Enum TraverseMethod


```
pub enum TraverseMethod<Node> {
    DepthFirst,
    BreadthFirst,
    BestFirst,
    Custom {
        cmp: Box<dyn Fn(&Node, &Node) -> Ordering>,
        cmp_superceeds_bound: bool,
    },
}
```

Order of traversing the subproblem tree with `solve`. See variants' docs for details.

Variants

`DepthFirst`

Depth-first search (DFS): descends into every subtree until reaches the leaf node (or determines that a subtree is not worth descending into because the boundary value is not better than the incumbent's objective score).

Nodes of the same layer will be processed in the order they are returned by the `Subproblem::branch_or_evaluate` method.

For typical boundary functions, uses significantly less memory compared to best-first and breadth-first search.

`BreadthFirst`

Breadth-first search (BFS): Traverses the subproblem tree layer by layer. The processing order among nodes on the same layer is unspecified.

For typical boundary functions, behaves similar to best-first search but uses a simpler internal data structure to store subproblems to be processed.

`BestFirst`

Best-first search (BeFS): traverses the tree in many directions simultaneously, on every iteration selects and evaluates the subproblem with the best value of the boundary function. All its children become candidates for the next selection (as long as their boundary value is better than the incumbent's objective score).

The processing order among subproblems with the same boundary value is unspecified.

For typical boundary functions, behaves similar to breadth-first search but selects subproblems more optimally.

`Custom`

Like best-first search but selects subproblems in the custom order, based on the given comparator `.cmp`.

Processes subproblems in the order specified by `.cmp`: subproblems that compare *greater* are processed *first*! The processing order among subproblems that compare equal is unspecified.

The processing order among nodes that compare equal according to `.cmp` is unspecified.

Set `.cmp_superceeds_bound` to `true` only if `.cmp` guarantees that

if `cmp(subproblem_a, subproblem_b) == Ordering::Less`

then `subproblem_a.bound() < subproblem_b.bound()`

(in other words, the order defined by `.cmp` is a specialized order / super-order with respect to the order defined by `Subproblem::bound`).

If `.cmp_superceeds_bound` is set, the search will terminate as soon as the candidate that is best according to `.cmp` has the boundary value less (i.e., worse) than that of the current incumbent.

Fields

```
cmp: Box<dyn Fn(&Node, &Node) -> Ordering>
cmp_superceeds_bound: bool
```

Auto Trait Implementations

```
impl<Node> Freeze for TraverseMethod<Node>
impl<Node> !RefUnwindSafe for TraverseMethod<Node>
impl<Node> !Send for TraverseMethod<Node>
impl<Node> !Sync for TraverseMethod<Node>
impl<Node> Unpin for TraverseMethod<Node>
impl<Node> !UnwindSafe for TraverseMethod<Node>
```

Blanket Implementations

```
impl<T> Any for T
where
    T: 'static + ?Sized,
impl<T> Borrow<T> for T
```

where
    T: ?Sized,

impl<T> BorrowMut<T> for T
where
    T: ?Sized,

impl<T> From<T> for T

impl<T, U> Into<U> for T
where
    U: From<T>,

impl<T, U> TryFrom<U> for T
where
    U: Into<T>,

impl<T, U> TryInto<U> for T
where
    U: TryFrom<T>,