 
How to efficiently determine $T$

$S_1 = \begin{bmatrix} R_2 \\ S_1 \end{bmatrix}$, because all elements in $S_1$ have better first coordinates than all elements in $R_2$.

As the first coordinate in $S_2$ is already always worse than the second coordinate in $S_1$, elements in $S_2$ can only qualify due to their other coordinates for $T$.

Hence, it is a $d - 1$ dimensional problem to find $\begin{bmatrix} R_1 \\ S_2 \end{bmatrix}$

$F_d(r, s) = \min_A \max_{|R|=r,|S|=s} f_d(A, R, S)$

$f_d(A, R, S)$ is the number of comparisons to solve $\begin{bmatrix} R \\ S \end{bmatrix}$ with $A$.

$F_d(r, s) \leq \underbrace{F_d(k, s/2)}_{\text{for} \begin{bmatrix} R_1 \\ S_1 \end{bmatrix}} + \underbrace{F_d(r - k, s/2)}_{\text{for} \begin{bmatrix} R_2 \\ S_2 \end{bmatrix}} + \underbrace{F_{d-1}(k, s/2)}_{\text{for} \begin{bmatrix} R_1 \\ S_2 \end{bmatrix}} + \underbrace{ds/2}_{\text{for} \begin{bmatrix} R_2 \\ S_1 \end{bmatrix}}$
How to efficiently find $T$ for 3-D

For solving $\begin{bmatrix} R \\ S \end{bmatrix}$ for $d = 3$, KLP propose an algorithm with running time $O(r \log(r)) + O(s \log(r))$:

The algorithm is similar to the previously described dimension sweep algorithm for computing the Maxima of a set of $3 - D$ vectors.

We obtain $F_4(r, s) \leq (\alpha_d r + \beta_d s) \log(r)(\log(s)) + dr$

By structural induction (not discussed in detail, here ..):

$$F_d(r, s) \leq (\alpha_d r + \beta_d s) \log(r)(\log(s))^{d-3} + dr$$

And hence:

$$C_d(V) \leq n \log(n)^{d-2}$$
Proof idea: Master Theorem

Let us divide a problem with input size \( n \) into \( \alpha \) sub-problems of size \( n/\beta \).

Let \( T(n) \) denote the time for solving the problem with \( n \) inputs.

Let \( M(n) \) denote the time to merge the results from the \( \alpha \) subproblems of size \( n/\beta \).

Solution: see Master Theorem of Algorithms.
For continuous argument \( x \) we can sketch the proof:

Assuming \( T(x) = 0 \) for \( x < 1 \) we can solve the recursion:

\[
T(x) = x + \alpha T(x/\beta) \\
= x + \alpha x/\beta + \alpha^2 T(x/\beta^2) \\
= x + \alpha x/\beta + \alpha^2 x/\beta^2 + \ldots + \alpha^t T(x/\beta^t)
\]

For \( \alpha = \beta \) and \( t = \log_\beta(x) \) we get \( T(x) = x \log_\beta x + T(1) \).
Efficient algorithm for 3-D merge (finding $T$)

Not discussed

For nondominated sets $R$ and $S$ of 3-D vectors, find $\begin{bmatrix} R \\ S \end{bmatrix}$.

1. Arrange $R$ as $(u_1, \ldots, u_r)$ descending in $x$-coordinate.

2. Arrange $S$ as $(v_1, \ldots, v_s)$ such that $v_i \leq u_j \leq v_h \Rightarrow i < h$. (binary insertion)

3. $i \leftarrow 1$; $j \leftarrow 1$; $T \leftarrow \emptyset$; $YZ \leftarrow \emptyset$

4. IF $(v_i^x < u_j^x)$ GOTO 7

5. IF $((u_j^y, u_j^z) \prec YZ)$ $j \leftarrow j + 1$; GOTO 4;

6. ELSE $j \leftarrow j + 1$; $YZ \leftarrow \text{Maxima}(YZ \cup \{(u_j^y, u_j^z)\})$

7. IF not($((v_i^y, v_i^z) \prec YZ)$ THEN $T \leftarrow T \cup \{v_j\}$

8. IF $(i = s)$ RETURN $T$ ELSE $i = i + 1$ GOTO 4
Take home messages

- KLP establish lower and upper bound for the complexity of finding maximal vectors

\[ C_d(n) \in O(n \log n) \text{ for } f = 2, 3 \]

\[ C_d(n) \in O(n(\log n)^{d-2}) \text{ for } d \geq 4, \text{ and} \]

\[ C_d(n) \in \Omega([\log(n!)]) \]

- Dimension sweep and Divide and Conquer paradigms are used in combination, to construct algorithm in 3-D (know construction of algorithm)
- Today, KLP bounds still hold and have only been improved for special cases; Upper bounds not sharp – can they be improved?
- KLP used in MODA Algorithms and Skyline Queries to Databases
- KLP-like algorithms also used for other questions in MODA, e.g. efficient Performance assessment and archiving