branch_and_bound

Trait Subproblem

```
pub trait Subproblem {
    type Score: Ord;

    // Required methods
    fn branch_or_evaluate(&mut self) -> SubproblemResolution<Se

    fn bound(&self) -> Self::Score;
}
```

A problem (subproblem) to be solved with branch-and-bound

Required Associated Types

`type Score: Ord`

Return type of the boundary and the objective function. Higher score is better.

Required Methods

`fn branch_or_evaluate(&mut self) -> SubproblemResolution<Self, Self::Score>`

Evaluates the subproblem space.

If the space is to be broken further into subproblems, returns a sequence of subproblems (may be empty, which discards the current subspace).

If the space consists of just one feasible solution to be solved directly, returns the score, which is the value of the objective function at the solution. The node is then considered a successful candidate.

The method may mutate `self` as follows:

- If `SubproblemResolution::Branched` is returned, the library shall discard the object after that, so any changes to `self` are allowed, even if after the changes it no longer represents the original subproblem;
- If `SubproblemResolution::Solved` is returned, the library will use the subproblem object as a successful candidate, so mutations to the internal state are allowed, as long as `self` continues to represent the same subproblem.

```
fn bound(&self) -> Self::Score
```

Value of the boundary function at the subproblem space.

The boundary function gives an upper-boundary of the best solution that could potentially be found in this subproblem space. The value of the boundary function must be greater than or equal to every value of the objective score of any subproblem reachable through consecutive `.branch_or_evaluate` calls.

If at some point in the search process a subproblem's `.bound()` value is less than or equal to the current best solution, the subproblem is discarded (because no better solution will be found in its subtree).

Dyn Compatibility

This trait is **not** dyn compatible.

In older versions of Rust, dyn compatibility was called "object safety", so this trait is not object safe.

Implementors