# Research paper summary:

Other anomaly detection methods work by creating a profile for normal form and data points that deviate from this are considered anomalies.   
Draw back from this is:

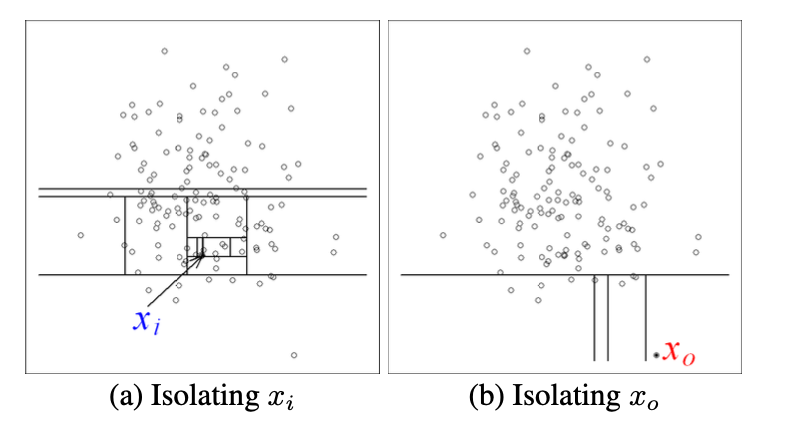
* *The anomaly detector is optimized to profile normal instances, but not optimized to detect anomalies—as a consequence, the results of anomaly detection might not be as good as expected, causing too many false alarms (having normal instances identified as anomalies) or too few anomalies being detected*
* *Many existing methods are constrained to low dimensional data and small data size because of their high computational complexity.*

Isolation forest takes advantage of 2 main properties of anomalies:

* *They are the minority consisting of fewer instances and*
* *They have attribute-values that are very different from those of normal instances.*

Because of these properties, anomalies are found closer to the root of a decision tree and normal data points are found deeper in the tree. Meaning it takes shorter steps to isolate an anomaly than a normal sample data.

In the image below, isolating Xo takes less steps than isolating Xi:



Isolation forest uses ensemble method, by creating multiple trees and anomalies are those whose average distance from the root of the tree are short. This leaves (**pun intended**) us with 2 parameters: number of trees and sub\_sampling size.   
The research paper aims to demonstrate that*—iForest’s detection performance converges quickly with a* ***very small number of trees,*** *and it only requires a* ***small sub-sampling size*** *to achieve high detection performance with high efficiency.*

In addition, it also has the following advantages compared to other anomaly detection methods:

* Since it is only interested in isolating anomalies, it can build partial decision trees because the deeper parts of the trees are not needed (they only show normal data points). This makes it possible to get good results with shallow trees.
* It doesn't require any distance measure to compute anomalies. That saves a substantial amount of computational time and cost.
* Linear time complexity and low memory usage.
* Ability to scale and handle large data and dimensions.

**Challenge:** The algorithm creates multiple trees using different sub-samples of the dataset. To consider a data point an anomaly, it should be isolated quickly, meaning it reaches a leaf after fewer splits from the root. To quantify this, it calculates the path length for each tree and finds the average path length. However, this distance — called path length **(h(x))** — is relative.

Because the algorithm could possibly create multiple trees of varying depth, in larger sub-sample the trees are deeper, and an anomaly might appear to have a long path. In smaller sub-sample, trees are shallow, so even normal points can appear to be isolated quickly. This makes direct comparison of h(x) across datasets misleading.

Ideally, we use path length h(x) — the number of edges from the root to the point — to decide how isolated a point is.

But here’s the problem: How do we normalize h(x) so it means the same thing regardless of how many data points/sub-sample size (n) we have?

We need to know if the data point was easily separated from other samples and we need to be able to compare that path distance across all trees. Using the depth of the tree (max depth) to normalize would bring a bias because an outlier could easily extend the depth of the tree and it wouldn’t communicate what the typical depth is for other data points *(N:B max depth reflects a worst-case path, not a typical experience in the tree)*.

Using the average depth is also sensitive to an outlier in the tree. Instead we need a value that communicates the typical depth of the tree and isn’t sensitive to outliers. Also note that the arithmetic average depth of a tree depends on the nature of the data and not the number of data points used in creating the tree. This also makes it empirically unreliable due to its unpredictable nature.

**Solution**: Since the iTree is similar to Binary Search Tree (BST), knowledge from it would be used to solve the problem. c(n) in BST calculates the average time it takes to get to an external node for a normal dataset (without outliers). This value varies for trees with different sub-sample size n in an unbiased manner, making it the best for normalizing the path length h(x). This way, a path of 3 in a small tree and a path of 3 in a large tree are interpreted comparatively to typical depths in their own trees (excluding outliers), because c(n) changes appropriately with n.

h(x) and c(n) are used to calculate s— an **anomaly score.**

s(x)= 2-(E(h(x)) / c(n))

Where:

* E(h(x)) is the average path length to isolate point x across all the trees.
* c(n) is the expected average path length for normal data points in a dataset of size n.

Anomaly Score S:

* If instances return anomaly score (s) very close to 1, then they are definitely anomalies.
* if instances have anomaly score much smaller than 0.5, then they are quite safe to be regarded as normal instances, and
* If all the instances return s ≈ 0.5, then the entire sample does not really have any distinct anomaly.

The isolation forest uses an anomaly threshold to classify data as anomaly or not. Meaning

If anomaly score >= threshold then -1 (anomaly) else 1 (normal).