DM841

Discrete Optimization

EasyLocal

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

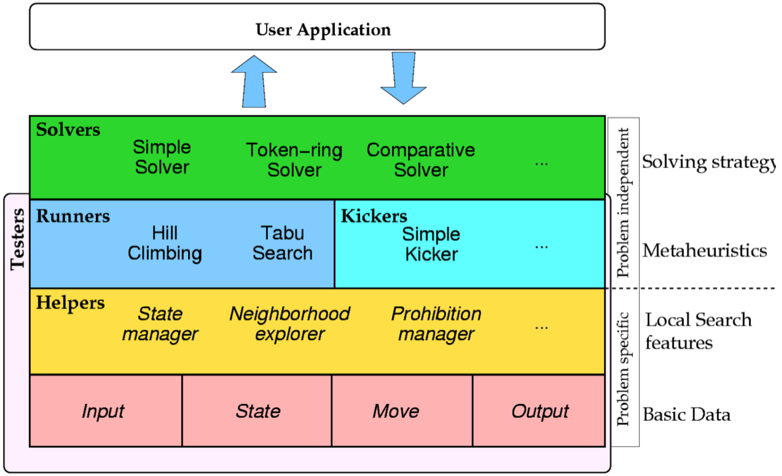
Framework

Framework set of abstract classes used by inheritance and definition of methods. It gives indication about where to put everything.
Like a library. But instead of calling it, it calls your methods.

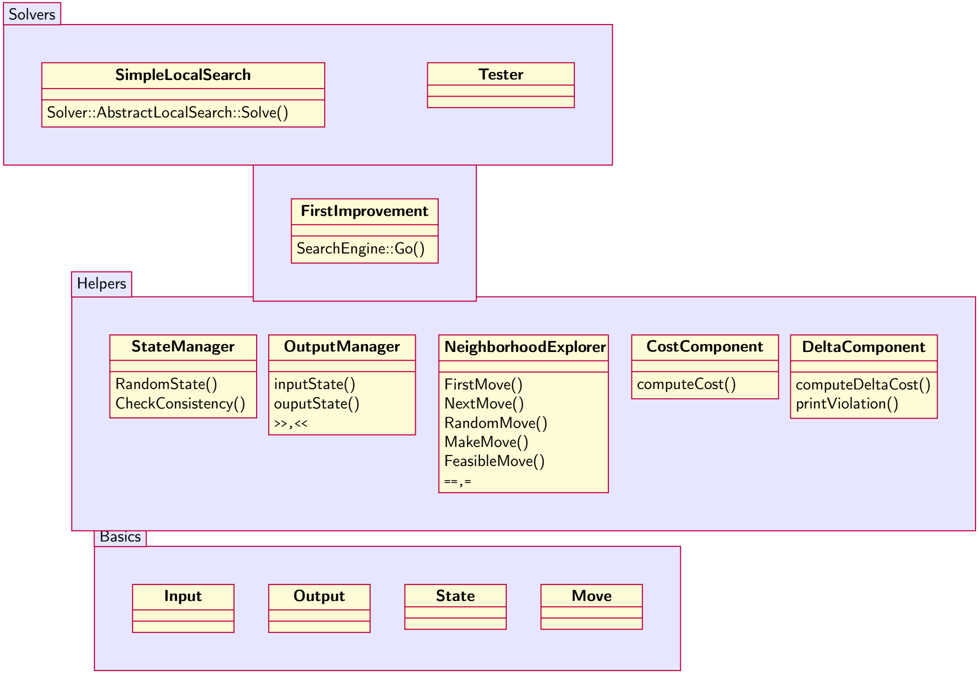
- Pure virtual methods are called hot spots.
- Warm spots (keep or redefine), virtual functions
- Cold spots are those already defined
  Hollywood principle: don't call us, we call you.

Outline

The Framework



https://bitbucket.org/satt/easylocal-3/src/master/


Solvers
SimpleLocalSearch
Solver::AbstractLocalSearch::Solve()
Tester
FirstImprovement
SearchEngine::Go()
Helpers
StateManager
RandomState()
CheckConsistency()
OutputManager
inputState()
ouputState()
>>,<<
NeighborhoodExplorer
FirstMove()
NextMove()
RandomMove()
MakeMove()
FeasibleMove()
==,=
CostComponent
computeCost()
DeltaComponent
computeDeltaCost()
printViolation()
Basics
Input
Output
State
Move

C++: Standard Template Library

- Static arrays `array<type>`
- Dynamic arrays `vector<type>`
- lists (no random access) `list<type>`
- sets (no repetition of elements allowed) `set<type>` (implemented as red-black trees)
- maps `map<keyttype, type>` associative containers that contain key-value pairs with unique keys. Keys are sorted. (similar to dictionaries in python) (implemented as red-black trees)
- unordered versions of sets and maps
- They require to include the std library:

```
#include<cstdlib>
#include<vector>
#include<list>
#include<map>
#include<set>
#include<algorithm>
#include<stdexcept>
using namespace std;
```

Iterators

- iterators are pointers to elements of STL containers

```
vector<int> A = {1,2,3,4};
vector<int>::iterator pt; //  or vector<int>::const_iterator
for (pt=A.begin(); pt!=A.end(); pt++)
  cout<<*pt;
```

- Type inference:

```
vector<int> A = {1,2,3,4};
vector<int>::iterator pt1 = A.begin();
auto pt2 = A.begin();
```

- for syntax:

```
for (auto &x : my_array) {
   x *= 2;
}
```

Outline

Solver::Solve()

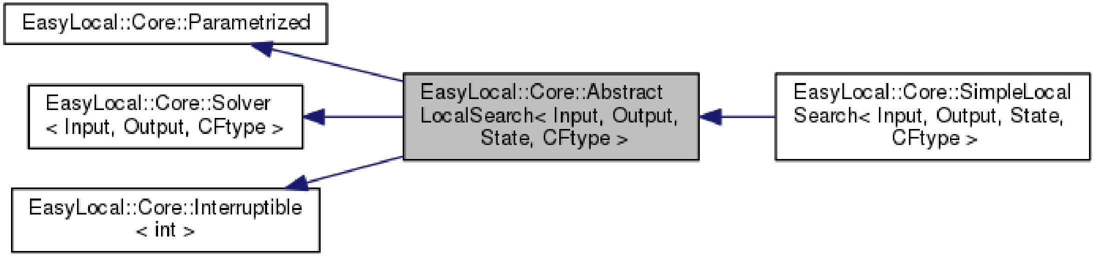
In `solver/abstractlocalsearch.hh`

```
template<class Input, class Output, class State, typename CFtype>
SolverResult<Input, Output, CFtype> AbstractLocalSearch<Input, Output, State, CFtype>::Solve() throw (ParameterNotSet, IncorrectParameter
    auto start = std::chrono::high_resolution_clock::now();
    InitializeSolve();
    FindInitialState();
    if (timeout.IsSet()) {
        SyncRun(std::chrono::milliseconds(static_cast<long long int>(timeout * 1000.0)));
    } else
        Go();
    p_out = std::make_shared < Output > (this->in);
    om.OutputState(*p_best_state, *p_out);
    TerminateSolve();

    double run_time = std::chrono::duration_cast < std::chrono::duration<double, std::ratio<1>>>(std::chrono::high_resolution_clock::now(

    return SolverResult<Input, Output, CFtype>(*p_out, sm.CostFunctionComponents(*p_best_state), run_time);
}
```

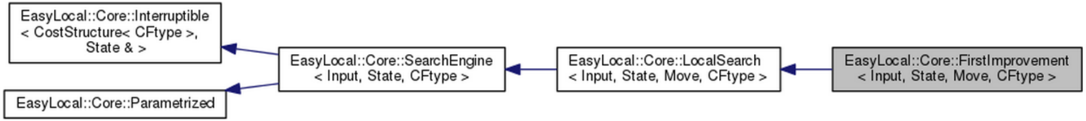
Inheritance Diagram


EasyLocal::Core::Parametrized
EasyLocal::Core::Solver < Input, Output, CFtype >
EasyLocal::Core::Interruptible < int >
EasyLocal::Core::AbstractLocalSearch< Input, Output, State, CFtype >
EasyLocal::Core::SimpleLocalSearch< Input, Output, State, CFtype >

LS Framework: SearchEngine

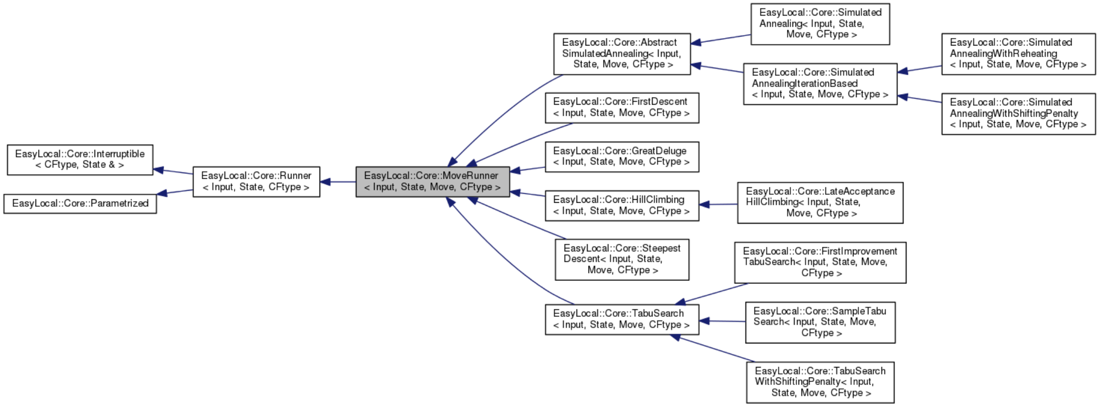
- SearchEngine classes are the algorithmic core of the framework.
- They are responsible for performing a run of a local search technique, starting from an initial state and leading to a final one.
- `SearchEngine` has only `Input` and `State` templates, and is connected to the solvers
- `LocalSearch` has also `Move`, and the pointers to the necessary helpers. It also stores the basic data common to all derived classes:
  - current state,
  - best state,
  - current move,
  - number of iterations.

Inheritance Diagram


EasyLocal::Core::Interruptible < CostStructure< CFtype >, State & >
EasyLocal::Core::Parametrized
EasyLocal::Core::SearchEngine < Input, State, CFtype >
EasyLocal::Core::LocalSearch < Input, State, Move, CFtype >
EasyLocal::Core::FirstImprovement < Input, State, Move, CFtype >

Inheritance Diagram

A potential development for local search engines
(here `Runner=SearchEngine`)


EasyLocal::Core::Interruptible < CFtype, State & >
EasyLocal::Core::Parametrized
EasyLocal::Core::Runner < Input, State, CFtype >
EasyLocal::Core::MoveRunner < Input, State, Move, CFtype >
EasyLocal::Core::Abstract SimulatedAnnealing< Input, State, Move, CFtype >
EasyLocal::Core::Simulated Annealing< Input, State, Move, CFtype >
EasyLocal::Core::Simulated AnnealingIterationBased < Input, State, Move, CFtype >
EasyLocal::Core::Simulated AnnealingWithReheating < Input, State, Move, CFtype >
EasyLocal::Core::Simulated AnnealingWithShiftingPenalty < Input, State, Move, CFtype >
EasyLocal::Core::FirstDescent < Input, State, Move, CFtype >
EasyLocal::Core::GreatDeluge < Input, State, Move, CFtype >
EasyLocal::Core::HillClimbing < Input, State, Move, CFtype >
EasyLocal::Core::LateAcceptance HillClimbing< Input, State, Move, CFtype >
EasyLocal::Core::Steepest Descent< Input, State, Move, CFtype >
EasyLocal::Core::TabuSearch < Input, State, Move, CFtype >
EasyLocal::Core::FirstImprovement TabuSearch< Input, State, Move, CFtype >
EasyLocal::Core::SampleTabu Search< Input, State, Move, CFtype >
EasyLocal::Core::TabuSearch WithShiftingPenalty< Input, State, Move, CFtype >

SearchEngine::Go()

In `SearchEngine.hh`

```
template <class Input, class State, typename CFtype>
CostStructure<CFtype> SearchEngine<Input, State, CFtype>::Go(State& s) throw (ParameterNotSet, IncorrectParameterValue)
{
  // std::shared_ptr<State> p_current_state;
  // std::shared_ptr<State> p_best_state;
  // state s is only used for input and output
  InitializeRun(s); // in searchengine.hh, calls InitializeRun() in localsearch.hh          (START)
  while (!MaxEvaluationsExpired() && !StopCriterion() && !LowerBoundReached() && !this->TimeoutExpired())
  {
    PrepareIteration();
    try
    {
      SelectMove(); // <== in firstimprovement.hh
      if (AcceptableMoveFound()) // <== in localsearch.hh
      {
        PrepareMove(); // does nothing but virtual
        MakeMove(); // in localsearch.hh where it calls MakeMove from NeighborhoodManager (MADE_MOVE)
        CompleteMove(); // does nothing but virtual
        UpdateBestState(); // in localsearch.hh                                             (NEW_BEST)
      }
    }
    catch (EmptyNeighborhood)
    {
      break;
    }
    CompleteIteration(); // does nothing but virtual
  }
  return TerminateRun(s); // in searchengine.hh, calls InitializeRun() in localsearch.hh     (END)
}
```

First Improvement in EasyLocal

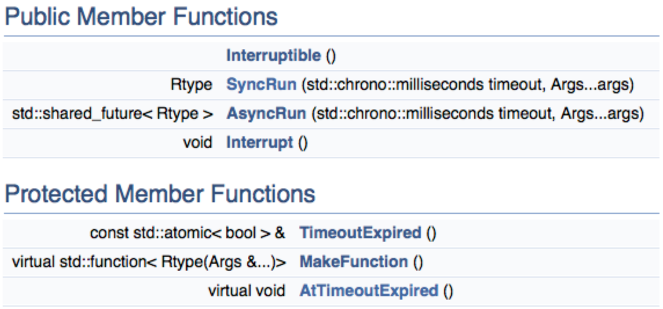
Definition of

- StopCriterion
- SelectMove

Interruptible

An inheritable class to add timeouts (in milliseconds) to anything.

`MakeFunction` produces a function object to be launched in a separate thread by SyncRun, AsyncRun or Tester



Public Member Functions

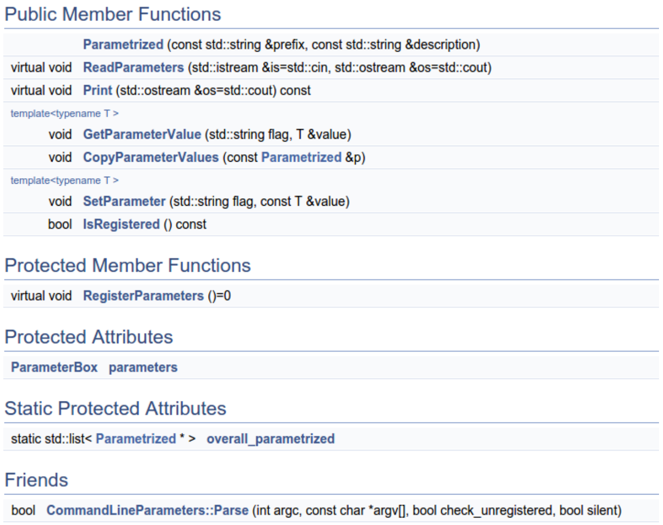
	Interruptible ()
Rtype	**SyncRun** (std::chrono::milliseconds timeout, Args...args)
std::shared_future< Rtype >	**AsyncRun** (std::chrono::milliseconds timeout, Args...args)
void	**Interrupt** ()

Protected Member Functions

const std::atomic< bool > &	**TimeoutExpired** ()
virtual std::function< Rtype(Args &...)>	**MakeFunction** ()
virtual void	**AtTimeoutExpired** ()

Parametrized

An inheritable class representing a parametrized component.



Public Member Functions

	Parametrized (const std::string &prefix, const std::string &description)
virtual void	ReadParameters (std::istream &is=std::cin, std::ostream &os=std::cout)
virtual void	Print (std::ostream &os=std::cout) const
template<typename T > void	GetParameterValue (std::string flag, T &value)
void	CopyParameterValues (const Parametrized &p)
template<typename T > void	SetParameter (std::string flag, const T &value)
bool	IsRegistered () const

Protected Member Functions

virtual void	RegisterParameters ()=0

Protected Attributes

ParameterBox	parameters

Static Protected Attributes

static std::list< Parametrized * >	overall_parametrized

Friends

bool	CommandLineParameters::Parse (int argc, const char *argv[], bool check_unregistered, bool silent)

In constructors, eg. `AbstractLocalSearch`

Observers

Infrastructure for printing debugging information on the runner
The command line parameter decides how much verbose the output must be:

- `--main::observer 1` for all runners with the observer attached, it writes some info on the costs everytime the runner finds a new best state.
- `--main::observer 2` it writes also all times that the runner makes a worsening move
- `--main::observer 3`, it write all moves executed by the runner.

C++: Lambda functions (aka Closures)

- A function that can be written inline in source code to pass to another function
- A tutorial: http://www.cprogramming.com/c++11/c++11-lambda-closures.html

```
auto func = [] () { cout << "Hello world"; };
func(); // now call the function
```

```
vector<int> v {1, 2};
for_each( v.begin(), v.end(), [] (int val) {  cout << val; } );
```

- `[a,&b]` where a is captured by value and b is captured by reference.
- `[this]` captures the this pointer by value
- `[&]` captures all variables in the body of the lambda by reference
- `[=]` captures all variables in the body of the lambda by value
- `[]` captures nothing

```
[] () { return 1; } // compiler knows this returns an integer
[] () -> int { return 1; } // now we're telling the compiler what we want
```