**Algorithm 1** VP-Tree similarity search

**algorithm** search(Node *node*)

# global variables: Point *query*, int *k*, Maxheap *result*

# *node* is the current node, which contains a vantage *point* and a *radius*

# they establish a hypersphere that splits the whole space into an *inner* subtree and an *outer* subtree

# *result* stores accepted points at a specific moment, ordered by distance to the *query* point

# *result* has a maximum capacity of *k*, further insertion implies replacement

**if** *result* is empty

*t* := ∞

**else**

*t* := max distance in *result*

# *t* is the threshold that determines whether new entries are accepted into *result*

*d* := distance(*node*.*point*, *query*)

# *d* is the distance between the point the current node contains and the query point

*r* := *node*.*radius*

# *r* is the radius of the current node that separates the inner and outer subtree

**if** *d<=t*

add *node.point* to *results*

# determine the current point

**if** *d<=r+t*

search(*inner*)

**#** search the inner subtree if it overlaps with the query area

**if** *d>=r-t*

search(*outer*)

**#** search the outer subtree if it overlaps with the query area

**Algorithm 2** VP-Tree Concurrent Filtering

**algorithm** search(Node *node*)

# global variables: Point *query*, int *k*, Maxheap *result*

# *node* is the current node, which contains a vantage *point* and a *radius*

# they establish a hypersphere that splits the whole space into an *inner* subtree and an *outer* subtree

# *result* stores accepted points at a specific moment, ordered by distance to the *query* point

# *result* has a maximum capacity of *k*, further insertion implies replacement

**if** *result* is empty

*t* := ∞

**else**

*t* := max distance in *result*

# *t* is the threshold that determines whether new entries are accepted into *result*

*d* := distance(*node*.*point*, *query*)

# *d* is the distance between the point the current node contains and the query point

*r* := *node*.*radius*

# *r* is the radius of the current node that separates the inner and outer subtree

**if** *d<=t* **and** *node.point* fulfills attribute filtering criteria

add *node.point* to *results*

# determine the current point

**if** *d<=r+t*

search(*inner*)

**#** search the inner subtree if it overlaps with the query area

**if** *d>=r-t*

search(*outer*)

**#** search the outer subtree if it overlaps with the query area