Approximate Polytope Membership Queries

Sunil Arya
Hong Kong University of Science and Technology

Guilherme D. da Fonseca
Universidade Federal do Estado do Rio de Janeiro (UNIRIO)

David M. Mount
University of Maryland, College Park

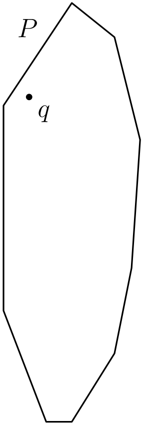
STOC 2011

Polytope Membership Queries

Polytope Membership Queries

Given a polytope P in d-dimensional space, preprocess P to answer membership queries:

Given a point q, is $q \in P$?

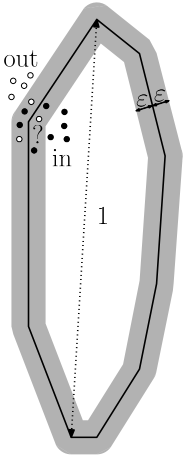


- Assume that dimension d is a constant and P is given as intersection of n halfspaces
- For $d \leq 3$, can be solved with storage $O(n)$ and query time $O(\log n)$ [BCKO10]
- Dual of halfspace emptiness searching

Approximate Polytope Membership Queries

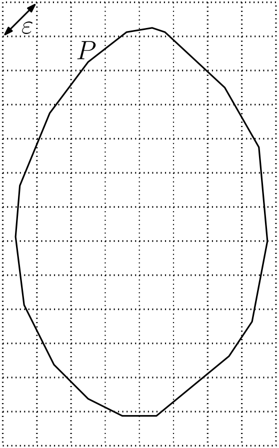
Approximate Version

- An approximation parameter ε is given (at preprocessing time)
- Assume the polytope has diameter 1
- If the query point's distance from P's boundary:
  - $> \varepsilon$: answer must be correct
  - $\leq \varepsilon$: either answer is acceptable



- Polytope approximation is a well studied topic
- We consider the first space-time tradeoffs for the query problem

Bentley *et al.* (Outer) Approximation [BFP82]

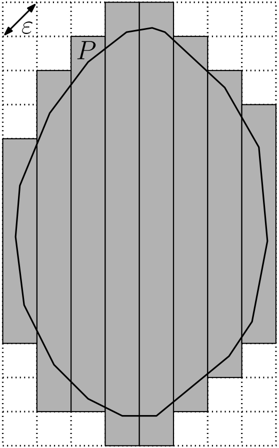


- Create a grid with cells of diameter ε
- For each column, store the topmost and bottommost cells intersecting P
- Query processing:
  - Locate the column that contains q
  - Compare q with the two extreme values

Time-Efficient Solution [BFP82]

- $O(1/\varepsilon^{d-1})$ columns
- Storage: $O(1/\varepsilon^{d-1})$
- Query time: $O(1)$ (by integer division)

Bentley *et al.* (Outer) Approximation [BFP82]

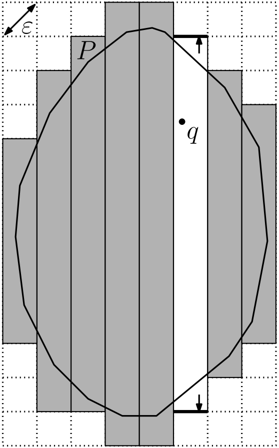


- Create a grid with cells of diameter ε
- For each column, store the topmost and bottommost cells intersecting P
- Query processing:
  - Locate the column that contains q
  - Compare q with the two extreme values

Time-Efficient Solution [BFP82]

- $O(1/\varepsilon^{d-1})$ columns
- Storage: $O(1/\varepsilon^{d-1})$
- Query time: $O(1)$ (by integer division)

Bentley *et al.* (Outer) Approximation [BFP82]

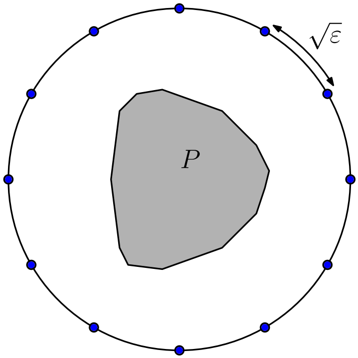


- Create a grid with cells of diameter ε
- For each column, store the topmost and bottommost cells intersecting P
- Query processing:
  - Locate the column that contains q
  - Compare q with the two extreme values

Time-Efficient Solution [BFP82]

- $O(1/\varepsilon^{d-1})$ columns
- Storage: $O(1/\varepsilon^{d-1})$
- Query time: $O(1)$ (by integer division)

Dudley's (Outer) Approximation [Dud74]



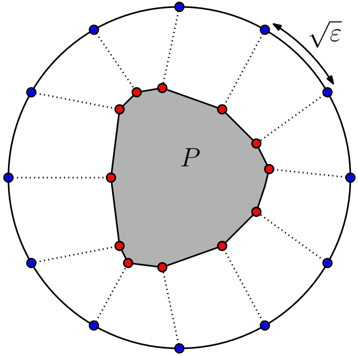
Every unit-diameter polytope can be ε-approximated as the intersection of $O(1/\varepsilon^{(d-1)/2})$ halfspaces [Dud74]

Space-Efficient Solution

Check whether q lies within each Dudley halfspace:

- Storage: $O(1/\varepsilon^{(d-1)/2})$
- Query time: $O(1/\varepsilon^{(d-1)/2})$
- Note: Each halfspace is used to cover a surface patch of size $\sqrt{\varepsilon}$

Dudley's (Outer) Approximation [Dud74]



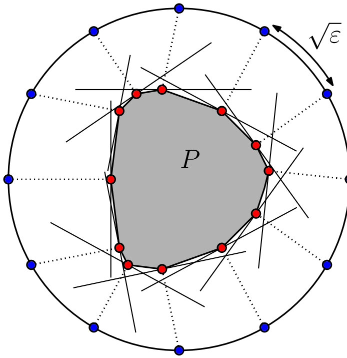
Every unit-diameter polytope can be ε-approximated as the intersection of $O(1/\varepsilon^{(d-1)/2})$ halfspaces [Dud74]

Space-Efficient Solution

Check whether q lies within each Dudley halfspace:

- Storage: $O(1/\varepsilon^{(d-1)/2})$
- Query time: $O(1/\varepsilon^{(d-1)/2})$
- Note: Each halfspace is used to cover a surface patch of size $\sqrt{\varepsilon}$

Dudley's (Outer) Approximation [Dud74]



Every unit-diameter polytope can be ε-approximated as the intersection of $O(1/\varepsilon^{(d-1)/2})$ halfspaces [Dud74]

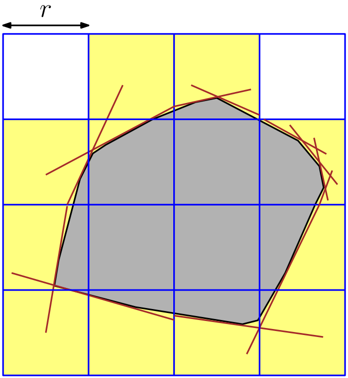
Space-Efficient Solution

Check whether q lies within each Dudley halfspace:

- Storage: $O(1/\varepsilon^{(d-1)/2})$
- Query time: $O(1/\varepsilon^{(d-1)/2})$
- Note: Each halfspace is used to cover a surface patch of size $\sqrt{\varepsilon}$

A Simple Tradeoff

- Generate a grid of diameter $r \in [\varepsilon, 1]$
- Preprocessing: For each cell Q intersecting P's boundary:
  - Apply Dudley to $P \cap Q$
  - $O((r/\varepsilon)^{(d-1)/2})$ halfspaces per cell
- Query Processing:
  - Find the cell containing q
  - Check whether q lies within every halfspace for this cell

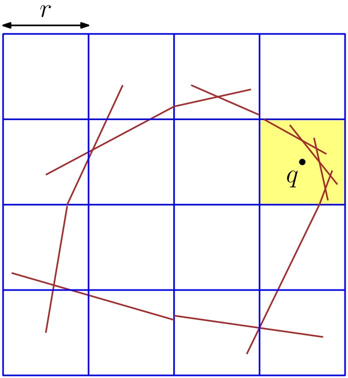


Tradeoff

- Storage: $O(1/(r\varepsilon)^{(d-1)/2})$
- Query time: $O((r/\varepsilon)^{(d-1)/2})$

A Simple Tradeoff

- Generate a grid of diameter $r \in [\varepsilon, 1]$
- Preprocessing: For each cell Q intersecting P's boundary:
  - Apply Dudley to $P \cap Q$
  - $O((r/\varepsilon)^{(d-1)/2})$ halfspaces per cell
- Query Processing:
  - Find the cell containing q
  - Check whether q lies within every halfspace for this cell

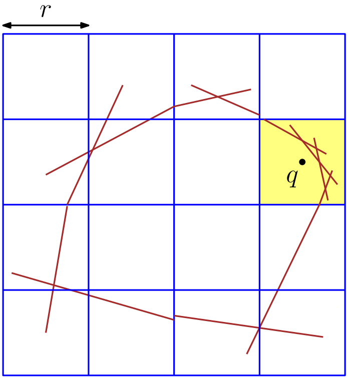


Tradeoff

- Storage: $O(1/(r\varepsilon)^{(d-1)/2})$
- Query time: $O((r/\varepsilon)^{(d-1)/2})$

A Simple Tradeoff

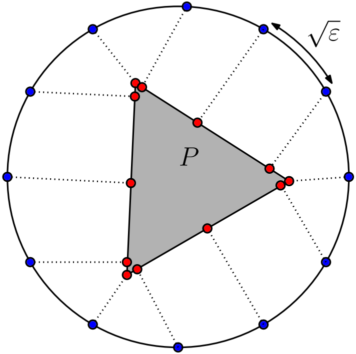
- Generate a grid of diameter $r \in [\varepsilon, 1]$
- Preprocessing: For each cell Q intersecting P's boundary:
  - Apply Dudley to $P \cap Q$
  - $O((r/\varepsilon)^{(d-1)/2})$ halfspaces per cell
- Query Processing:
  - Find the cell containing q
  - Check whether q lies within every halfspace for this cell



Tradeoff

- Storage: $O(1/(r\varepsilon)^{(d-1)/2})$
- Query time: $O((r/\varepsilon)^{(d-1)/2})$

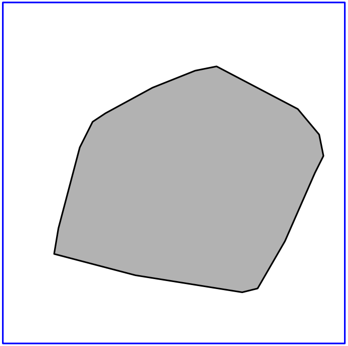
Can we do better? Need a little sensitivity



- Dudley tends to oversample regions of very low and very high curvature
- Finding the smallest number of halfspaces reduces to set cover
- A $\log(1/\varepsilon)$-approximation can be found efficiently (Mitchell and Suri [MS95], Clarkson [Cla93])
- Simple Idea: Recursively subdivide space (quadtree) until the number of approximating halfspaces is small enough

Split-Reduce

$t = 2$



Preprocess:

- Input P, ε, and desired query time t
- $Q \leftarrow$ unit hypercube
- Split-Reduce(Q)

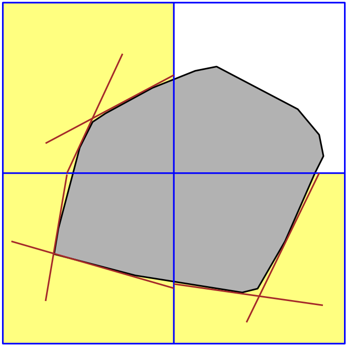
Split-Reduce(Q)

- Find an ε-approximation of $Q \cap P$
- If at most t facets, then Q stores them
- Otherwise, subdivide Q and recurse

- Query time: $O(\log(1/\varepsilon) + t)$
- Storage: ???

Split-Reduce

$t = 2$



Preprocess:

- Input P, ε, and desired query time t
- $Q \leftarrow$ unit hypercube
- Split-Reduce(Q)

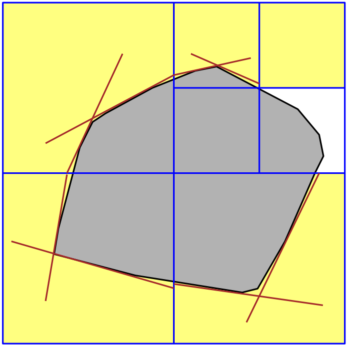
Split-Reduce(Q)

- Find an ε-approximation of $Q \cap P$
- If at most t facets, then Q stores them
- Otherwise, subdivide Q and recurse

- Query time: $O(\log(1/\varepsilon) + t)$
- Storage: ???

Split-Reduce

$t = 2$



Preprocess:

- Input P, ε, and desired query time t
- $Q \leftarrow$ unit hypercube
- Split-Reduce(Q)

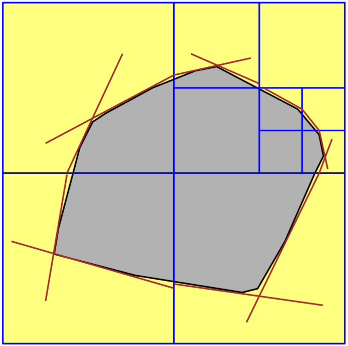
Split-Reduce(Q)

- Find an ε-approximation of $Q \cap P$
- If at most t facets, then Q stores them
- Otherwise, subdivide Q and recurse

- Query time: $O(\log(1/\varepsilon) + t)$
- Storage: ???

Split-Reduce

$t = 2$



Preprocess:

- Input P, ε, and desired query time t
- $Q \leftarrow$ unit hypercube
- Split-Reduce(Q)

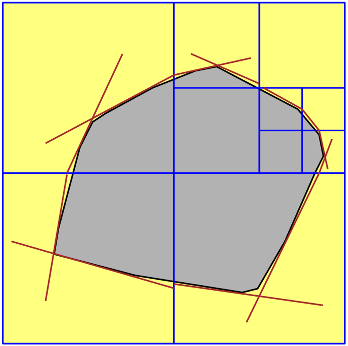
Split-Reduce(Q)

- Find an ε-approximation of $Q \cap P$
- If at most t facets, then Q stores them
- Otherwise, subdivide Q and recurse

- Query time: $O(\log(1/\varepsilon) + t)$
- Storage: ???

Split-Reduce

$t = 2$



Preprocess:

- Input P, ε, and desired query time t
- $Q \leftarrow$ unit hypercube
- Split-Reduce(Q)

Split-Reduce(Q)

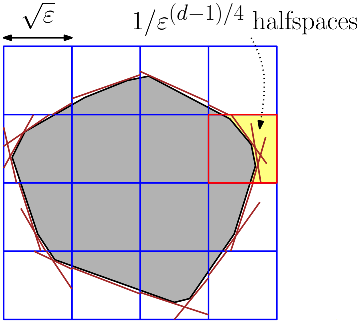
- Find an ε-approximation of $Q \cap P$
- If at most t facets, then Q stores them
- Otherwise, subdivide Q and recurse

- Query time: $O(\log(1/\varepsilon) + t)$
- Storage: ???

Why it pays to be sensitive

Easy Analysis

Split-Reduce reduces the query time from $1/\varepsilon^{(d-1)/2}$ to $1/\varepsilon^{(d-1)/4}$ with the same $O(1/\varepsilon^{(d-1)/2})$ storage



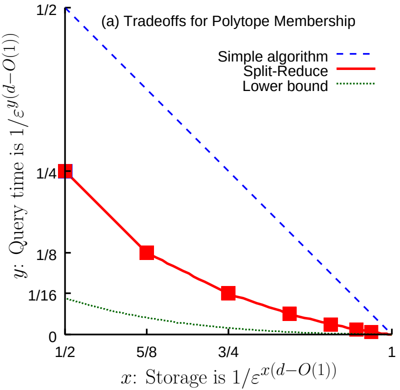
- By Dudley, if diameter $\leq \sqrt{\varepsilon}$, need only $1/\varepsilon^{(d-1)/4}$ halfspaces
  $\Rightarrow$ cells of size $\leq \sqrt{\varepsilon}$ are not subdivided
- Each Dudley halfspace is only needed within a radius of $\sqrt{\varepsilon}$
  $\Rightarrow$ Each halfspace hits only $O(1)$ cells of size $\geq \sqrt{\varepsilon}$
  $\Rightarrow$ The total number of halfspaces needed is $O(1/\varepsilon^{(d-1)/2})$

General Tradeoff

An inductive application of the previous argument yields a space-time tradeoff

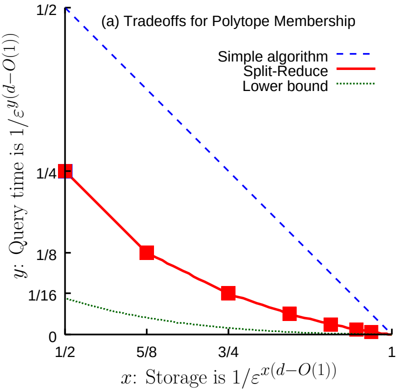
Theorem:

Using Split-Reduce we can answer ε-approximate polytope membership queries with
Storage: $O(1/\varepsilon^{(d-1)/(1-k/2^k)})$
Query time: $O(1/\varepsilon^{(d-1)/2^k})$

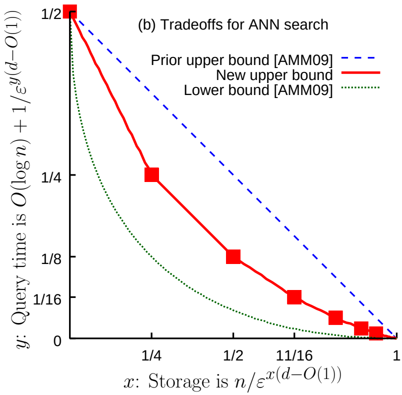


Lower Bound

- The above analysis is not necessarily tight
- We establish a lower bound on Split-Reduce
- The input polytope is a cylinder formed by extruding a $(d-k)$-dimensional ball in k dimensions
- k is chosen to maximize the storage for a given query time



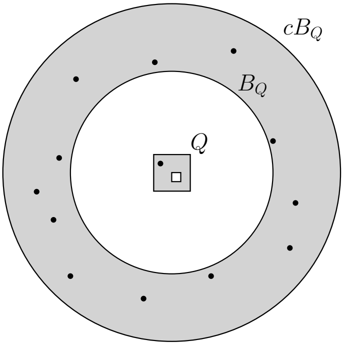
Approximate Nearest Neighbor (ANN) Searching



- ANN: Preprocess n points such that, given a query point q, can find a point within at most $1+\varepsilon$ times the distance to q's nearest neighbor
- Arya, *et al.* [AMM09] gave a solution that is optimal in the extremes of the space-time tradeoff and gave a lower bound
- Our new results improve the tradeoff throughout the middle of the spectrum

Approximate Nearest Neighbor (ANN) Searching

- Arya *et al.* show that it is possible to partition space into cells, each associated with candidates to be the ANN for query points in the cell, such that:
  - Total number of candidates is $\widetilde{O}(n)$
  - All but 1 candidate is inside a constant-radius annulus
- Using lifting we can reduce the search to $\log(1/\varepsilon)$ approximate polytope membership queries



Concluding Remarks

- Improved upper bounds for approximate polytope membership queries
- First space-time tradeoffs
- Simple algorithm – Split-Reduce
- Significant improvements to ANN searching

- Open problem: Tighten the analysis

Thank you!

Bibliography

- [AMM09] S. Arya, T. Malamatos, and D. M. Mount. Space-time tradeoffs for approximate nearest neighbor searching. *J. ACM*, 57:1–54, 2009.
- [BFP82] J. L. Bentley, M. G. Faust, and F. P. Preparata. Approximation algorithms for convex hulls. *Commun. ACM*, 25(1):64–68, 1982.
- [BCKO10] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2010.
- [Cla93] K. L. Clarkson. Algorithms for polytope covering and approximation. In *Proc. 3rd Workshop Algorithms Data Struct. (WADS)*, pages 246–252, 1993.
- [Dud74] R. M. Dudley. Metric entropy of some classes of sets with differentiable boundaries. *Approx. Theory*, 10(3):227–236, 1974.
- [MS95] J. S. B. Mitchell and S. Suri. Separation and approximation of polyhedral objects. *Comput. Geom.*, 5:95–114, 1995.