Nearest neighbors

Focus on tree-based methods

Clément Jamin, GUDHI project, Inria
March 2017

Introduction

- Exact and approximate nearest neighbor search
- Essential tool for many applications
- Huge bibliography

- In GUDHI:
  - High ambient dimension
  - Low to medium intrinsic dimension

  ➔ We need algorithms whose complexity depends on the intrinsic dimension.

Spaces

- Most general case :
  - Set of points *P*
  - Distance function *D(x,y)* defined for any *x, y* ∈ *P*

- Adding a few constraints, a *metric space* qualifies a space where the following conditions are satisfied:
  - Non-negativity: $D(x, y) \geq 0$
  - Small self-distance: $D(x, x) = 0$
  - Isolation: x != y implies D(x, y) > 0
  - Symmetry: D(x, y) = D(y, x)
  - The triangle inequality: D(x, z) ≤ D(x, y) + D(y, z)

- The Euclidean distance is of particular interest since a lot of (A)NN methods are relying on it.

Spaces

- Dimension
  - The **complexity** of most algorithms depends on it.
  - **Intrinsic** *vs* **ambient** dimension.
  - Abstract metric spaces ➔ **implicit structure** of the metric?
    - Try and define an analogous notion of dimensionality
    - Most common: **Assouad (or doubling) dimension**

> A metric space X with metric d is said to be **doubling** if there is **some constant M > 0** such that **for any x in X and r > 0**, it is **possible to cover the ball B(x, r)** with the **union of at most M many balls of radius r/2**. The base-2 logarithm of *M* is often referred to as the **doubling dimension** of *X*.

- Example: Euclidean space $\mathbb{R}$d

  ➔ doubling space where M depends on the dimension

Approximate nearest neighbor?

- ***ϵ*-approximation**
  - A data point *p* is a **(1 + ϵ)-approximate nearest neighbor** of *q* if its distance from *q* is within a factor of (1 + ϵ) of the distance to the **true nearest neighbor**.
  - More generally, for ***1 ≤ k ≤ n***, a *k*th *(1 + ϵ)*-approximate nearest neighbor of *q* is a data point whose relative error from the **true *k*th nearest neighbor** of *q* is *ϵ*.
- **Recall**
  - The recall is the **fraction of true nearest neighbors returned**:

    *Number of correct answers / (k * number of queries)*
  - **Example** for a 10-NN search: for each query, count the number of neighbors (among the 10 returned) than are among the true 10 nearest neighbors.
  - This approach is thus a **statistical approach**, which does not give an actual control on how big the error is, but only **on the probability of an error**.

Tree-based methods

- Widely used
- Organize data in a way that allow fast queries
- Numerous variants:
  - kd-tree
  - Balanced Box-Decomposition trees (BBD trees) [Arya et al. 1994]
  - Vantage-point trees (also called Metric trees) [Uhlmann 1991; Yianilos 1993]
  - Random Projection trees (RP trees) [Dasgupta and Freund 2008; Hyvönen et al. 2015]
  - RKD-trees [Muja and Lowe 2009]
  - kd-GeRaF [Avrithis, Emiris, and Samaras 2016]
  - Randomly-oriented RKD-trees [Nicolopoulos 2014]
  - Spill trees [Liu et al. 2004]
  - …

In green: ϵ-approximation
In orange: recall

Searching in trees

- Query point q

- The easy way: *defeatist search strategy*
  - **Recursively visit** the subtree containing q, ending up in **the leaf where q lies**.
  - Hopefully with a few of its closest data points.
  - **Fast**, but **may fail**: the nearest neighbors might lie in neighboring cells.
  - No way to guarantee an ϵ-approximation of the problem.

Searching in trees

- The ϵ-accurate way 1: *descending* (or *standard*) search
  - The bounded set N of **current closest neighbors** is maintained, along with their distance to q
  - The tree is explored in depth-first manner
  - At each node, the branch whose **bounding box** is the **closest to q** is first explored
  - When done, only explore the other branch **if its bounding box might contain** a point closer than the current "worst" element of N
    - Note: this is where ϵ is taken into account
  - E.g.: Flann, CGAL Spatial Searching.

- The ϵ-accurate way 2: *priority* search
  - Subtrees are not visited in the order they are encountered
  - Maintain a *priority queue*
  - While descending the tree:
    - Not-visited children are possibly enqueued
    - Priority is inversely proportional to their distance to q

Searching in trees

- Only trees where the splits are **orthogonal to an axis** are usually queried using the **descending or priority search**
  - kd-tree
  - BDD-tree
  - Randomly-oriented RKD-trees [Nicolopoulos 2014]
    - Forest of randomized kd-trees
    - All trees queried at the same time: *priority search* with **only one common priority queue**

    ➔ ϵ-approximation with better performance than a single kd-tree

- Note: we could not find any paper or implementation attempting to adapt such strategies to other kind of trees such as RP trees.

Searching in trees

- In trees that cut space in other ways (random projections, etc.):
  - Defeatist searches = relatively **high probability of failure**

  - **Balanced** by the use of **multiple randomized trees**, often called forest of trees [O'Hara and Draper 2013; Avrithis, Emiris, and Samaras 2016]
  - Trees are built so that they are **as different as possible from each other**
    - E.g. by randomly drawing the position of the split
  - The *recall* mainly depends on the number of trees

  - In the kd-GeRaF [Avrithis, Emiris, and Samaras 2016], this strategy is used with ***kd-trees*** (with randomized cutting position).

Trees and the *curse of dimensionality*

- Tree-based methods are affected by the *curse of dimensionality*
  - Exponential complexities
  - Sparse data
  - The difference in one coordinate is no longer a good lower bound for the distance.

  ➔ For high dimension, it is difficult to outperform the linear scan

- Possible solution: having complexities depend on the **intrinsic dimension** rather than **ambient dimension**.

Trees and the *curse of dimensionality*

- [Vempala 2012]
  - Starts from the fact that **kd-trees remain popular**
    - Even though they are supposed to be struck by the *curse of dimensionality*
  - How to get rid of pathological cases?
    - E.g. when points are distributed along *n* orthogonal lines, one parallel to each axis.

    ➔ Random rotation of the data points
  - Shows that kd-trees on randomly rotated data adapts to the intrinsic dimension

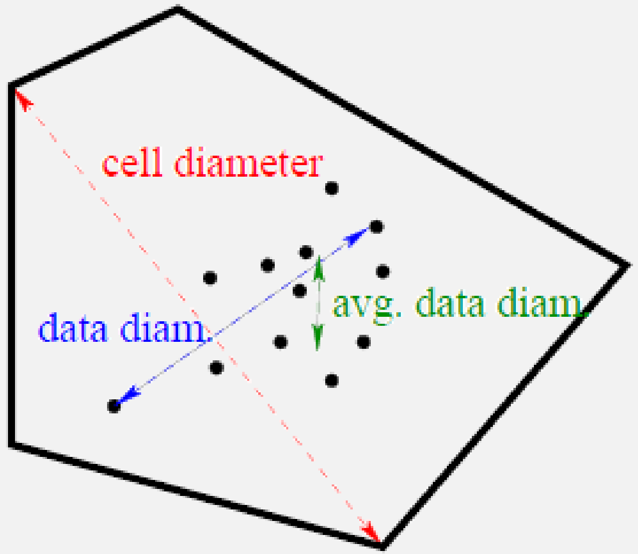
    + fast traversal time

    + most real-life cases are randomly oriented

  ➔ explains why kd-trees remain popular.

Trees and the *curse of dimensionality*

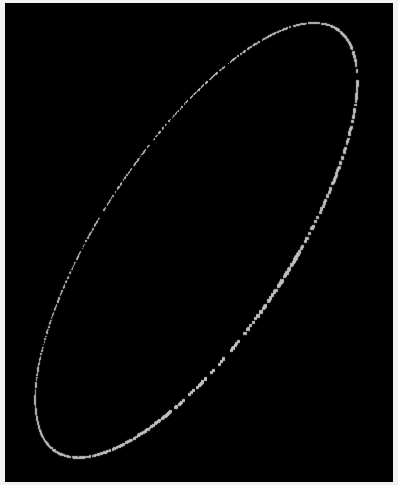
- [Verma, Kpotufe, and Dasgupta 2009]
  - "Which Spatial Partition Trees Are Adaptive to Intrinsic Dimension?"
  - Define the diameter of a tree cell ➔
  - Measure how the **average data diameter** decreases when going down the tree.



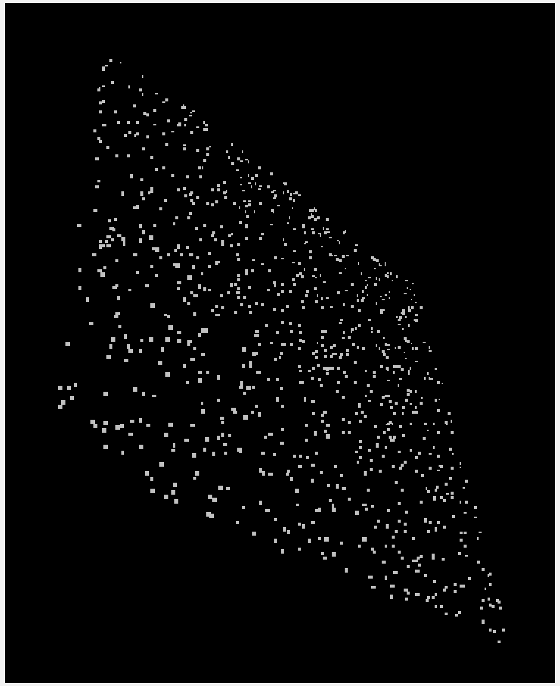
➔ kd-trees, RP trees, PCA trees, 2-means trees adapt to the intrinsic dimension

In practice

- Experiments:
  - **Synthetic datasets** ➔ control on intrinsic and ambient dimensions
  - Points on d-sphere and d-plane where ***d*** is the **intrinsic dimension**
  - Embedded in **ambient space of dimension *D***…
  - … with a random rotation in ambient space
  - **Query points**: lying close to existing points
  - Examples:

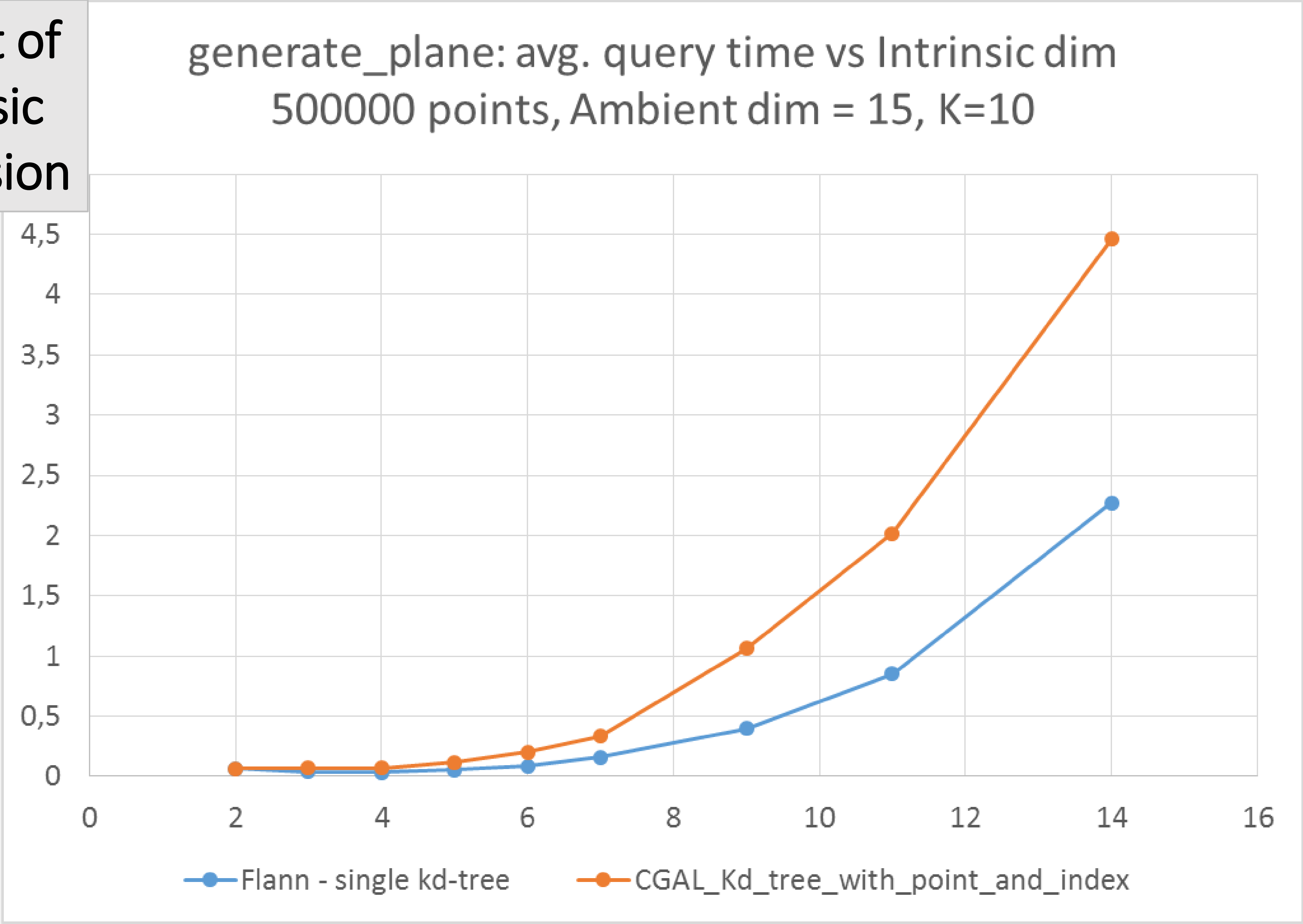


d=1
D=3

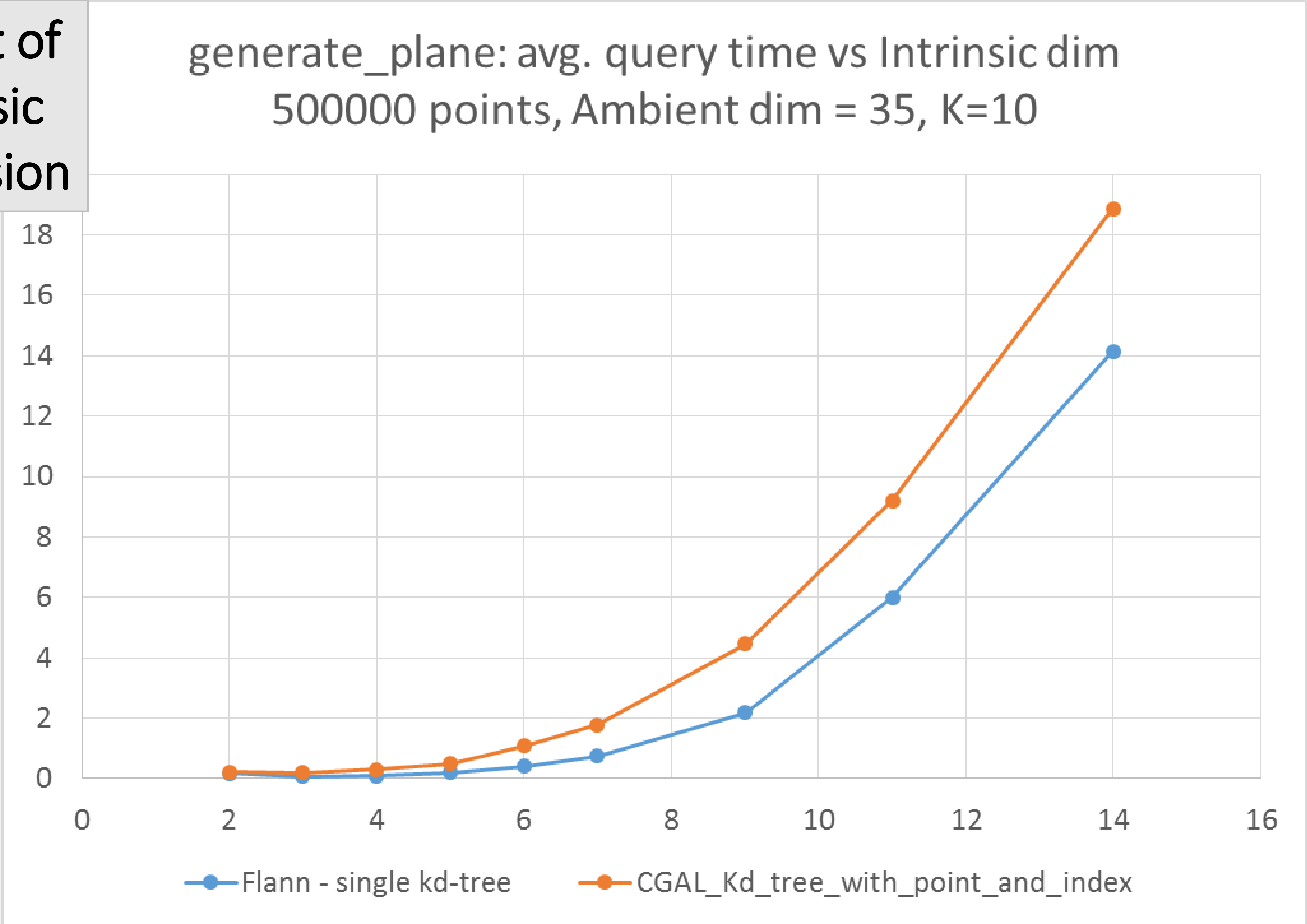


d=2
D=3

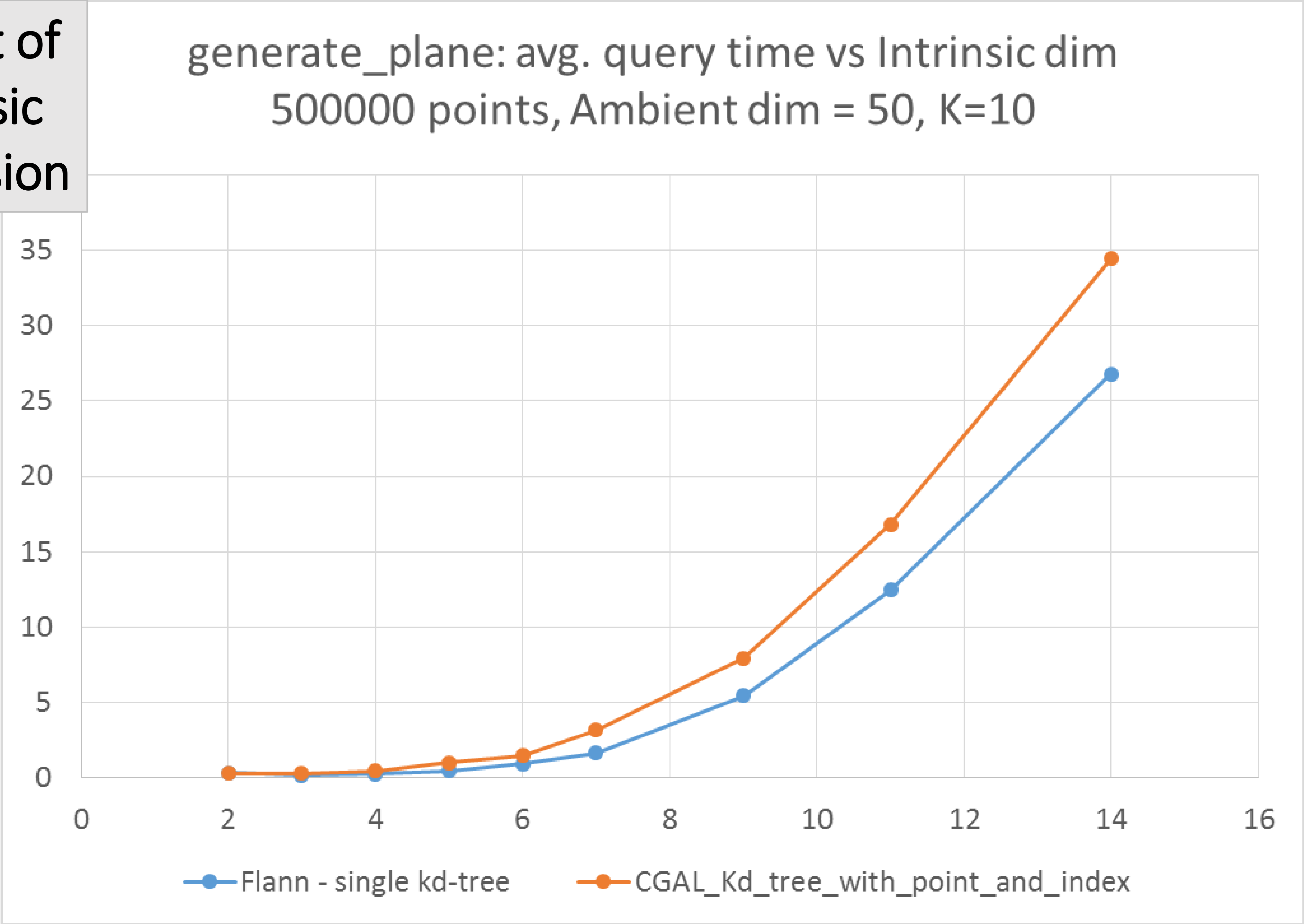
Impact of intrinsic dimension



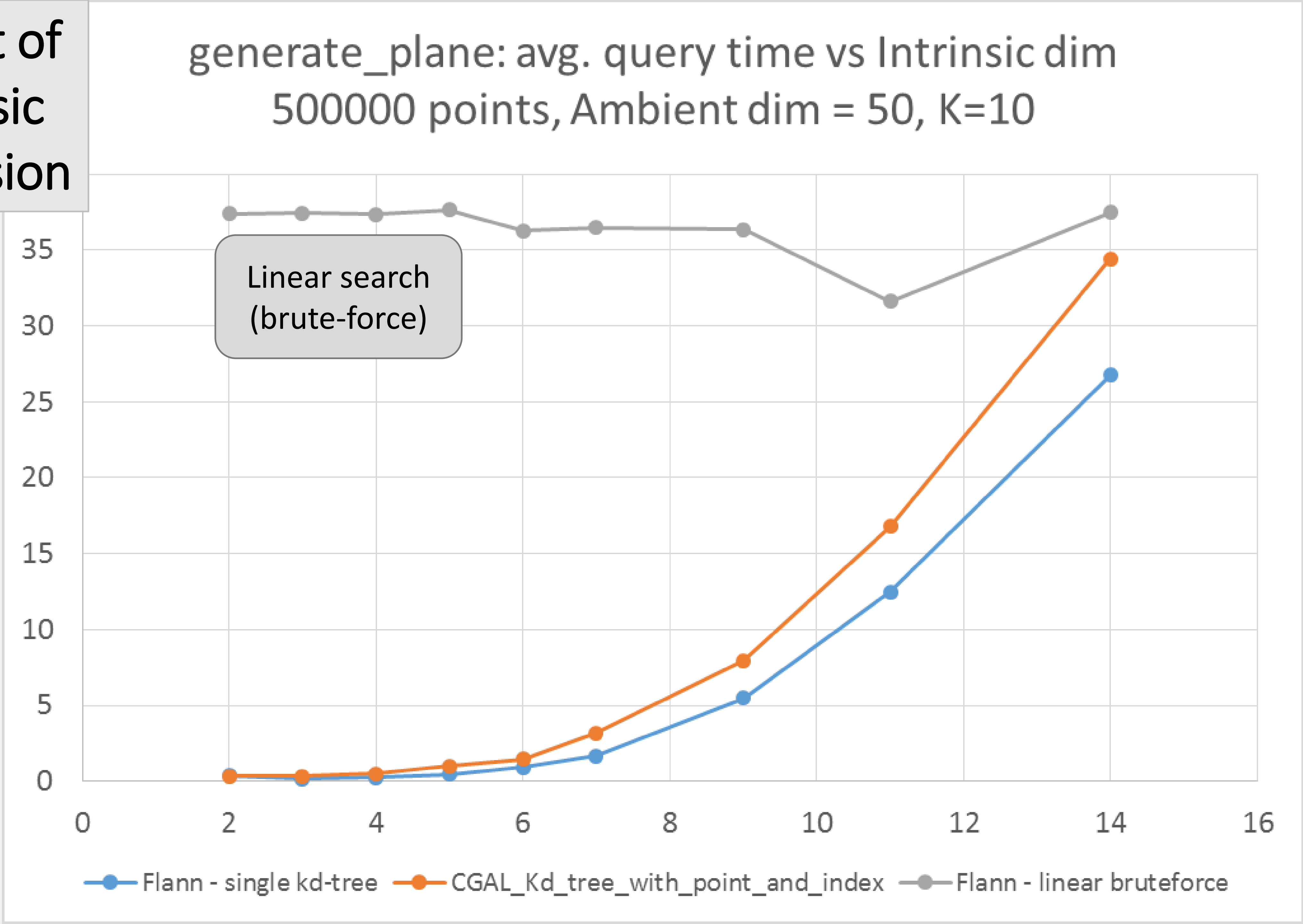
Impact of intrinsic dimension



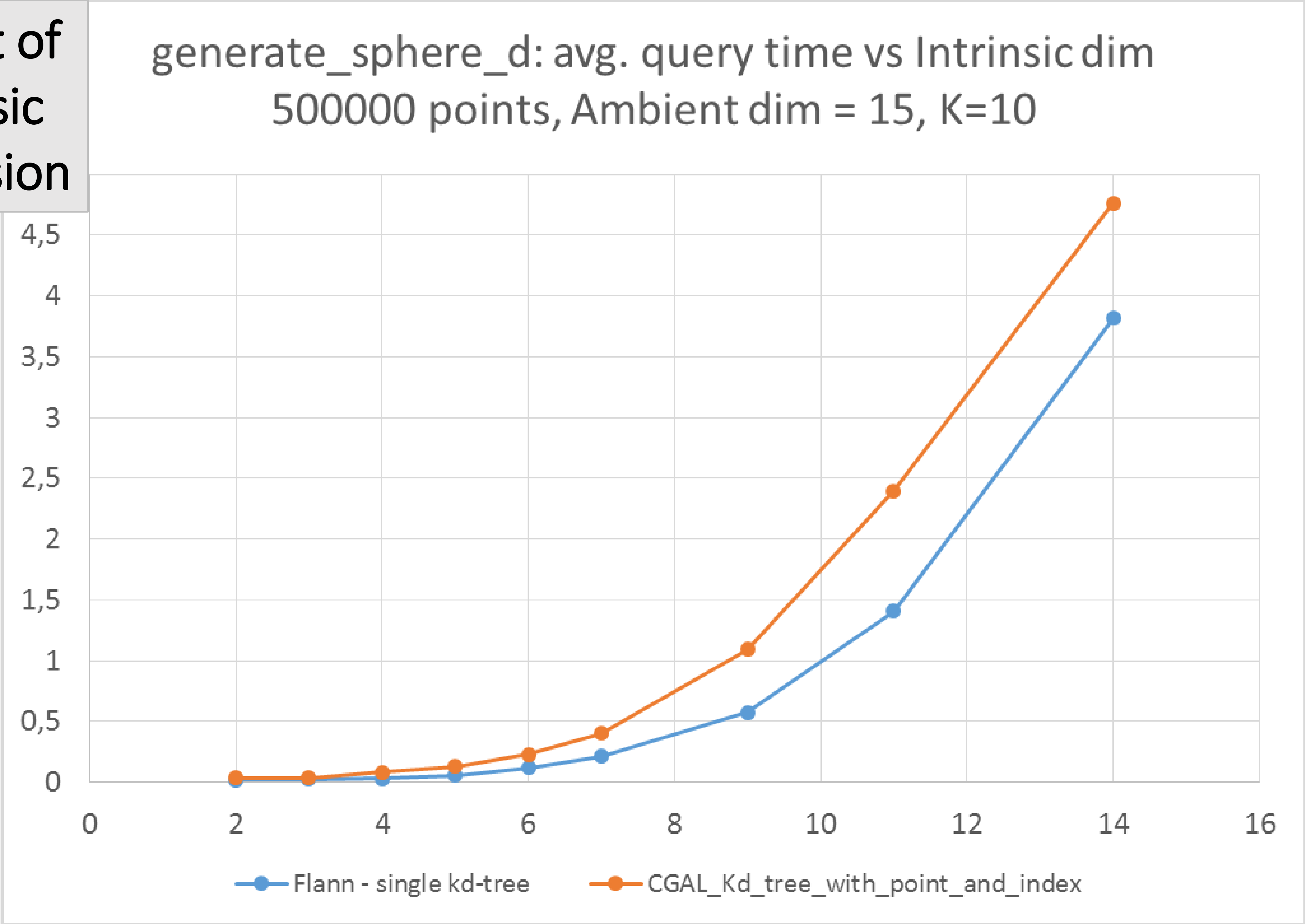
Impact of intrinsic dimension



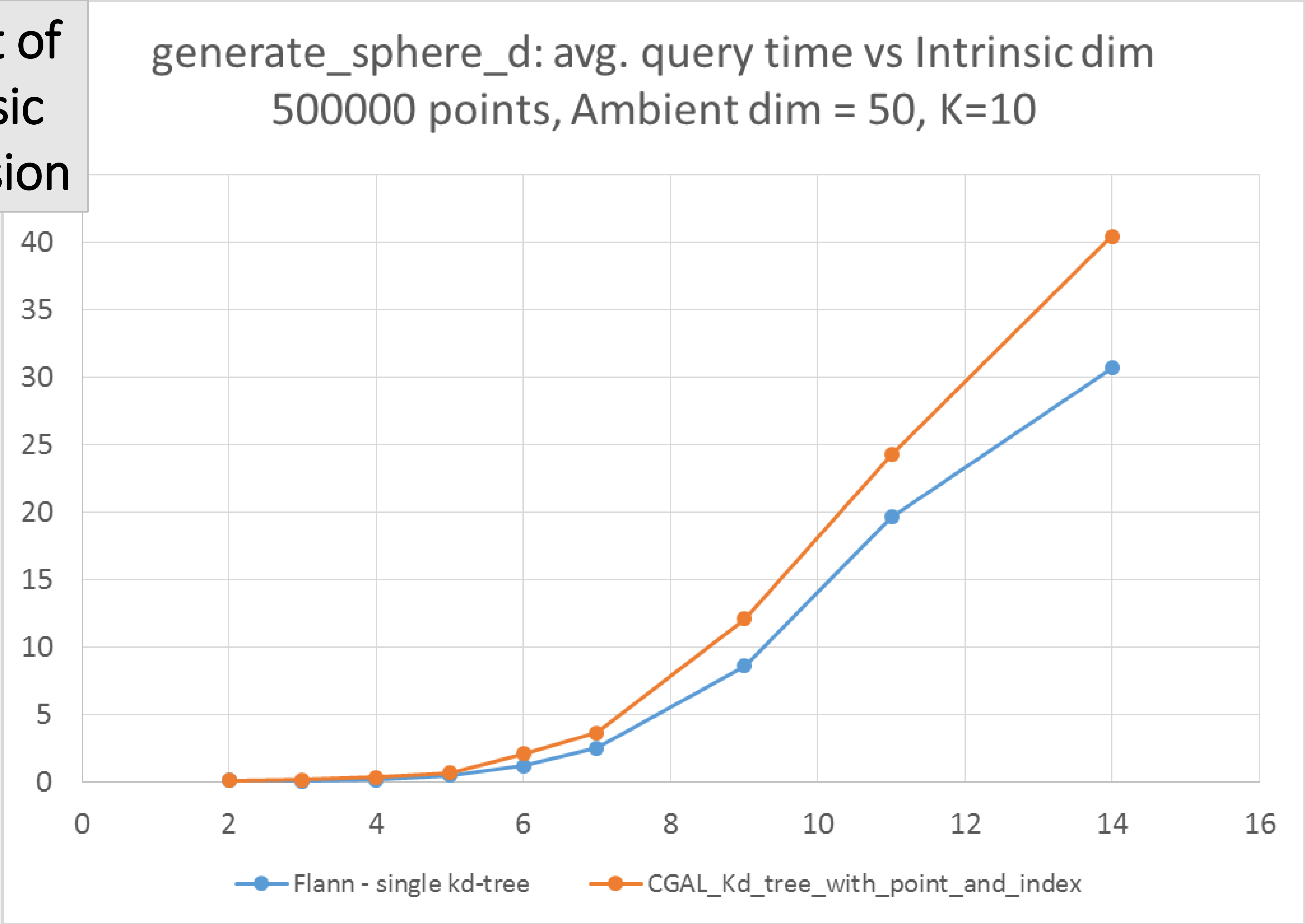
Impact of intrinsic dimension



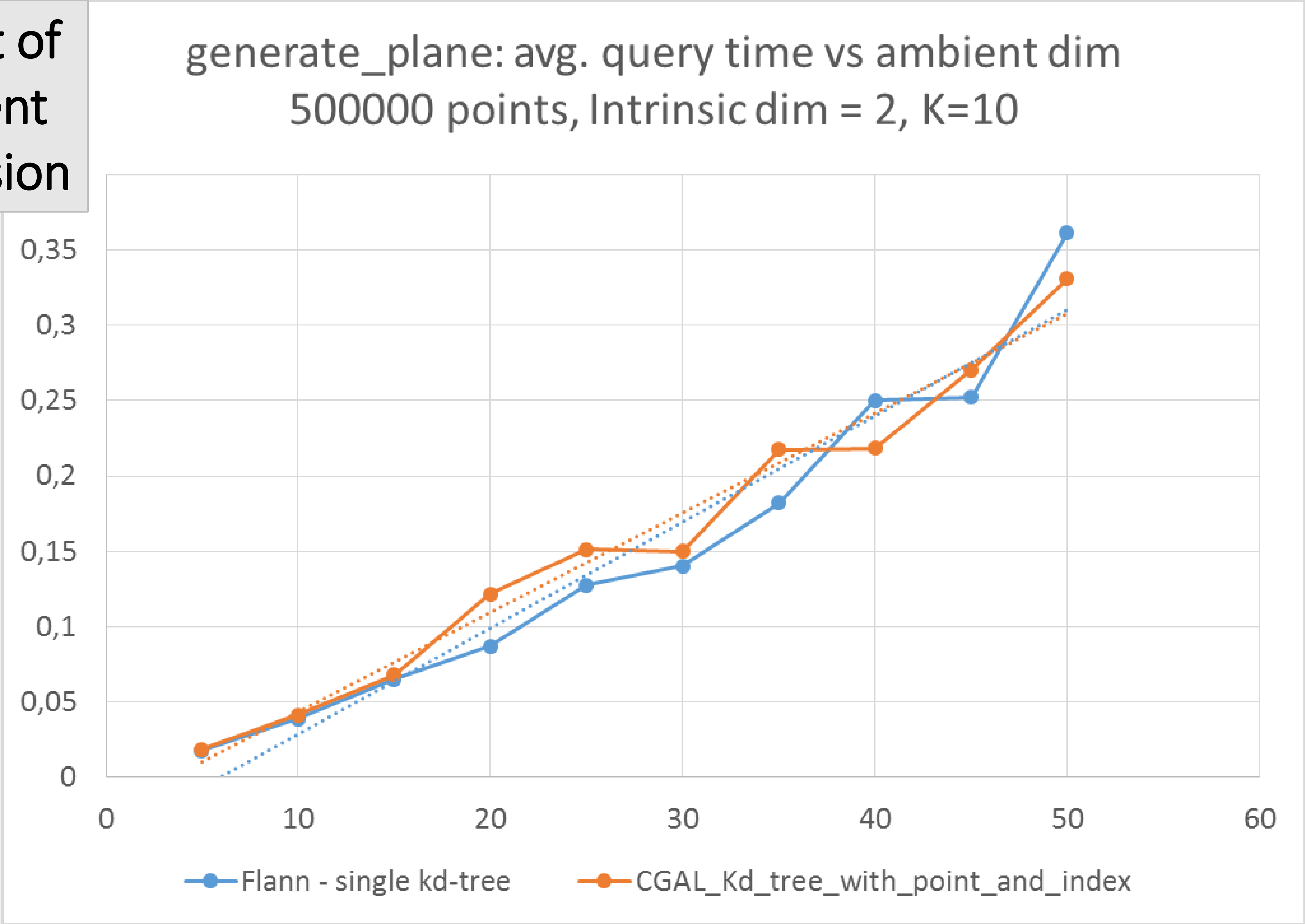
Impact of intrinsic dimension



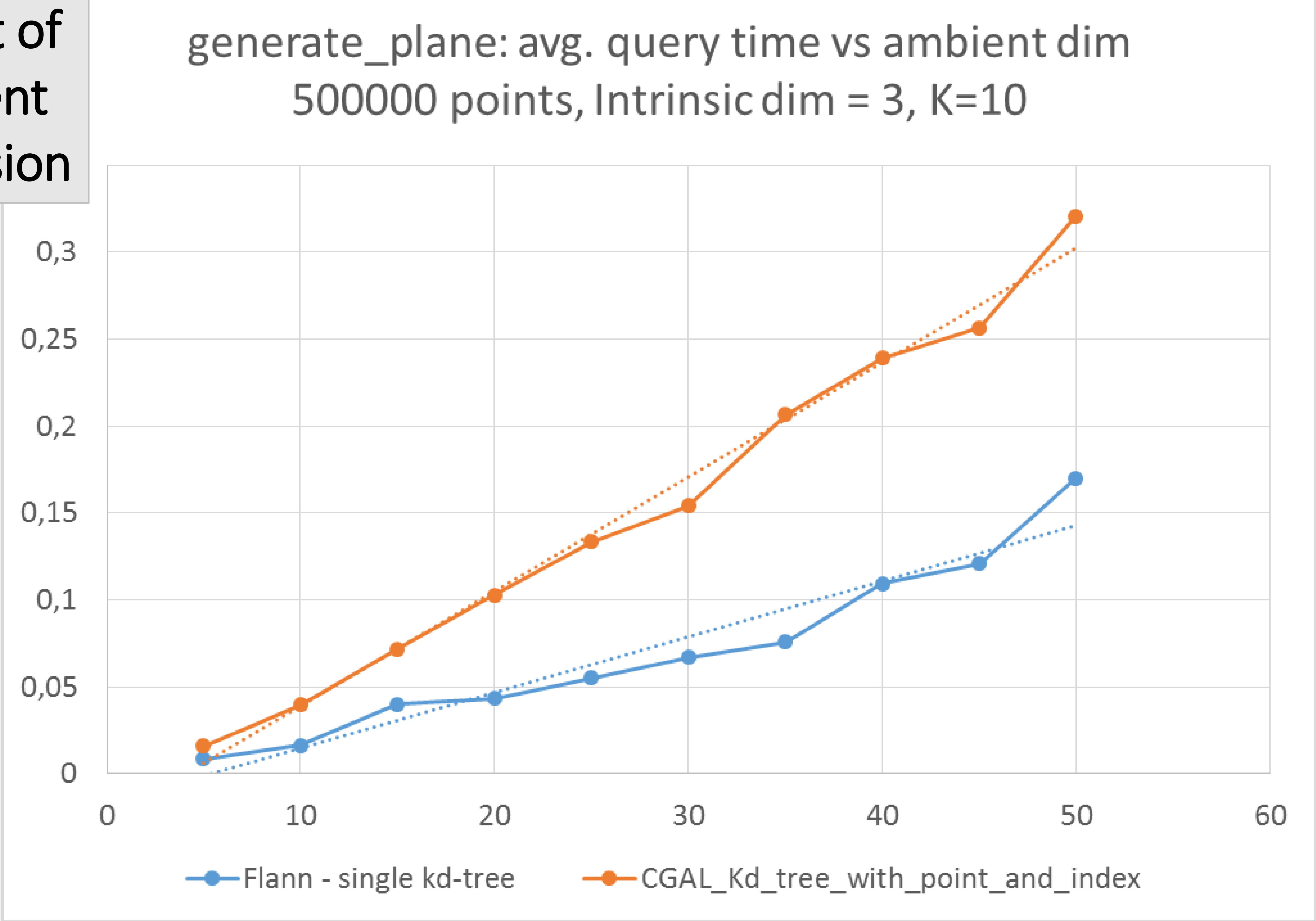
Impact of intrinsic dimension



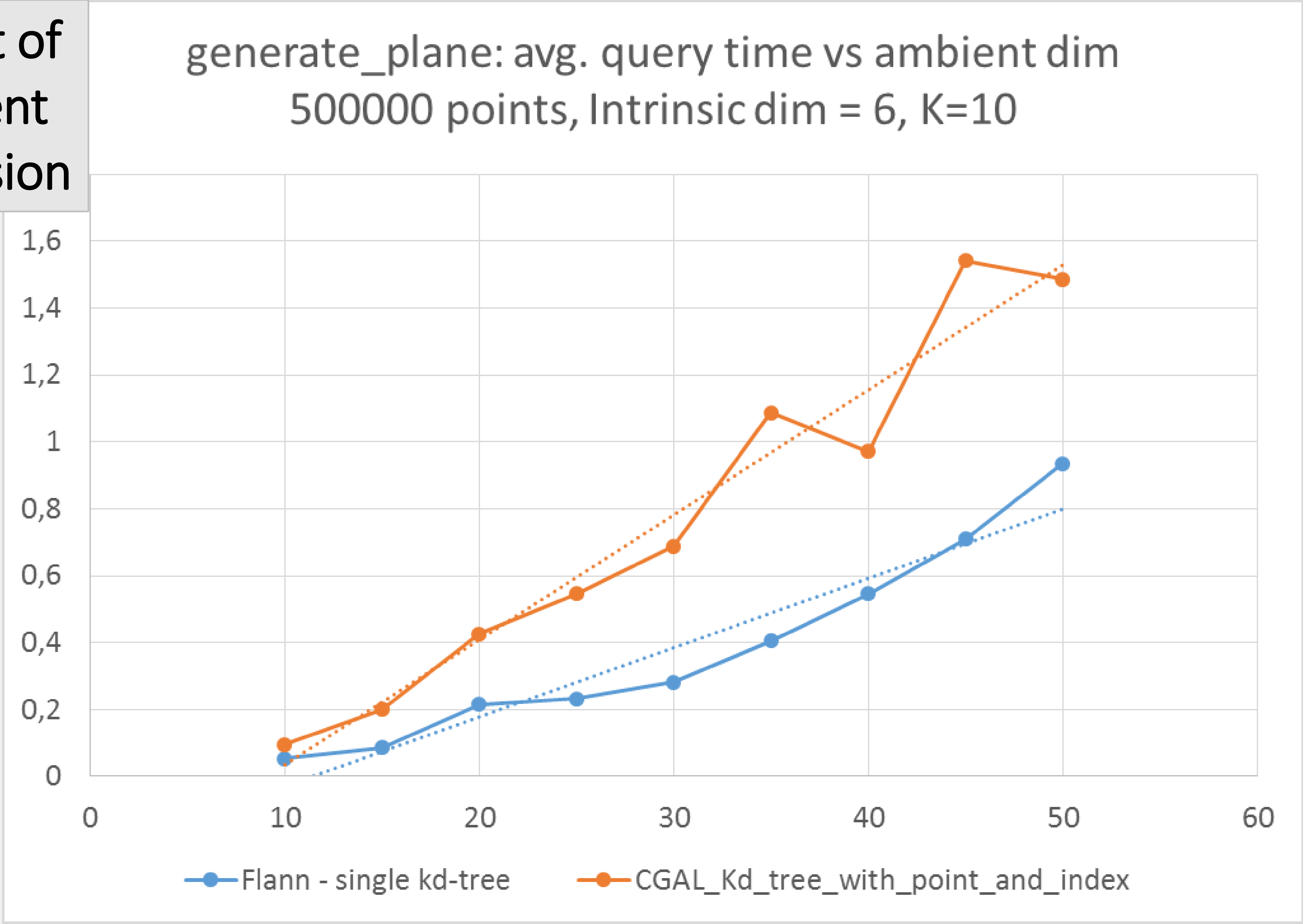
Impact of ambient dimension



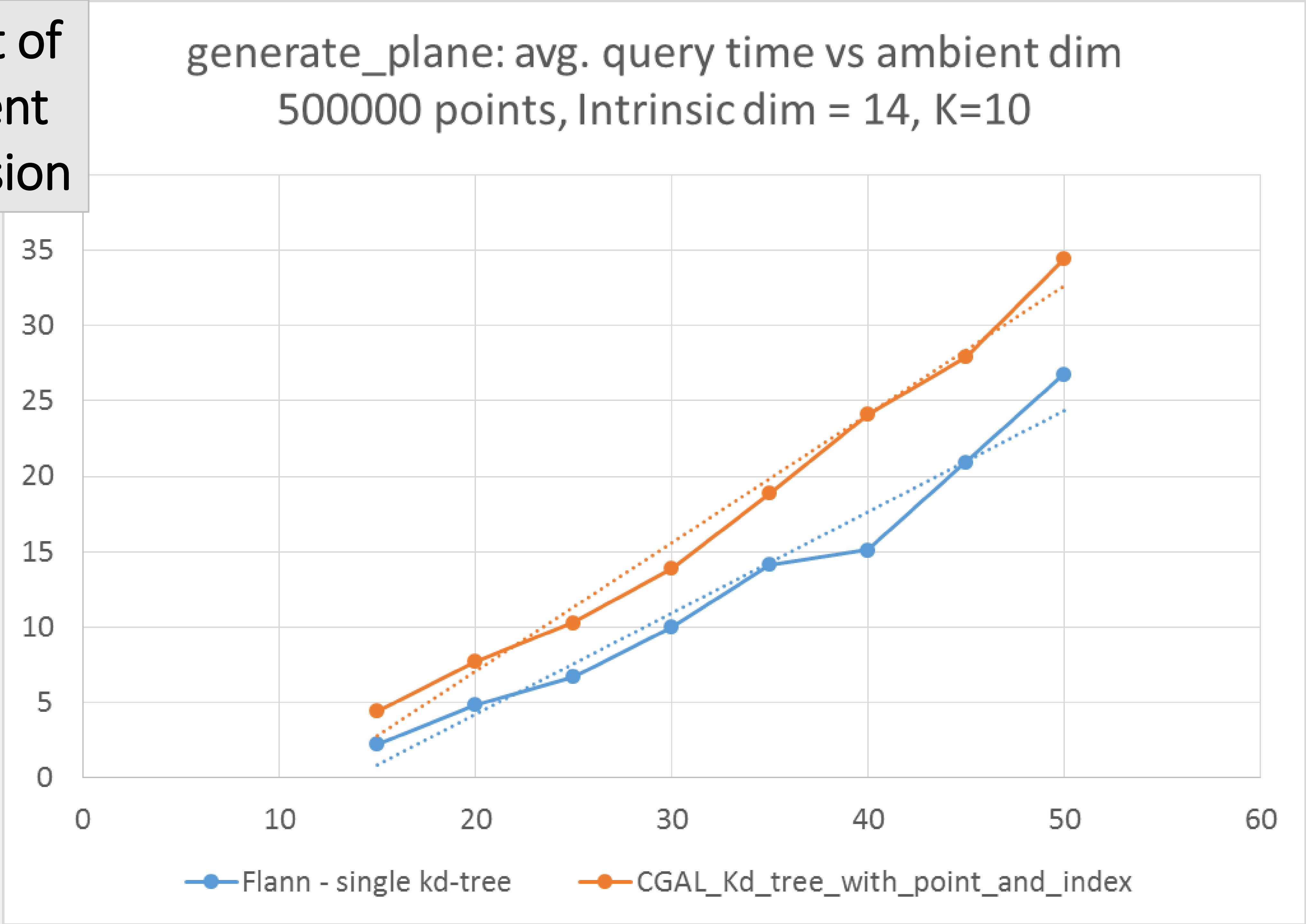
Impact of ambient dimension



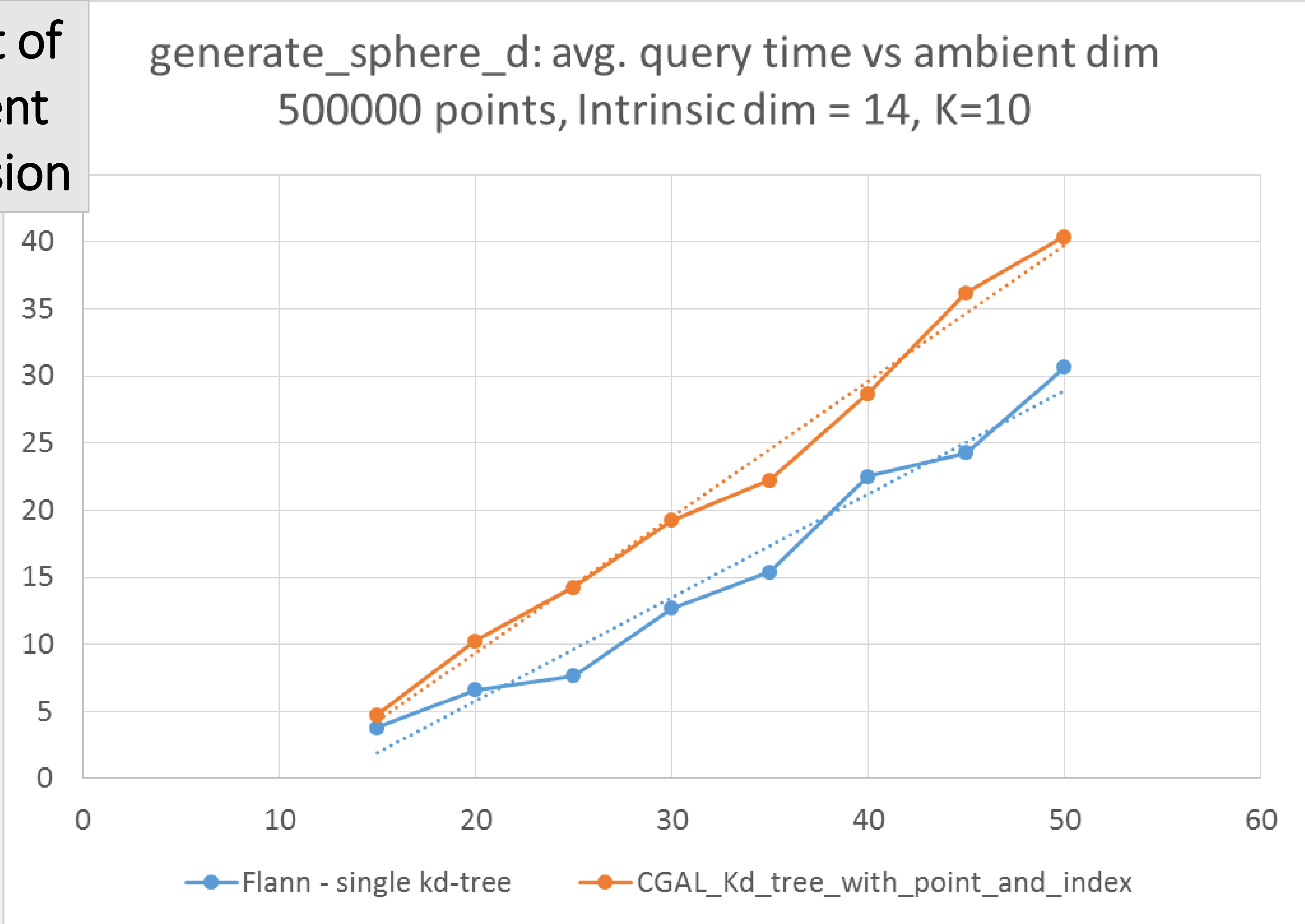
Impact of ambient dimension



Impact of ambient dimension



Impact of ambient dimension



Impact of the number of points

Conclusion

- Practical complexity of **kd-tree search**
  - **Exponential** in intrinsic dimension *d*
  - **Linear** in ambient dimension *D*
  - **Logarithmic** in the number of points *n*

- For GUDHI, we focus on:
  - **Low to medium** intrinsic dimension
  - **Medium to high** ambient dimension
  - **Exact** and **ϵ-approximated** searches

➔The *kd*-tree is a good candidate

Conclusion

- We need:
  - CGAL's genericity:
    - Custom data points
    - Several splitting techniques
  - Flann's speed

➔Short-term perspective:
optimize CGAL to match up with Flann's speed.

ευχαριστώ !