

Tabu Search in HL2.0

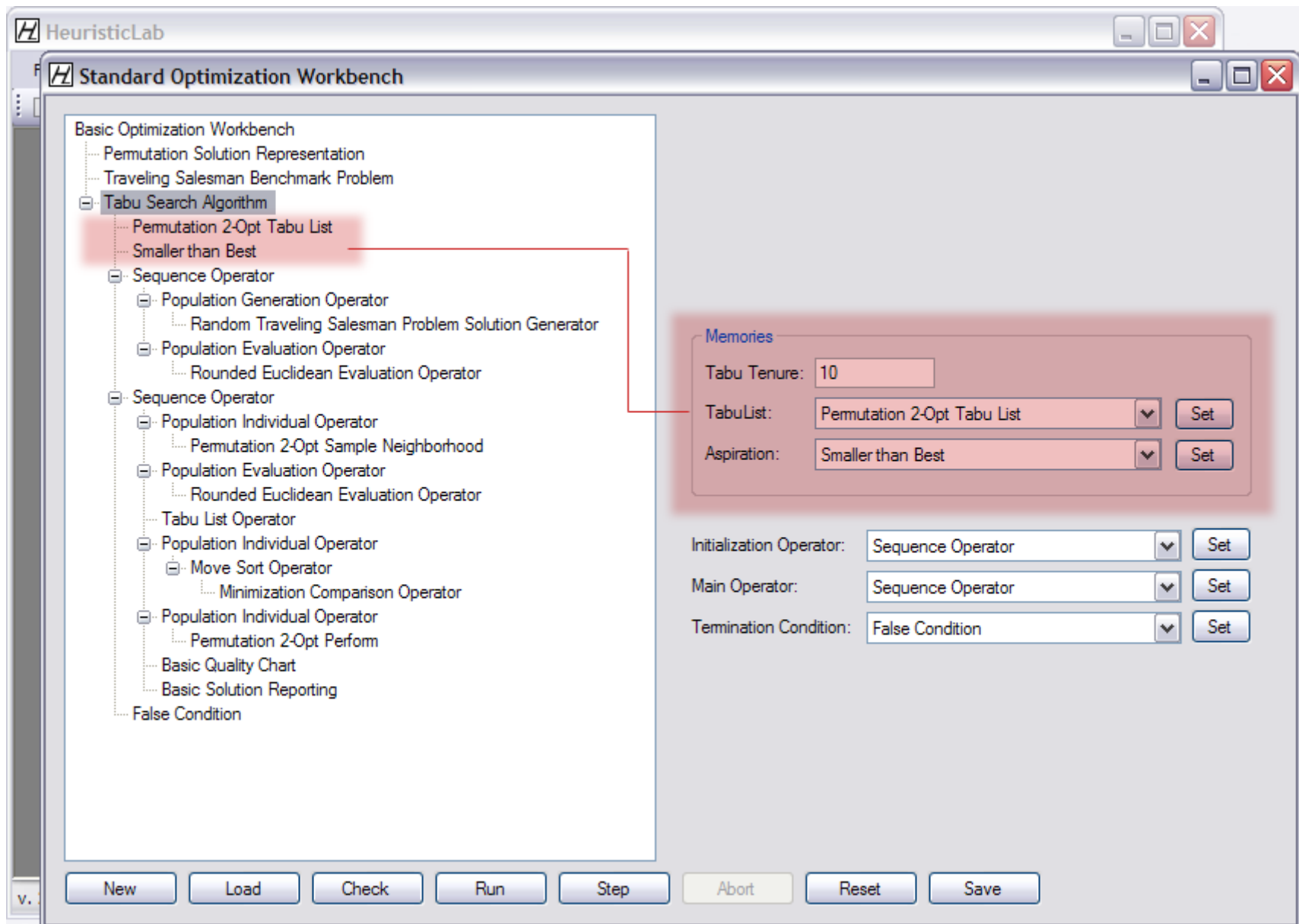
Concepts

- Trajectory based search
 - Definition of Neighborhood
 - Moves
- Memory
 - Tabu List
 - Aspiration criteria
 - Intermediate or Long Term Memories

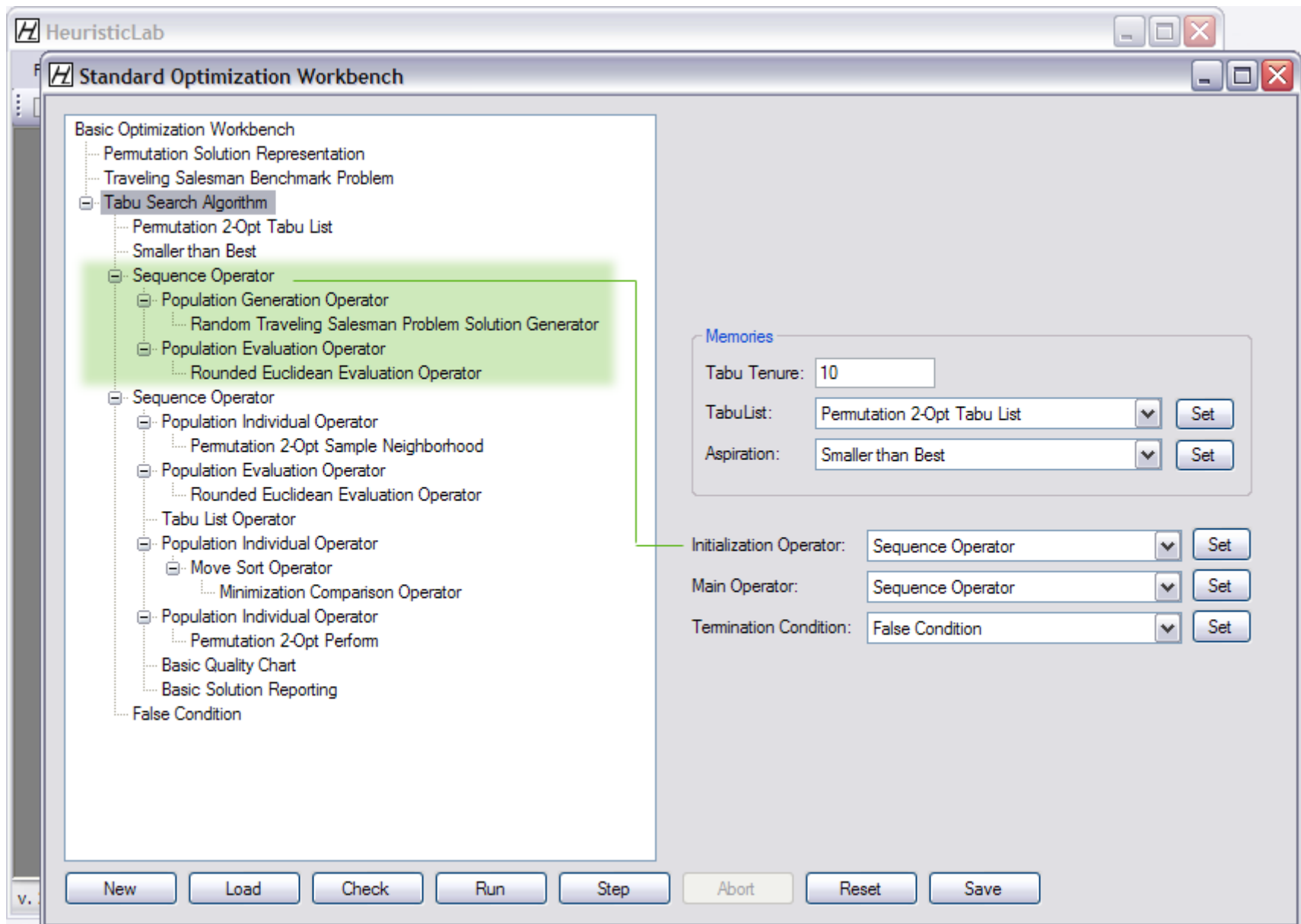
Tabu Search Code

```
protected override void ExecuteStep() {  
    if (initialize) {  
        if (InitializationOperator != null) solutionsCollection = InitializationOperator.Apply(solutionsCollection);  
        initialize = false;  
    }  
    for (int i = 0 ; i < solutionsCollection.Length ; i++) {  
        Aspiration.Learn(solutionsCollection[i].Quality, null);  
    }  
    if (MainOperator != null) solutionsCollection = MainOperator.Apply(solutionsCollection);  
    if (TerminationCondition != null) Terminated = TerminationCondition.Check();  
    if (Terminated) {  
        TabuList.Reset();  
        Aspiration.Reset();  
    }  
}
```

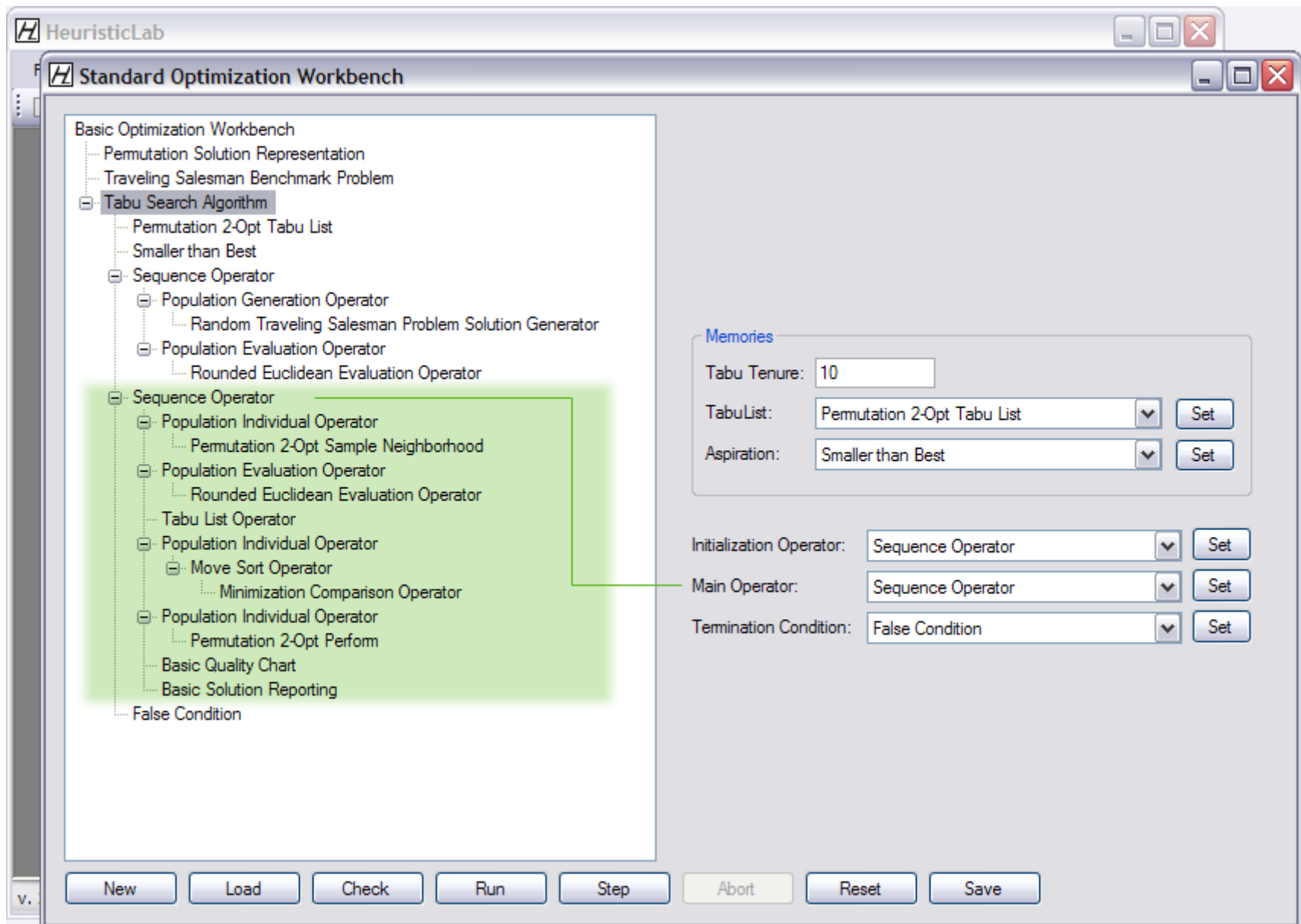
User Interface



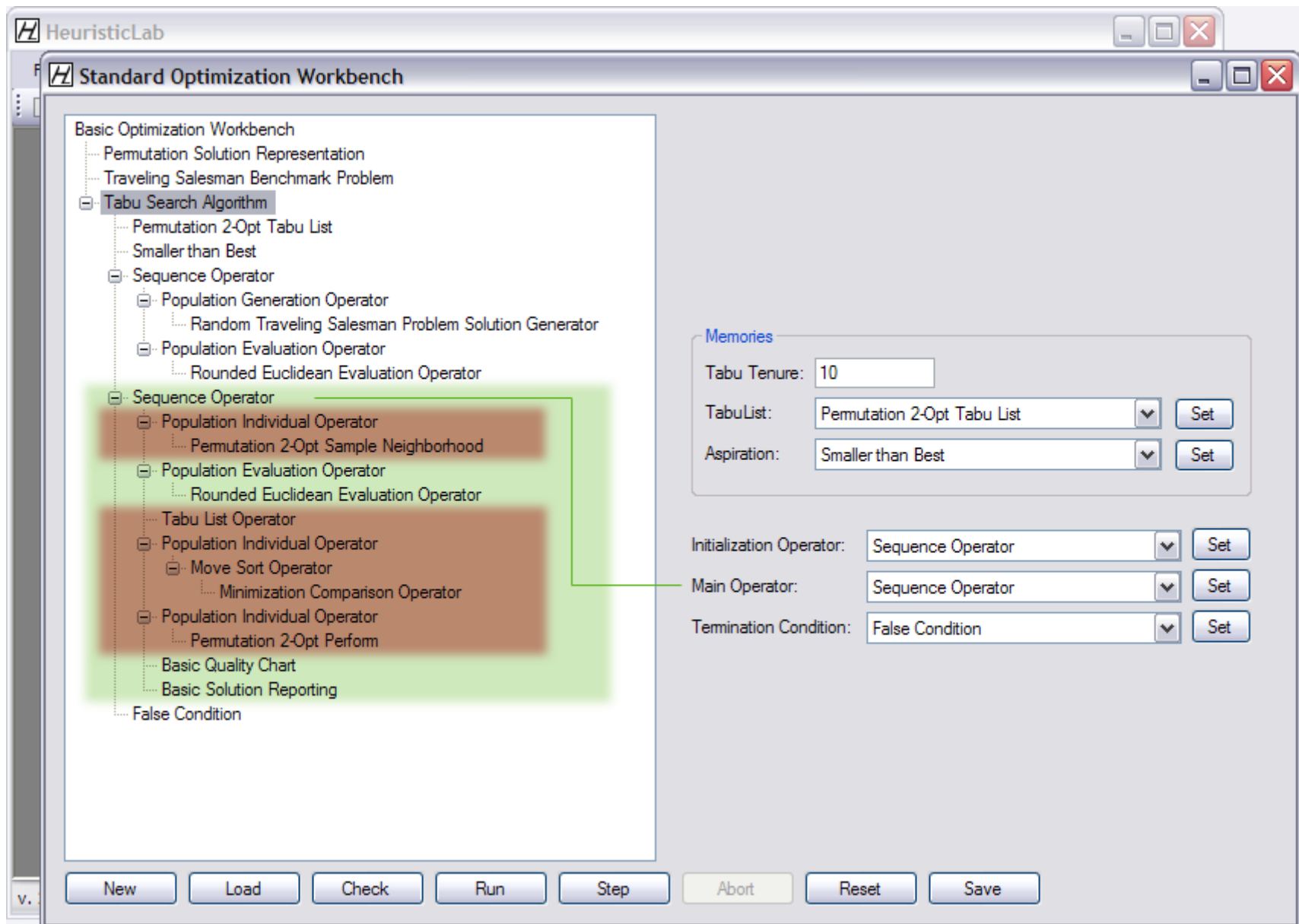
Memories are held within the algorithm plugin



No surprise: Initialization phase seems familiar



Main Operator

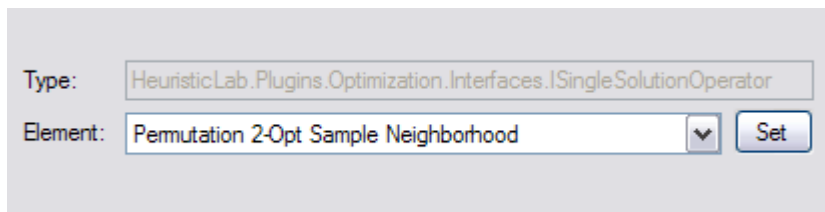


Several new Operators; Will be explained in more detail

Code Insights

New Container Operator

- PopulationIndividualOperator
 - Similar to a PopulationProcessingOperator
 - but has return type: **ISolution** Apply(**ISolution**)
 - instead of: Apply(**ISolution**)
 - Has an embedded operator „Operator“ of type **ISingleSolutionOperator**
 - Loops over all solutions in a collection



The image shows a configuration dialog box with a light gray background. It contains two rows of controls. The first row is labeled "Type:" and has a text input field containing the full namespace path: "HeuristicLab.Plugins.Optimization.Interfaces.ISingleSolutionOperator". The second row is labeled "Element:" and has a dropdown menu with the text "Permutation 2-Opt Sample Neighborhood" and a small downward arrow icon. To the right of the dropdown is a button labeled "Set".

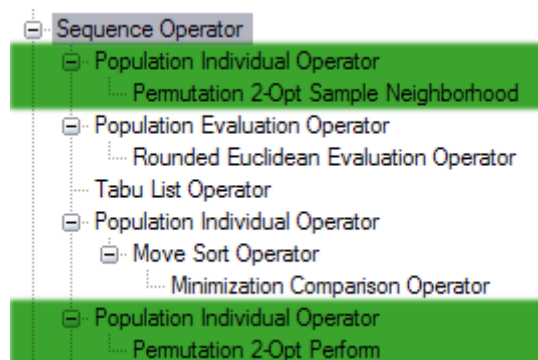
ISingleSolutionOperator

```
namespace HeuristicLab.Plugins.Optimization.Interfaces {  
    public interface ISingleSolutionOperator : IOptimizationElement {  
        ISolution Apply(ISolution solution);  
    }  
}
```

- Takes a solution
- Returns a solution (same or different)
- Similar to ISolution(Processing|Manipulation|Evaluation)Operator

Neighborhood Operator

- Split into generation of moves and application of a single move
- Two new base classes in Optimization.Core
 - MoveNeighborhoodOperatorBase
 - MovePerformOperatorBase
- Both implement **ISingleSolutionOperator**



Generate Neighborhood

- Sample or Full Neighborhood
 - Full Neighborhood is generated first time only
 - Sample Neighborhood != sample of Full Neighborhood, but exploits a wider range of possibilities
- Solution object changes(!!) e.g.
PermutationSolution -> PermutationTSSolution
 - =>PopulationProcessingOperator does not work

ITSSolution and IMove

- ITSSolution specifies that the implementing class carries a collection of moves

```
namespace HeuristicLab.Plugins.Optimization.Interfaces {  
    public interface ITSSolution {  
        ArrayList Moves { get; set; }  
    }  
}
```

- IMove

```
namespace HeuristicLab.Plugins.Optimization.Interfaces {  
    public interface IMove {  
        double Estimate { get; set; }  
        double[] Estimates { get; set; }  
        double Quality { get; set; }  
        double[] Qualities { get; set; }  
        bool UseEstimate { get; set; }  
        String MoveType { get; set; }  
  
        IMove Copy();  
        IValuesContainer GetValues(string id);  
    }  
}
```

Partial Reevaluation

- In the evaluation operator:


```
switch (move.MoveType) {  
  case "PermutationSwap2": {  
    PermutationSwap2Move m = (move as PermutationSwap2Move);  
    int i1 = Math.Min(m.Index1, m.Index2), i2 = Math.Max(m.Index1, m.Index2);  
    int idx1 = tour[i1], idx2 = tour[i2];  
    int prevIdx1 = tour[((i1 == 0) ? (tour.Length - 1) : (i1 - 1))], nextIdx1 = tour[(i1 + 1) % tour.Length];  
    int prevIdx2 = tour[((i2 == 0) ? (tour.Length - 1) : (i2 - 1))], nextIdx2 = tour[(i2 + 1) % tour.Length];  
    if (distanceMatrix != null) {  
      if (i2 - i1 < tour.Length - 1) {  
        quality -= distanceMatrix[prevIdx1, idx1];  
        quality += distanceMatrix[prevIdx1, idx2];  
        quality -= distanceMatrix[idx2, nextIdx2];  
        quality += distanceMatrix[idx1, nextIdx2];  
      }  
      if (i2 - i1 > 1) {  
        quality -= distanceMatrix[idx1, nextIdx1];  
        quality += distanceMatrix[idx2, nextIdx1];  
        quality -= distanceMatrix[prevIdx2, idx2];  
        quality += distanceMatrix[prevIdx2, idx1];  
      }  
    }  
  }  
}
```

- not necessary to reevaluate the whole tour

Perform Move

- Selects the first move in the Neighborhood
 - Sorting can influence choice of move
- Performs this move on the solution
- Updates the TabuList with the chosen Move
 - Needs reference to the TabuList!



Referenced Parameters and Status:

.....  Tabu List ([Tabu Search Algorithm].TabuList)

TabuList Operator

- Looks through the moves and deletes those that are tabu
- Solutions pass through with the same or a reduced collection of moves
 - in the case there is no move, the operator works on the solution objects
- Needs reference to TabuList
- Needs reference to Aspiration criterium

Referenced Parameters and Status:

	Tabu List ([Tabu Search Algorithm].TabuList)
	Aspiration ([Tabu Search Algorithm].Aspiration)

Memories

- New generalized interface: **IMemoryOperator**

```
namespace HeuristicLab.Plugins.Optimization.Interfaces {  
    public interface IMemoryOperator : IOptimizationElement {  
        int Size { get; set; } // size of the memory  
        int Count { get; } // number of elements currently in the memory  
        bool Learn(object action, object situation); // learn a pair of action and corresponding situation  
        object Predict(object action); // predict a situation from an action  
        object Infer(object situation); // infer an action given a situation  
        void Reset(); // resets/initializes the memory  
    }  
}
```

- Four methods:
 - Learn, Predict, Infer and Reset

TabuList

- Is of type `IMemoryOperator`
- Implemented for each Neighborhood operation
- Learn = `SetTabu(object move, object solution)`
- Predict = `IsTabu`
- Infer is unused, Reset is clear

Aspiration

- Is of type `IMemoryOperator`
- Learn teaches the memory a new fitness value
- Predict asks if the current value satisfies some criterion against the learnt value

Changes to the previous version

- Memories are stored in the algorithm operator
 - Lifetime of a memory usually bound to the lifetime of the algorithm
 - Central place for each operator to access to (i.e. shared resource)
- Separate Neighborhood operators for generating and performing
- Supports partial solution reevaluation
- More possibilities, but also more effort to implement new operators required

Demo

Questions?