How to Design C++ Implementations of Complex Combinatorial Algorithms

Piotr Wygocki
paal.mimuw.edu.pl

May 16, 2014

Plan

Local Search

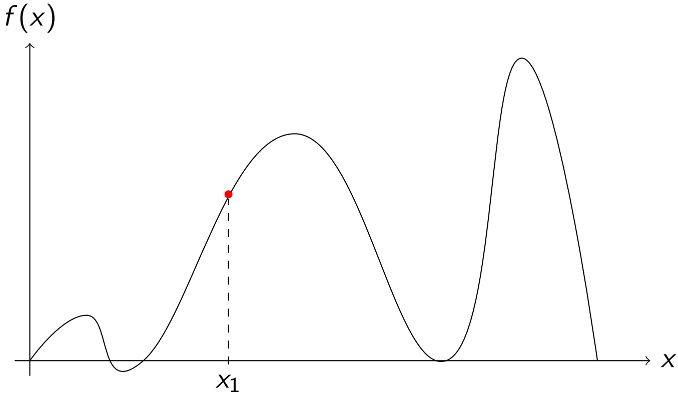
Design

More specific usage

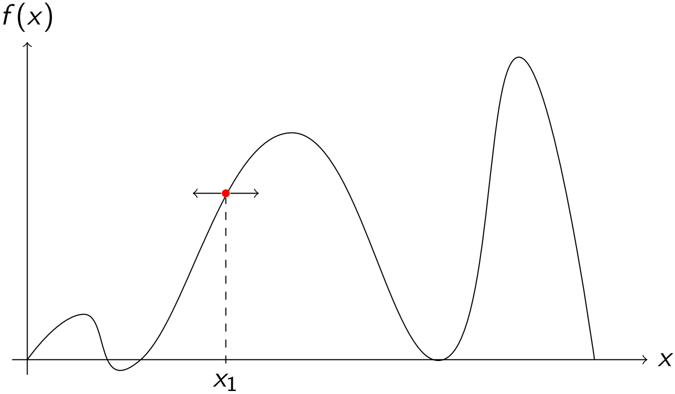
Implementation

Summary

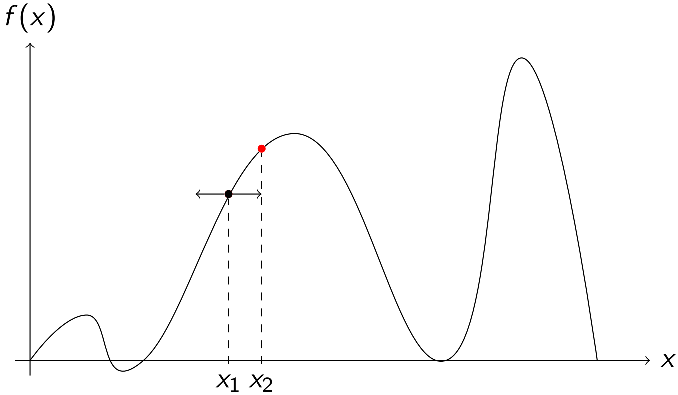
local search



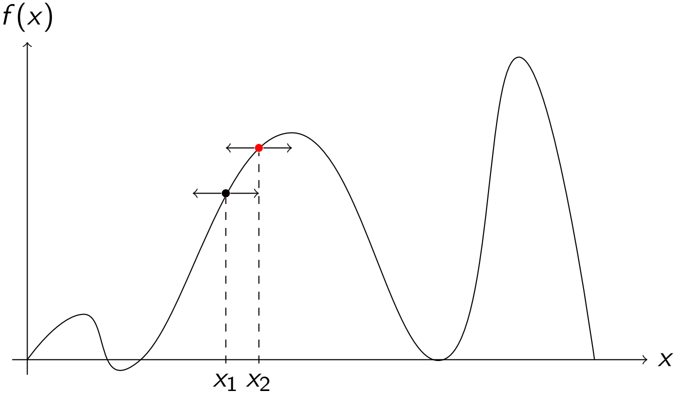
local search



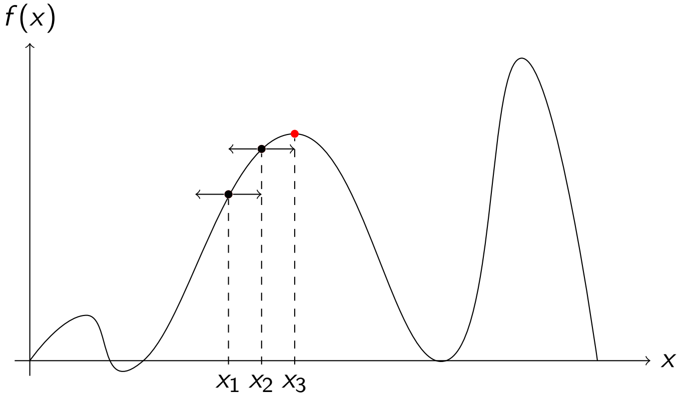
local search



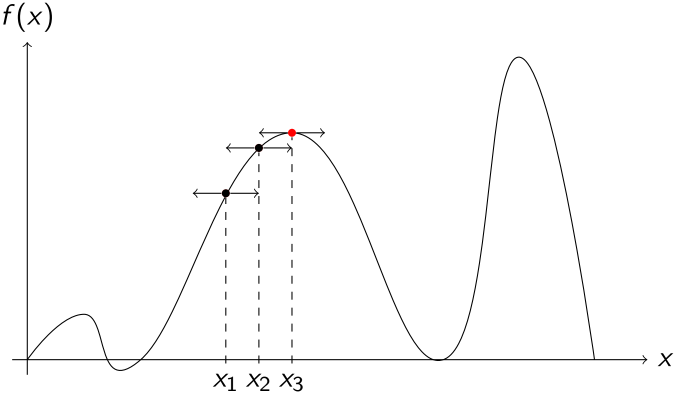
local search



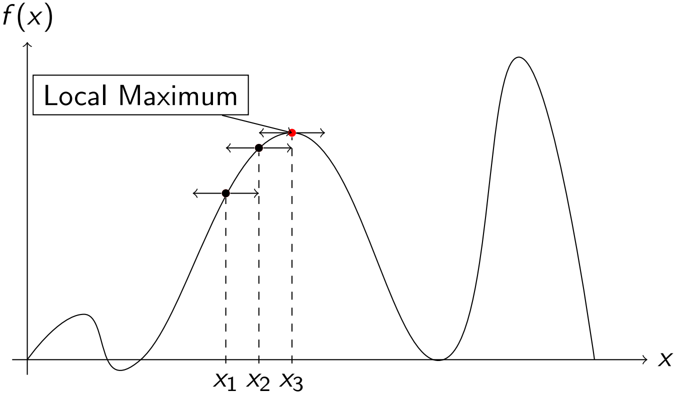
local search



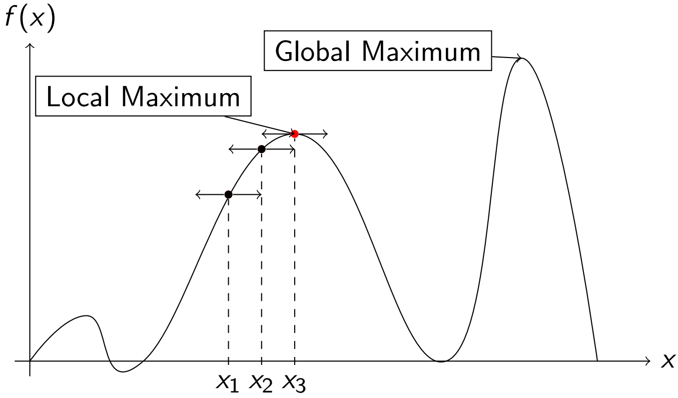
local search



local search



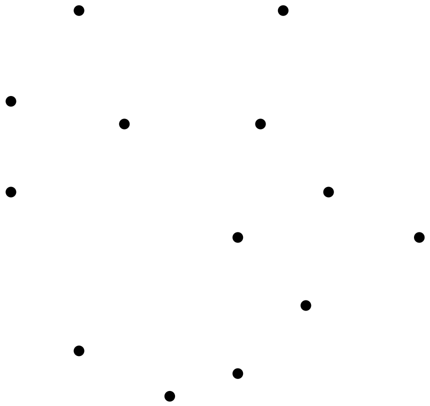
local search



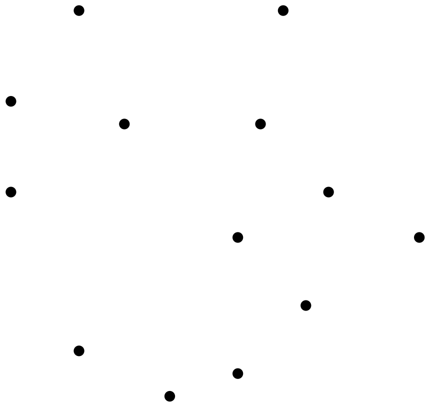
pseudocode

```
local_search(x)
{
   do {
      for_each(Move m in Neighborhood(x))
      {
         if(gain(apply m on x) > 0)
         {
            x <— apply m on x
            break;
         }
      }
   } while(success)
   return x
}
```

facility location

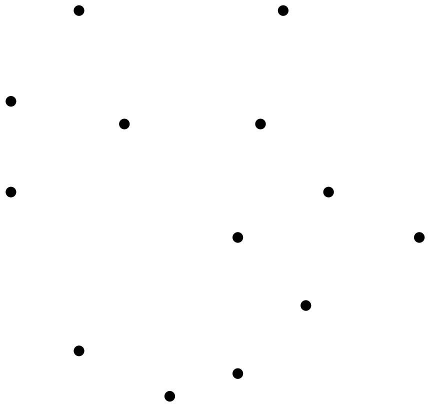


facility location

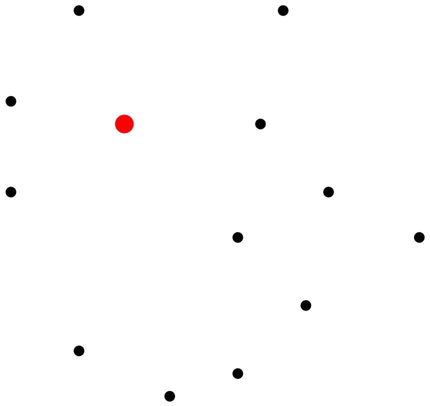


Each point represents a client and a place where we can build a facility

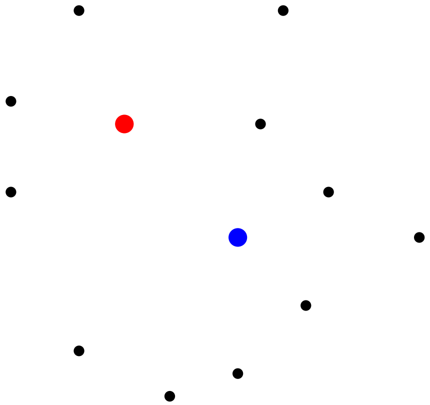
facility location



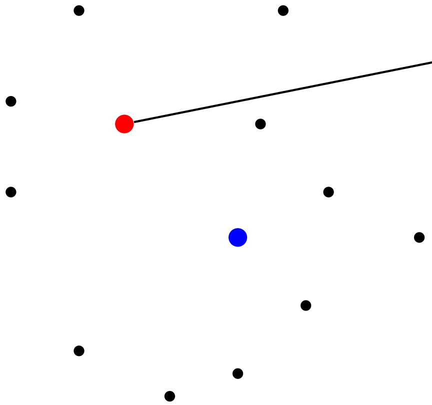
facility location



facility location

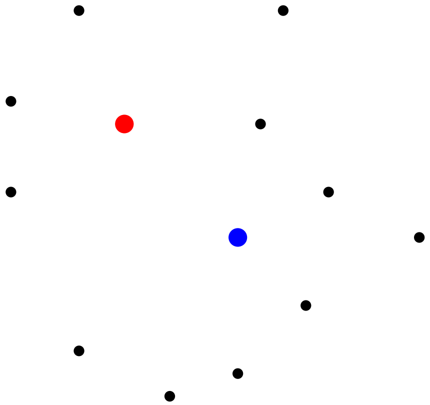


facility location



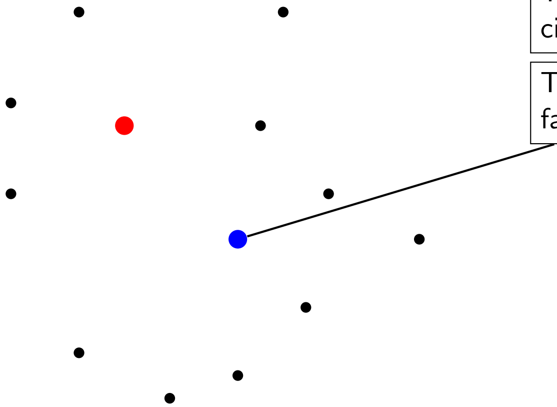
The cost of the red facility is 30$

facility location



The cost of the red facility is 30$

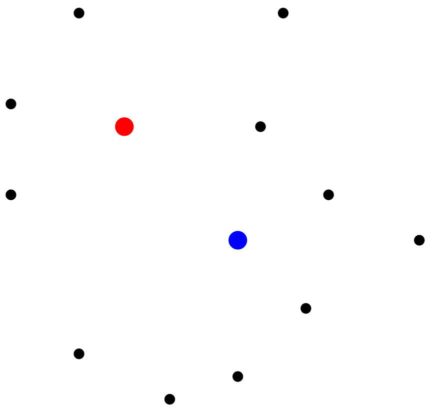
facility location



The cost of the red facility is 30$

The cost of the blue facility is 40$

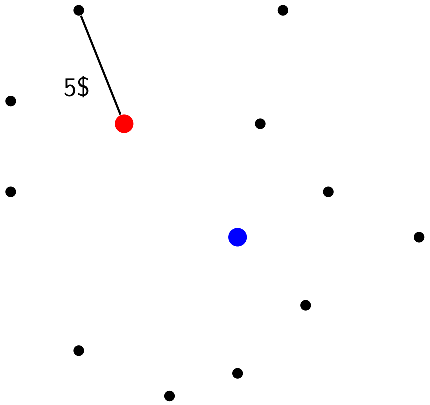
facility location



The cost of the red facility is 30$

The cost of the blue facility is 40$

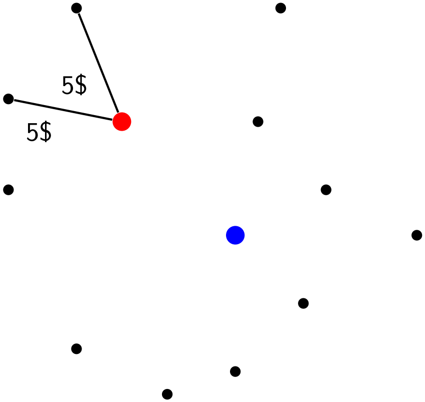
facility location



The cost of the red facility is 30$

The cost of the blue facility is 40$

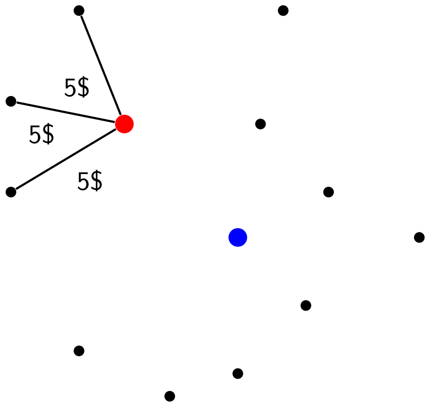
facility location



The cost of the red facility is 30$

The cost of the blue facility is 40$

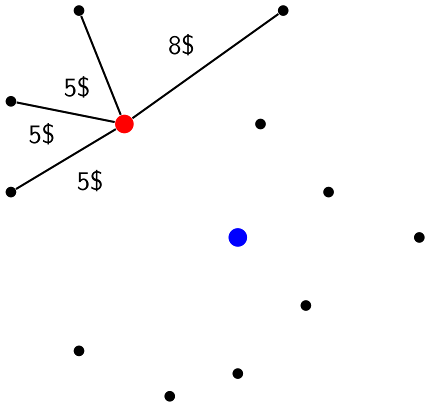
facility location



The cost of the red facility is 30$

The cost of the blue facility is 40$

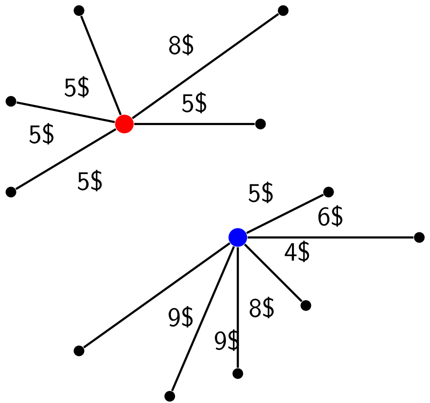
facility location



The cost of the red facility is 30$

The cost of the blue facility is 40$

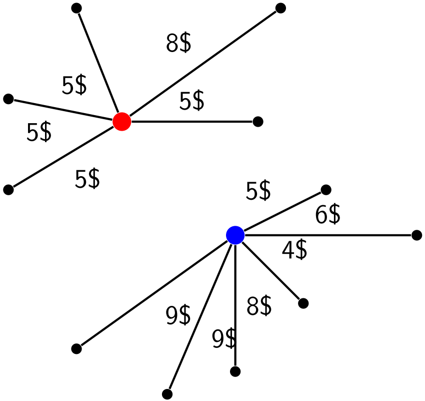
facility location



The cost of the red facility is 30$

The cost of the blue facility is 40$

facility location



The cost of the red facility is 30$

The cost of the blue facility is 40$

The total cost is 30$ + 40$ + 5$ + 5$ + 5$ + 8$+ 5$ + 4$ + 9$ + 9$ + 5$ + 8$ + 6$

facility location local search

How to solve this problem using the local search method?

facility location local search

How to solve this problem using the local search method?
The solution of this problem can be represented by the set of chosen facilities.

facility location local search

How to solve this problem using the local search method?
The solution of this problem can be represented by the set of chosen facilities.
Possible types of moves:

- Add - we can add one facility which is not chosen

facility location local search

How to solve this problem using the local search method?
The solution of this problem can be represented by the set of chosen facilities.
Possible types of moves:

- Add - we can add one facility which is not chosen
- Remove - we can remove one facility which is chosen

facility location local search

How to solve this problem using the local search method?
The solution of this problem can be represented by the set of chosen facilities.
Possible types of moves:

- Add - we can add one facility which is not chosen
- Remove - we can remove one facility which is chosen
- Swap - in one move we can add one facility which is not chosen and remove another which was chosen.

facility location local search

How to solve this problem using the local search method?
The solution of this problem can be represented by the set of chosen facilities.
Possible types of moves:

- Add - we can add one facility which is not chosen
- Remove - we can remove one facility which is chosen
- Swap - in one move we can add one facility which is not chosen and remove another which was chosen.
- ...

goals

- Easy to use
- Speed
- Loose coupling
- Extensibility

first improving

Let us implement a simple strategy that explores the neighborhood and applies the first move that improves the current solution.

implementing model

- $GetMoves : Solution- > MovesRange$
- $Gain : Solution * Move- > Delta$
- $Commit : Solution * Move- > bool$

the first idea

```
namespace ls = local_search;
ls::first_improving(solution, get_moves, gain, commit);
```

possible implementation

```
template <typename Solution, typename GetMoves, typename Gain, typename Commit>
bool first_improving(Solution & solution, GetMoves get_moves, Gain gain, Commit commit) {
    bool success = true;
    while(success) {
        success = false;
        for(auto move : get_moves(solution)) {
            if(gain(solution, move) > 0) {
                success = commit(solution, move);
                if(success) {
                    break;
                }
            }
        }
    }
    return ...;
}
```

extensibility

```
auto print_commit_adaptor = [=](auto & solution, auto move) {
    cout << ”performing commit, move = ” << move << endl;
    return commit(solution, move);
}

ls::first_improving(solution, get_moves, gain, print_commit_adaptor);
```

extensibility

```
//this component is loosely coupled!!!
auto print_commit_adaptor = [=](auto & solution, auto move) {
    cout << "performing commit, move = " << move << endl;
    return commit(solution, move);
}
ls::first_improving(solution, get_moves, gain, print_commit_adaptor);
```

problems

Problems?

managing dependencies

ls::first_improving(solution, get_moves, gain, commit);

- get_moves, gain, commit are often connected and we wish to keep them together.

managing dependencies

This looks like object oriented interface!!!

- $GetMoves : Solution- > MovesRange$
- $Gain : Solution * Move- > Delta$
- $Commit : Solution * Move- > bool$

managing dependencies

```
ls::first_improving(solution
, facility_location_get_moves
, facility_location_gain
, facility_location_commit);
```

Do we have to enumerate all functors associated with the facility location problem?

possible solution

Maybe get_moves, gain, commit should be function members of the same class?

possible solution

Assume that someone written

- 10 versions of get_moves
- 5 versions of gain
- 2 versions of commit

possible solution

Assume that someone written

- 10 versions of get_moves
- 5 versions of gain
- 2 versions of commit

In that case the user has to write 100 classes...

possible solution

Assume that someone written

- 10 versions of get_moves
- 5 versions of gain
- 2 versions of commit

In that case the user has to write 100 classes... Adaptation changes to inheritance (not so nice anymore)

ls::components

```
namespace local_search {
    template <class GetMoves, class Gain, class Commit>
    class components {
        .
        .
        .
    private:
        GetMoves m_get_moves;
        Gain      m_gain;
        Commit    m_commit;
    };
}//!local_search
```

ls::components

```
namespace local_search {
    template <class GetMoves, class Gain, class Commit>
    class components {
        .
        .
        .
    private:
        GetMoves m_get_moves;
        Gain     m_gain;
        Commit   m_commit;
    };
}//!local_search

auto comps = ls::make_components(get_moves, gain, commit);

auto new_comps = replace_gain(comps, new_gain);
```

ls::components

```
auto ls_comps = ls::make_components(get_moves, gain, commit);
ls::first_improving(solution, ls_comps);
```

ls::components

ls::first_improving(solution, facility_location_comps<>{});

ls::components

Problems?

ls::components

Problems?
Non-scalable solution.

ls::components

Use boost::fusion::map!!! (or something similar)

components

First we introduce the names of the components.

```
struct GetMoves;
struct Gain;
struct Commit;
```

components

First we introduce the names of the components.

```
struct GetMoves;
struct Gain;
struct Commit;
```

```
namespace local_search {
    template <typename... Args>
    using components = ::components<GetMoves, Gain, Commit>::type<Args...>;
}//!local_search
```

components

```
ls::components<GetMovesImpl, GainImpl, CommitImpl> comps;
```

components

```
ls::components<GetMovesImpl, GainImpl, CommitImpl> comps;

comps.get<GetMoves>(); //getting GetMoves component
```

components

```
ls::components<GetMovesImpl, GainImpl, CommitImpl> comps;
```

```
comps.get<GetMoves>(); //getting GetMoves component
```

```
GainImpl anotherImplementation(42);
comps.set<Gain>(anotherImplementation); //setting Gain component
```

components

```
ls::components<GetMovesImpl, GainImpl, CommitImpl> comps;
```

```
comps.get<GetMoves>(); //getting GetMoves component
```

```
GainImpl anotherImplementation(42);
comps.set<Gain>(anotherImplementation); //setting Gain component
```

```
comps.call<Commit>(solution, move); // you can directly call a component if it is a functor
```

components

```
ls::components<GetMovesImpl, GainImpl, CommitImpl> comps;

comps.get<GetMoves>(); //getting GetMoves component

GainImpl anotherImplementation(42);
comps.set<Gain>(anotherImplementation); //setting Gain component

comps.call<Commit>(solution, move); // you can directly call a component if it is a functor

auto comps_with_replaced_commit = replace<Commit>(new_commit_with_different_type, comps);
```

components

```
ls::first_improving(solution, facility_location_comps<>{});
```

problems

Problems?

problems

Problems?
Moves can have various types.

problems - various types of moves

Moves can have various types.
Facility location has 3 different types of moves:

- add
- remove
- swap

Each of these types is represented by a different c++ type.

problems - different types of moves

How to implement get_moves functor?

problems - different types of moves

How to implement get_moves functor?

- dynamic polymorphism
- strange class with enums?

problems - different types of moves

How to implement get_moves functor?

- dynamic polymorphism
- strange class with enums?

Writing gain and commit functors is not fun either.

separate components for each kind of move

```
ls::first_improving(solution
, facility_location_components_add<>{}
, facility_location_components_remove<>{}
, facility_location_components_swap<>{});
```

separate components for each kind of move

```
ls::first_improving(solution
, facility_location_components_add<>{}
, facility_location_components_remove<>{}
, facility_location_components_swap<>{});
```

```
ls::first_improving(solution
, facility_location_components_add<>{}
, facility_location_components_remove<>{});
```

separate components for each kind of move

```
ls::first_improving(solution
, facility_location_components_add<>{}
, facility_location_components_remove<>{}
, facility_location_components_swap<>{});
```

```
ls::first_improving(solution
, facility_location_components_remove<>{}
, facility_location_components_add<>{});
```

```
ls::first_improving(solution
, facility_location_components_add<>{}
, facility_location_components_remove<>{});
```

separate components for each kind of move

```
ls::first_improving(solution
, facility_location_components_add<>{}
, facility_location_components_remove<>{}
, facility_location_components_swap<>{});
```

```
ls::first_improving(solution
, facility_location_components_add<>{}
, facility_location_components_remove<>{});
```

```
ls::first_improving(solution
, facility_location_components_remove<>{}
, facility_location_components_add<>{});
```

```
ls::first_improving(solution
, facility_location_components_add<>{}
, facility_location_components_add<>{}
, facility_location_components_remove<>{});
```

facility location implementation

```
class facility_location {
    int addFacility(Facility); // returns cost diff
    int remFacility(Facility); // returns cost diff
    UnchosenRange getUnchosen();
    ChosenRange   getChosen();
};
```

facility location implementation

```
class facility_location {
    int addFacility(Facility); // returns cost diff
    int remFacility(Facility); // returns cost diff
    UnchosenRange getUnchosen();
    ChosenRange   getChosen();
};
```

```
struct facility_location_get_moves_add {
    template <typename Solution>
    auto operator()(const Solution & sol)
    {
        return sol.getUnchosen();
    }
};
```

```
struct facility_location_commit_add {
    template <typename Solution, typename UnchosenElement>
    bool operator()(Solution & s, UnchosenElement e)
    {
        s.add_facility(e);
        return true;
    }
};
```

```
struct facility_location_gain_add {
    template <typename Solution, typename UnchosenElement>
    auto operator()(Solution & s, UnchosenElement e)
    {
        auto ret = s.add_facility(e);
        auto back = s.remove_facility(e);
        assert(ret == -back);
        return -ret;
    }
};
```

example techniques, with rough descriptions

- Hill Climbing (choose only improving moves)
- Random Walk (accept each move)
- Simulated Annealing (with small probability choose also non-improving moves)
- Tabu Search (remember the last visited 100 solutions and filter them out from the neighborhood)

simulated annealing

Assume that we have implemented three functors: get_moves, gain, commit.

simulated annealing

Assume that we have implemented three functors: get_moves, gain, commit.

```
auto cooling = [](){return 5.0;};
```

simulated annealing

Assume that we have implemented three functors: get_moves, gain, commit.

```
auto cooling = [](){return 5.0;};


auto gain_sa = ls::make_simulated_annealing_gain_adaptor(gain, cooling); // we create new gain by
 adapting the old one
```

simulated annealing

Assume that we have implemented three functors: get_moves, gain, commit.

```
auto cooling = [](){return 5.0;};


auto gain_sa = ls::make_simulated_annealing_gain_adaptor(gain, cooling); // we create new gain by
 adapting the old one


ls::first_improving(solution, ls::make_components(get_moves, gain_sa, commit));// we run local search
```

simulated annealing

Assume that we have implemented three functors: get_moves, gain, commit.

```
auto cooling = ls::exponential_cooling_schema_dependent_on_iteration(1000, 0.999); //this is just a functor returning double

auto gain_sa = ls::make_simulated_annealing_gain_adaptor(gain, cooling); // we create new gain by adapting the old one

ls::first_improving(solution, ls::make_components(get_moves, gain_sa, commit));// we run local search
```

recording solution

recording solution

```
auto record_solution_commit =
      ls::make_record_solution_commit_adapter(
            best, //the reference to the best found solution which is going to be updated during the search
            commit);

ls::first_improving(solution, ls::make_components(get_moves, gain_sa, record_solution_commit));// we run local
    search
```

tabu search

tabu search

```
auto gain_tabu =  ls::make_tabu_gain_adaptor(
                  paal::data_structures::tabu_list_remember_solution_and_move<Move, Solution>(20), gain);
ls::first_improving(solution, ls::make_components(get_moves, gain_tabu, record_solution_commit));// we run
     local search
```

tabu search + simulated annealing

tabu search + simulated annealing

```
auto gain_tabu_sa =  ls::make_tabu_gain_adaptor(
                paal::data_structures::tabu_list_remember_solution_and_move<Move, Solution>(20), gain_sa);
ls::first_improving(solution, ls::make_components(get_moves, gain_tabu_sa, record_solution_commit));// we run local search
```

different strategies of searching the neighborhood

-
  ```
  ls::first_improving(solution, comps...);
  ```

different strategies of searching the neighborhood

- ls::first_improving(solution, comps...);

  ls::best_improving(solution, comps...);

different strategies of searching the neighborhood

- ls::first_improving(solution, comps...);

  ls::best_improving(solution, comps...);
- ls::best(solution, comps...);

different strategies of searching the neighborhood

- ls::first_improving(solution, comps...);

  ls::best_improving(solution, comps...);

- ls::best(solution, comps...);

  ls::local_search(solution, strategy, post_search_action, stop_condition, comps...);

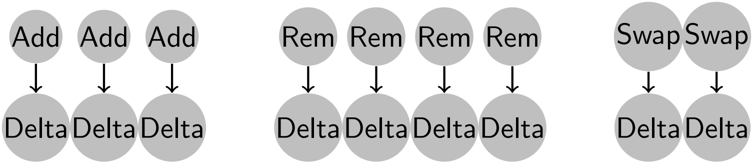
best_improving implementation

Add Add Add

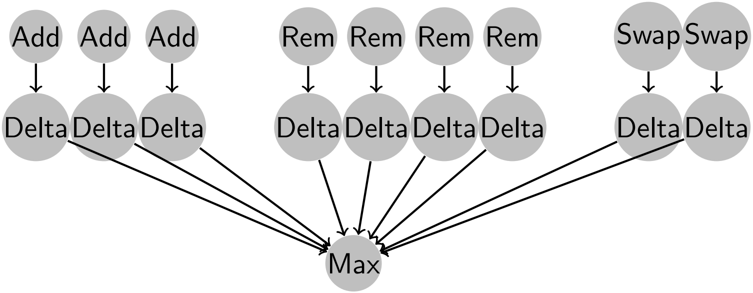
Rem Rem Rem Rem

Swap Swap

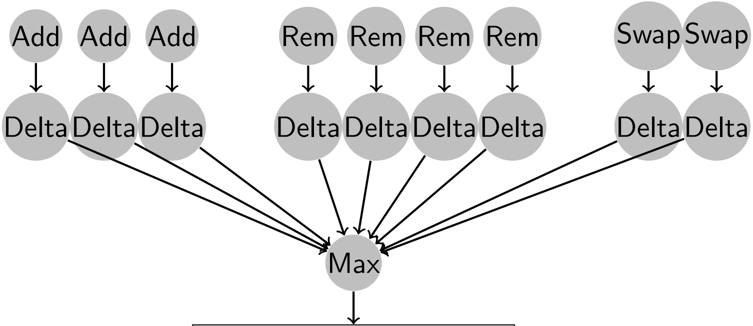
best_improving implementation



best_improving implementation



best_improving implementation



If the best found move is improving, apply appropriate commit functor

simpler task

simpler task

Print the maximum of a polymorphic_list.

```
boost::fusion::vector<int, float, long long> v(12, 5.5f, 1ll);
```

Solution

Solution

$$fold(sequence, accumulator, functor)$$

Solution

$fold(sequence, accumulator, functor)$ //Analogous to std::accumulate

Solution

$fold(sequence, accumulator, functor)$ //Analogous to std::accumulate

$(x_1, x_2, x_3, ..., x_n) = sequence$

Solution

$fold(sequence, accumulator, functor)$ //Analogous to std::accumulate

$(x_1, x_2, x_3, ..., x_n) = sequence$

$functor(accumulator, x_1)$

Solution

$fold(sequence, accumulator, functor)$ //Analogous to std::accumulate

$(x_1, x_2, x_3, ..., x_n) = sequence$

$functor(accumulator, x_1)$

$functor(functor(accumulator, x_1), x_2)$

Solution

$fold(sequence, accumulator, functor)$ //Analogous to std::accumulate

$(x_1, x_2, x_3, ..., x_n) = sequence$

$functor(accumulator, x_1)$

$functor(functor(accumulator, x_1), x_2)$

$functor(...functor(functor(accumulator, x_1), x_2..., x_n)$

Solution...

Assume the accumulator contains the biggest number found so far.

```
struct F {
    template <class Best, class Number>
    auto operator()(Best best, Number num) {
        if(num > best) {
            return num;
        } else {
            return best;
        }
    }
};
```

fold implementation

```
template <class Functor, class AccumulatorFunctor, class AccumulatorData, class Sequence>
  auto polymorphic_fold(
        Functor f,
        AccumulatorFunctor accumulatorFunctor,
        AccumulatorData accumulatorData,
        Sequence & seq) const
  {
    return Fold{}(f, accumulatorFunctor, accumulatorData,
          boost::fusion::begin(seq), boost::fusion::end(seq));
  }
```

fold implementation

```
struct Fold {
   template <class Functor, class IterEnd,  class AccumulatorFunctor, class AccumulatorData>
      auto operator()(Functor,
                  AccumulatorFunctor accumulatorFunctor,
                  AccumulatorData accumulatorData,
                  IterEnd,
                  IterEnd) const {
         return accumulatorFunctor(accumulatorData);
      }

      .
      .
      .

};
```

fold implementation

```
struct Fold {
  template <class Functor, class IterEnd,  class AccumulatorFunctor, class AccumulatorData>
    auto operator()(Functor,
            AccumulatorFunctor accumulatorFunctor,
            AccumulatorData accumulatorData,
            IterEnd,
            IterEnd) const {
      return accumulatorFunctor(accumulatorData);
    }

 template <class Functor, class IterBegin,  class IterEnd, class AccumulatorFunctor, class AccumulatorData>
    auto  operator()(Functor f,
             AccumulatorFunctor accumulatorFunctor,
             AccumulatorData accumulatorData,
             IterBegin begin,
             IterEnd end) const {
      auto continuation = ???;

      return f(*begin, accumulatorFunctor, accumulatorData, continuation);
    }
};
```

fold implementation

```
struct Fold {
  template <class Functor, class IterEnd,  class AccumulatorFunctor, class AccumulatorData>
    auto operator()(Functor,
                    AccumulatorFunctor accumulatorFunctor,
                    AccumulatorData accumulatorData,
                    IterEnd,
                    IterEnd) const {
      return accumulatorFunctor(accumulatorData);
    }

 template <class Functor, class IterBegin,  class IterEnd, class AccumulatorFunctor, class AccumulatorData>
    auto  operator()(Functor f,
                     AccumulatorFunctor accumulatorFunctor,
                     AccumulatorData accumulatorData,
                     IterBegin begin,
                     IterEnd end) const {
      auto continuation = std::bind(*this, f,
              std::placeholders::_1,
              std::placeholders::_2,
              boost::fusion::next(begin),
              end);

      return f(*begin, accumulatorFunctor, accumulatorData, continuation);
    }
};
```

computing polymorphic max using fold

```
struct F {
template <class Num, class AccumulatorFunctor, class AccumulatorData, class Continuation>
auto operator()(Num num, AccumulatorFunctor accFunctor, AccumulatorData accData,  Continuation
    continuation) {
      if(accData < num) {
         auto newAccFunctor = [](auto num){std::cout << num << std::endl;};
         return continuation(newAccFunctor, num);
      } else {
         return continuation(accFunctor, accData);
      }
   }
};
```

computing polymorphic max using fold

```
struct F {
template <class Num, class AccumulatorFunctor, class AccumulatorData, class Continuation>
auto operator()(Num num, AccumulatorFunctor accFunctor, AccumulatorData accData,  Continuation
    continuation) {
      if(accData < num) {
         auto newAccFunctor = [](auto num){std::cout << num << std::endl;};
         return continuation(newAccFunctor, num);
      } else {
         return continuation(accFunctor, accData);
      }
   }
};
```

```
polymorphic_fold(F{}, [](auto){cout << ”Empty Collection” << endl;}, minus_infinity{}, v);
```

what the code looks like?

```
boost::fusion::vector<int, float, long long,int, float, long long, int> v(12, 5.5f, 1ll, 30, 2.2f, 1ll, 45);
polymorphic_fold(F{}, [](auto){cout << ”Empty Collection” << endl;}, minus_infinity{}, v);
```

what the code looks like?

```
boost::fusion::vector<int, float, long long,int, float, long long, int> v(12, 5.5f, 1ll, 30, 2.2f, 1ll, 45);
polymorphic_fold(F{}, [](auto){cout << ”Empty Collection” << endl;}, minus_infinity{}, v);
```

n - number of types, m - number of objects passed

what the code looks like?

```
boost::fusion::vector<int, float, long long,int, float, long long, int> v(12, 5.5f, 1ll, 30, 2.2f, 1ll, 45);
polymorphic_fold(F{}, [](auto){cout << ”Empty Collection” << endl;}, minus_infinity{}, v);
```

n - number of types, m - number of objects passed

here n = 3, m = 7

what the code looks like?

```
boost::fusion::vector<int, float, long long,int, float, long long, int> v(12, 5.5f, 1ll, 30, 2.2f, 1ll, 45);
polymorphic_fold(F{}, [](auto){cout << ”Empty Collection” << endl;}, minus_infinity{}, v);
```

n - number of types, m - number of objects passed

here n = 3, m = 7

The compiler generates $O(n * m)$ specializations of template functions.

what the code looks like?

```
boost::fusion::vector<int, float, long long,int, float, long long, int> v(12, 5.5f, 1ll, 30, 2.2f, 1ll, 45);
polymorphic_fold(F{}, [](auto){cout << ”Empty Collection” << endl;}, minus_infinity{}, v);
```

n - number of types, m - number of objects passed

here n = 3, m = 7

The compiler generates $O(n * m)$ specializations of template functions.

What happens if the compiler decides to inline these functions?

what the code looks like?

```
boost::fusion::vector<int, float, long long,int, float, long long, int> v(12, 5.5f, 1ll, 30, 2.2f, 1ll, 45);
polymorphic_fold(F{}, [](auto){cout << ”Empty Collection” << endl;}, minus_infinity{}, v);
```

n - number of types, m - number of objects passed

here n = 3, m = 7

The compiler generates $O(n * m)$ specializations of template functions.

What happens if the compiler decides to inline these functions?

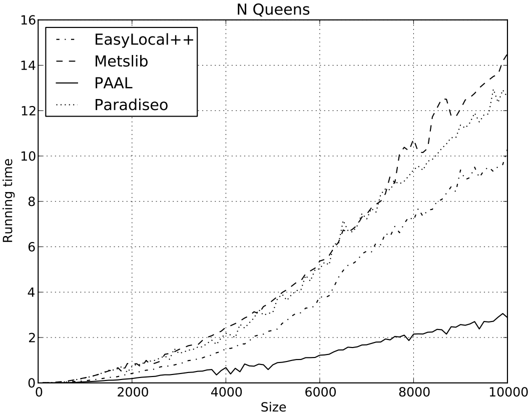
We're going to get the code of size $O(2^m)$.

what the code looks like?

```
boost::fusion::vector<int, float, long long,int, float, long long, int> v(12, 5.5f, 1ll, 30, 2.2f, 1ll, 45);
polymorphic_fold(F{}, [](auto){cout << ”Empty Collection” << endl;}, minus_infinity{}, v);
```

n - number of types, m - number of objects passed
here n = 3, m = 7
The compiler generates $O(n * m)$ specializations of template functions.
What happens if the compiler decides to inline these functions?
We're going to get the code of size $O(2^m)$. (It is an interesting exercise to compute it more precisely)

The comparison with other libraries



Framework	Classes	Functions
PAAL	0	3
Paradiso	4	7
Metslib	1	5
EasyLocal	7	19

Table : Numbers of classes and functions that must be implemented by a programmer in order to use hill climbing in different LS frameworks.

the end

See paal.mimuw.edu.pl

the end

Thank you!