optframe

Release 5.0.18

OptFrame Core Developers

May 16, 2024

CONTENTS

Welcome to OptFrame Documentation, please select the desired content section.

CHAPTER

ONE

INTRODUCTION

OptFrame is a framework for modeling and solving challenging optimization problems via (meta)-heuristic techniques. It is developed in modern C++, aiming to provide both high computational efficiency and easy of use. The project has started in 2008 at Universidade Federal de Ouro Preto (UFOP) and considerably improved since then. Several master and PhD thesis have been developed on it, mostly at the Computing Institute of Universidade Federal Fluminense (UFF). Its latest version v4 has been released in 2020, with the introduction of several functional programming features and C++11/14/17/20 capabilities.

OptFrame supports several state-of-the-art metaheuristics, such as:

- Genetic Algorithm and Evolution Strategies
- Greedy Randomized Adaptive Search Procedures (GRASP)
- Iterated Local Search (ILS)
- Simulated Annealing
- Tabu Search
- Variable Neighborhood Search (VNS)

And also multi-objective optimization metaheuristics, such as NSGA-II.

The theoretical background behind OptFrame modeling will be described in details in *concepts* section. Every OptFrame component can be specialized for the problem at hand, to achieve maximum performance during search. This caused OptFrame users (up to v3) to have several files in a single project, that required some extra setup to start new projects. Since v4, we provide *OptFrame Functional Core (FCore)* abstractions, so that the intended metaheuristic can be implemented in a few lines of C++ code.

If you have already *installed* OptFrame, you can jump to *quick start* to see how it works on practice.

1.1 Acknowledgements

OptFrame has been developed with , thanks to many contributors and advices from brillant minds from academia. Several researchers worldwide have contributed with ideas that represent the core of OptFrame, specially:

- Marcone Jamilson Freitas Souza *(for metaheuristic teaching and support)*
- Nenad Mladenovic *(for great works on neighborhood exploration)*
- Luiz Satoru Ochi *(for metaheuristic teaching and support)*
- Thibaut Lust *(for great ideas on multi-objective optimization)*
- El-Ghazali Talbi *(for great books on metaheuristics and optimization)*

1.2 Contributors

Several contributors made this possible, please refer to the CONTRIBUTORS file on OptFrame project on GitHub.

This project is currently maintained by Igor M. Coelho and Vitor N. Coelho.

1.3 License

Project is free, and source code is released under LGPLv3 copyleft license.

Some side projects such as Scanner++ are released under MIT License.

See complete *license*.

CHAPTER

TWO

INSTALLATION

OptFrame is a standalone C++ project, such that users can simply download a *.zip* from the GitHub repository and use contents from *src/* folder. Instead of downloading a .zip, we recommend users to use git and clone the project (so that they can keep updated with latest fixes).

2.1 Cloning from GitHub

To clone OptFrame repository from GitHub:

```
git clone https://github.com/optframe/optframe.git
```

Then you can run basic tests:

```
make test
```

Output should be: *All tests passed*.

2.2 Linux Installation

By default, OptFrame installer will put headers on */usr/local/include/*, and examples/tests on */usr/local/optframe/*.

You need sudo to execute the script:

```
sudo ./install.sh
```

You need to accept the installation path:

```
Installing OptFrame (with OptFCore) into /usr/local/include and /usr/local/optframe
Continue? y/n
```

Message *Finished installation* will confirm that process has finished.

2.3 Windows Installation

For Windows users it is recommended to follow the steps from *Cloning from GitHub* and use the project locally.

If you want system wide installation, we assume you already have MinGW installed on your PC. If not, this is covered in the MinGW Getting Started Wiki. You'll need both C and C++ compilers (latest MinGW already supports C++17).

2.3.1 Alternative One

Once you have downloaded the OptFrame package (or cloned repository), create a folder on your PC which is readable by all users, for example "C:Program FilesCommon FilesMinGWOptFrame". Copy the directories "libs/" and "src/" from the zip archive to that location.

When compiling with MinGW, remember to add *-I* option, passing include directories:

```
g++ -o myExample example.c -I"C:\Program Files\Common Files\MinGW\OptFrame\src" -I"C:\
↪Program Files\Common Files\MinGW\OptFrame\libs"
```

Hint: Remember that *src* contains all OptFrame files, and *libs* contains all its dependencies.

2.3.2 Alternative Two

A somewhat easier approach is to simply copy content from *libs/* and *src/* to MinGW default include folder. *Note that this may vary on user installation*, but let's assume *C:mingwinclude* to exist.

Files could be placed as: - C:mingwincludeOptFrame - C:mingwincludeOptFCore - C:mingwincludecatch2 - C:mingwincludescannerpplib

If everything works fine, user would be able to use OptFrame without explicit *-I* option:

```
g++ -o myExample example.cpp
```

CHAPTER

THREE

QUICK START

OptFrame is a C++ framework to solve challenging optimization problems, specially by means of metaheuristic techniques. It has no external dependencies, except a C++ compiler compatible with C++17 (or C++20).

After installing OptFrame (or just cloning it), user can start building a solution to the problem at hand.

3.1 Welcome Message

A useful abstraction was introduced since OptFrame 4.1, called OptFrame Functional Core (FCore). The idea of FCore is to provide a simplified access to OptFrame programming classes, by means of closures and lambdas. This way, a project that typically had 8 to 10 files is reduced to a single file (all included in headers!).

The first test on FCore is to see if it's correctly compiling. So let's just include its main header, and print its *welcome()* message:

`File 'mytest.cpp' located in 'demo/01_QuickstartWelcome/'`

3.1.1 Build with make

One way to compile it locally is just copy src/ folder inside your project and see the results (using GCC C++ compiler):

```
g++ --std=c++20 mytest.cpp -o fcore_mytest
./fcore_mytest
# Welcome to OptFrame Functional Core (FCore) - version 4.1-dev
```

If a local copy of OptFrame is being used in other folder (via git clone), one needs to pass its location via flag *-I* to the compiler (*considering relative location of ./optframe/src/*):

`File 'makefile' located in 'demo/01_QuickstartWelcome/'`

```
all: demo_quickstart_welcome

demo_quickstart_welcome:
	g++ --std=c++20 -I../../include src/mytest.cpp -o demo_welcome
	# Expects: Welcome to OptFrame Functional Core (FCore) - version 4.1-dev
	./demo_welcome


build_with_bazel:
	bazel build ... --cxxopt=-std=c++17
```

3.1.2 Build with bazel

If one wants to build in Windows or any other system, easiest manner is to use Bazel. Just create a *WORKSPACE.bazel* file that points to remote OptFrame git repository:

`File 'WORKSPACE.bazel' located in 'demo/01_QuickstartWelcome/'`

```
load('@bazel_tools//tools/build_defs/repo:git.bzl', 'git_repository')
git_repository(
    name='OptFrame',
    remote='https://github.com/optframe/optframe.git',
    branch='master'
)
```

And use the following *BUILD.bazel* file (with dependency to @OptFrame):

`File 'BUILD.bazel' located in 'demo/01_QuickstartWelcome/'`

```
load("@rules_cc//cc:defs.bzl", "cc_library", "cc_binary")
cc_binary(
    name = "demo_welcome",
    srcs = ["src/mytest.cpp"],
    deps=["@OptFrame//include:OptFCore"],
    copts = select({
        "@bazel_tools//src/conditions:windows": ["/std:c++17"],
        "@bazel_tools//src/conditions:darwin": ["--std=c++17"],
        "//conditions:default": ["--std=c++17"],
    }),
)
```

Just invoke bazel build system (assuming MinGW C/C++ toolchain on Windows or just remove option `--compiler` for Linux):

```
bazel build --compiler=mingw-gcc ...
bazel run :demo_welcome
# Welcome to OptFrame Functional Core (FCore) - version 4.1-dev
```

A deeper explanation of OptFrame theoretical foundations can be found on Concepts section, so we will move fast here!

3.2 First Example: 0-1 Knapsack Problem and Simulated Annealing

Let's consider a classic problem: the 0-1 Knapsack Problem (KP).

Given a set of items I, the KP consists in selecting some items $S \subseteq I$, such that the sum of weights w_i (for each selected item) do not exceed knapsack capacity Q, and profit p_i of the selected items is maximized.

$$maximize\; \sum_{i \in S} p_i$$ $$\sum_{i \in S} w_i \leq Q$$ $$S \subseteq I$$

3.2.1 Solution and Evaluation Types

Users must first define a *Representation*, which is a data structure that represents an element in the *Solution Space* for the KP. A natural candidate here is an *array of booleans*, since we need to decide if we are going to carry each item or not. In C++, an interesting approach is to use stl containers, such as a `std::vector<int>`.

User also needs to specify the data type for the *Objective Space*, in general a numeric type. In this case, we will simply choose an `Evaluation<int>` (just ignore the *Evaluation* container for now...).

We declare a XESolution pair that aggregates both spaces as a single type `ESolutionKP` (meaning an *evaluated solution for the knapsack problem*):

```cpp
// SPDX-License-Identifier:  MIT OR LGPL-3.0-or-later
// Copyright (C) 2007-2022 - OptFrame developers
// https://github.com/optframe/optframe

// OptFrame Demo 0-1 Knapsack Problem + Simulated Annealing
// File: KP-fcore-ex.hpp
#pragma once

// C++
#include <algorithm>
#include <functional>
#include <iostream>
#include <utility>
#include <vector>
//
#include <OptFCore/FCore.hpp>
#include <OptFrame/Core.hpp>
#include <OptFrame/Heuristics/Heuristics.hpp>  // many metaheuristics here...
#include <OptFrame/Scanner++/Scanner.hpp>
#include <OptFrame/Util/Matrix.hpp>
#include <OptFrame/printable/printable.hpp>

using namespace std;         // NOLINT
using namespace optframe;    // NOLINT
using namespace scannerpp;   // NOLINT

// namespace para o problema da mochila
// namespace KP_fcore {

// Solução para o problema da mochila, e elemento de avaliação

// ================================
//    Definição do 'ESolution'
// Par: (Representação, Avaliação)
// ================================

using ESolutionKP = std::pair<std::vector<bool>,  // (representation)
                              Evaluation<int>     // (objective value)
                              >;
```

Hint: As long as fields `first` and `second` are provided on XESolution concept, any class can be used instead of

`std::pair`, as demonstrated on Explicit XESolution section.

Warning: Since OptFrame v4, any data structure can be used as a *solution representation*, as long as it implements `operator<<`. OptFrame Classic (`<=` v3) required a templated class named `Solution<R,ADS>`, passing both a *representation* type `R` and *auxiliary data structure* type `ADS`. This is no longer mandatory, but still widely used on the available examples.

3.2.2 Problem Context

Users will need to store general problem information (such as profits and weights of items), so a *ProblemContext* can be introduced. For easy interoperability with file and string inputs (on Linux/Windows), we use *Scanner* class to process problem data (some details of 'load' function will only be discussed in a later moment):

```
class ProblemContext {
 public:
  int n{0};       // numero de itens
  int Q{-1};      // capacidade máxima da mochila
  vector<int> p;  // lucro 'p' de cada item
  vector<int> w;  // peso 'w' de cada item
  //
  // leitor de dados "estilo Java" (Scanner)
  void load(Scanner& scanner) {
    n = *scanner.nextInt();  // leitura do numero de itens na mochila
    Q = *scanner.nextInt();  // leitura da capacidade da mochila
    p = vector<int>(n);      // realoca espaço
    w = vector<int>(n);      // realoca espaço
    //
    cout << "n=" << n << " Q=" << Q << endl;
    //
    // leitura do lucro do item i
    for (int i = 0; i < n; i++) {
      p[i] =
          *scanner.nextInt();  // '*' é usado pq saída do 'nextInt' é 'optional'
      cout << "p[" << i << "]=" << p[i] << " ";
    }
    cout << endl;
    //
    // leitura do peso do item i
    for (int i = 0; i < n; i++) {
      w[i] =
          *scanner.nextInt();  // '*' é usado pq saída do 'nextInt' é 'optional'
      cout << "w[" << i << "]=" << w[i] << " ";
    }
    cout << endl;
  }
};
// Instanciar um problema da mochila pKP
// ProblemContext pKP; // NOLINT
```

Hint: ProblemContext is a user-defined class that can have any desired format. A 'load' function is just a suggestion,

but not at all necessary. The object pKP is declared in global scope just to simplify the example, but it could be embedded in any other component if user desires.

3.2.3 Random Constructive

We need to have some initial solution for the search process, so we just proceed in a random manner. For simplicity, we allow infeasible solutions to be generated (as if capacity was infinite).

```
std::vector<bool> frandom(sref<ProblemContext> pKP) {
  vector<bool> v(pKP->n, false);  // começa sem nenhum item escolhido
  for (unsigned i = 0; i < v.size(); i++)
    v[i] = rand() % 2;  // sorteia se leva item ou não (resultado 0 ou 1)
  // retorna solução
  return v;
}

// Gerador de solução inicial (método construtivo)
// FConstructive<std::vector<bool>, ProblemContext> randomConstructive{pKP,
// frandom};
```

Hint: User can also define many advanced constructive techniques in a similar manner, such as greedy and greedy randomized approaches.

3.2.4 Evaluator

Now it's time to define an evaluation (or objective) function. According to the goal of maximizing the profits, we iterate over selected items to accumulate profit and weights. As discussed in constructive section, we allow accumulated weight to surpass knapsack capacity, for infeasible configurations. To discourage that, we introduce negative penalization whenever capacity is exceeded (assuming weight -1000000):

```
// ============================
//    Função de avaliação
// ============================

// função de avaliação da mochila
Evaluation<int> fevaluate(sref<ProblemContext> pKP,
                          const std::vector<bool>& s) {
  int f = 0;
  int somaPeso = 0;
  for (int i = 0; i < pKP->n; i++)
    if (s[i])  // se elemento foi escolhido
    {
      f += pKP->p[i];         // acumula lucro do elemento i
      somaPeso += pKP->w[i];  // acumula peso do elemento i
    }
  // verifica capacidade excedida
  if (somaPeso >= pKP->Q)
    f -= 1000000 *
         (somaPeso - pKP->Q);  // punição proporcional ao excesso de peso
```

(continues on next page)

(continued from previous page)

```
  //
  return Evaluation<int>{f};
}

// Evaluate (false -> maximização)
// FEvaluator<ESolutionKP, MAXIMIZE> evalKP{fevaluate};
```

We have defined `fevaluate` function to create an `Evaluator` object named `evalKP`, in a `MAXIMIZE` direction.

Hint: User can choose `MINIMIZE` if dealing with a minimization problem. For multi-objective problems and Pareto optimization, user should visit Multi-Objective section.

3.2.5 Neighborhood Structure

In order to improve a given solution, several metaheuristics employ Local Search Optimization techniques based on the concept of Neighborhood Structure. Every neighborhood is related to a move operator, which is required (on FCore) to have an undo operation (capable of reverting the effects of the move).

We create a `BitFlip` move, that changes the `true/false` selection of a given item k. In this case, the *move structure* (representation of the move) is just an `int`, that represents the flipped item.

Note the `makeMoveBitFlip` move generator based on FCore library.

```
// MoveBitFlip (moveData = 'int' (k))
using MoveBitFlip = FMove<int, ESolutionKP>;

// vizinhança: operador de "movimento" que faz "bit flip" (0 -> 1; 1 -> 0) na
// posição k
int fApplyFlip(const int& k, ESolutionKP& se) {
  // se.first[k] = 1 - se.first[k]; // inverte elemento 'k'
  se.first[k] = !se.first[k];  // inverte elemento 'k'
  return k;                    // movimento reverso
}

uptr<Move<ESolutionKP>> makeMoveBitFlip(int k) {
  return uptr<Move<ESolutionKP>>(new MoveBitFlip{k, fApplyFlip});
}
```

The definition of the move can be done using inheritance, if one finds easier (which is what we will adopt).

```
//
class MoveBitFlip : public Move<ESolutionKP> {
 public:
  int k;  // MoveBitFlip (moveData = 'int' (k))

  explicit MoveBitFlip(int _k) : k{_k} {}

  uptr<Move<ESolutionKP>> apply(ESolutionKP& se) override {
    se.first[k] = !se.first[k];                          // reverts element 'k'
    return uptr<Move<ESolutionKP>>(new MoveBitFlip{k});  // returns reverse move
  }
```

(continues on next page)

(continued from previous page)

```
};

uptr<Move<ESolutionKP>> makeMoveBitFlip(int k) {
  return uptr<Move<ESolutionKP>>(new MoveBitFlip{k});
}
```

Now, it's time to define a neighborhood generator for the move. OptFrame has three main types of neighborhoods: `NS`, `NSSeq` and `NSEnum`.

In this example, we will use `NS`, since it only requires the generation of random moves:

```
// gerador de movimentos aleatórios do tipo BitFlip
uptr<Move<ESolutionKP>> fRandomFlip(sref<ProblemContext> pKP,
                                    const ESolutionKP& se) {
  int k = ::rand() % pKP->n;  // sorteia um item entre [0..n-1]
  // cria um "movimento" de flip, na posição k, com operação 'fApplyFlip'
  return uptr<Move<ESolutionKP>>(makeMoveBitFlip(k));
}

// Estrutura de Vizinhança para BitFlip (NS)
// FNS<ESolutionKP> nsFlip{fRandomFlip};

// ================== EVERYTHING TOGETHER ====================

class OptFrameDemoKP {
 public:
  FConstructive<std::vector<bool>, ProblemContext> randomConstructive;
  FEvaluator<ESolutionKP, MAXIMIZE, ProblemContext> evalKP;
  FNS<ESolutionKP, ProblemContext> nsFlip;

  explicit OptFrameDemoKP(sref<ProblemContext> p)
      : randomConstructive{p, frandom},
        evalKP{FEvaluator<ESolutionKP, MAXIMIZE, ProblemContext>{p, fevaluate}},
        nsFlip{FNS<ESolutionKP, ProblemContext>{p, fRandomFlip}} {}
};
```

Hint: It is usually a good idea to start developing over the simplest neighborhood, which is `NS`. Most (non-deterministic) metaheuristics only requires a `NS`, as it only requires the generation of random moves. More advanced neighborhoods based on iterators, such as `NSSeq` and `NSEnum` are only required for advanced Local Search methods.

3.2.6 Time to Test!

At this point, you can already test many nice metaheuristics and solve your knapsack problem! We use the following code to load a problem instance (see *Complete Example* after):

```
std::cout << "======== Carregando Problema ========" << std::endl;
// semente pseudo-aleatória fixa em zero
srand(time(NULL));

std::string sinstance = "knapsack-example.txt";
```

(continues on next page)

(continued from previous page)

```
File f{sinstance};

if (!f.isOpen()) {
  std::cerr << "Problema '" << sinstance << "' não encontrado no diretório!"
            << std::endl;
  return 1;
}

Scanner scanner{std::move(f)};

sref<ProblemContext> pKP{new ProblemContext{}};
pKP->load(scanner);
std::cout << "número de elementos na mochila:" << pKP->n << std::endl;

OptFrameDemoKP demo{pKP};
```

Hint: It is useful to test every FCore structure independently, so as to develop unit testing for them.

To test the constructive and evaluator:

```
std::cout << "======== Testa Construtivo Aleatório ========" << std::endl;
// invoca método 'generateSolution' do FCore 'FConstructive' para construtivo
// aleatório
std::vector<bool> sol = *demo.randomConstructive.generateSolution(0.0);
// imprime solução inicial
std::cout << sol << std::endl;
//
std::cout << "======== Testa Avaliador ========" << std::endl;
// avalia solução inicial e cria um par 'ESolution'
ESolutionKP esol(sol, demo.evalKP.evaluate(sol));
// imprime avaliação da solução inicial
esol.second.print();
```

Now we give an example with of the most well-known metaheuristics: the Simulated Annealing. It has few parameters, including: initial temperature `T0`, cooling factor `alpha`, and iterations per temperature `iterT`.

```
std::cout << "======== Executa Simulated Annealing ========" << std::endl;
// Especifica um gerador aleatório para o Simulated Annealing
RandGen rg;
//
// Cria objeto da classe 'InitialSearch' (parecido com 'construtivoAleatório')
BasicInitialSearch<ESolutionKP> initRand(demo.randomConstructive, demo.evalKP);
// Instancia um Simulated Annealing com alpha=98%, iterações na temp = 100,
// temperatura inicial = 99999
BasicSimulatedAnnealing<ESolutionKP> sa{
    demo.evalKP, initRand, demo.nsFlip, 0.98, 100, 99999, rg};
// executa o SA e coleta o 'status' de saída
// passa um 'Criterio de Parada' por tempo (= 10 segundos)
auto searchOut = sa.search(
    StopCriteria<ESolutionKP::second_type>{10.0});  // 10.0 seconds max
// pega melhor solução do método SA
```

(continues on next page)

(continued from previous page)

```
ESolutionKP melhor = *searchOut.best;  //*sa.getBestSolution();
std::cout << "======== Imprime melhor solução do SA ========" << std::endl;
// imprime representação da melhor solução
cout << melhor.first << endl;
// imprime avaliação da melhor solução
melhor.second.print();
```

3.3 Complete Example

Warning: We present a complete example below. Note that some small differences may exist due to updates in tutorial, including language details. Feel free to check folder `OptFrame/Examples` for other examples on FCore and OptFrame Classic.

Example is divided in two files: `KP-fcore-ex.hpp` and `mainKP-fcore-ex.cpp`.

Hint: This example could be made in a single file, to be even simpler. However, we recommend users to have a clear separation for the header declaration of *FCore components* (on `KP-fcore-ex.hpp`) from the `main()` entrypoint (on `mainKP-fcore-ex.cpp`), since unit testing is much simpler when these are decoupled.

KP-fcore-ex.hpp

`File 'KP-fcore-ex.hpp' located in 'demo/02_QuickstartKP_SA/'`

```
// SPDX-License-Identifier:  MIT OR LGPL-3.0-or-later
// Copyright (C) 2007-2022 - OptFrame developers
// https://github.com/optframe/optframe

// OptFrame Demo 0-1 Knapsack Problem + Simulated Annealing
// File: KP-fcore-ex.hpp
#pragma once

// C++
#include <algorithm>
#include <functional>
#include <iostream>
#include <utility>
#include <vector>
//
#include <OptFCore/FCore.hpp>
#include <OptFrame/Core.hpp>
#include <OptFrame/Heuristics/Heuristics.hpp>  // many metaheuristics here...
#include <OptFrame/Scanner++/Scanner.hpp>
#include <OptFrame/Util/Matrix.hpp>
#include <OptFrame/printable/printable.hpp>

using namespace std;        // NOLINT
using namespace optframe;   // NOLINT
using namespace scannerpp;  // NOLINT
```

(continues on next page)

(continued from previous page)

```

// namespace para o problema da mochila
// namespace KP_fcore {

// Solução para o problema da mochila, e elemento de avaliação

// ================================
//    Definição do 'ESolution'
// Par: (Representação, Avaliação)
// ================================

using ESolutionKP = std::pair<std::vector<bool>,  // (representation)
                              Evaluation<int>     // (objective value)
                              >;
class ProblemContext {
 public:
  int n{0};       // numero de itens
  int Q{-1};      // capacidade máxima da mochila
  vector<int> p;  // lucro 'p' de cada item
  vector<int> w;  // peso 'w' de cada item
  //
  // leitor de dados "estilo Java" (Scanner)
  void load(Scanner& scanner) {
    n = *scanner.nextInt();  // leitura do numero de itens na mochila
    Q = *scanner.nextInt();  // leitura da capacidade da mochila
    p = vector<int>(n);      // realoca espaço
    w = vector<int>(n);      // realoca espaço
    //
    cout << "n=" << n << " Q=" << Q << endl;
    //
    // leitura do lucro do item i
    for (int i = 0; i < n; i++) {
      p[i] =
          *scanner.nextInt();  // '*' é usado pq saída do 'nextInt' é 'optional'
      cout << "p[" << i << "]=" << p[i] << " ";
    }
    cout << endl;
    //
    // leitura do peso do item i
    for (int i = 0; i < n; i++) {
      w[i] =
          *scanner.nextInt();  // '*' é usado pq saída do 'nextInt' é 'optional'
      cout << "w[" << i << "]=" << w[i] << " ";
    }
    cout << endl;
  }
};
// Instanciar um problema da mochila pKP
// ProblemContext pKP; // NOLINT

std::vector<bool> frandom(sref<ProblemContext> pKP) {
  vector<bool> v(pKP->n, false);  // começa sem nenhum item escolhido
```

(continues on next page)

(continued from previous page)

```
  for (unsigned i = 0; i < v.size(); i++)
    v[i] = rand() % 2;  // sorteia se leva item ou não (resultado 0 ou 1)
  // retorna solução
  return v;
}

// Gerador de solução inicial (método construtivo)
// FConstructive<std::vector<bool>, ProblemContext> randomConstructive{pKP,
// frandom};

// ============================
//    Função de avaliação
// ============================

// função de avaliação da mochila
Evaluation<int> fevaluate(sref<ProblemContext> pKP,
                          const std::vector<bool>& s) {
  int f = 0;
  int somaPeso = 0;
  for (int i = 0; i < pKP->n; i++)
    if (s[i])  // se elemento foi escolhido
    {
      f += pKP->p[i];         // acumula lucro do elemento i
      somaPeso += pKP->w[i];  // acumula peso do elemento i
    }
  // verifica capacidade excedida
  if (somaPeso >= pKP->Q)
    f -= 1000000 *
         (somaPeso - pKP->Q);  // punição proporcional ao excesso de peso
  //
  return Evaluation<int>{f};
}

// Evaluate (false -> maximização)
// FEvaluator<ESolutionKP, MAXIMIZE> evalKP{fevaluate};
//
class MoveBitFlip : public Move<ESolutionKP> {
 public:
  int k;  // MoveBitFlip (moveData = 'int' (k))

  explicit MoveBitFlip(int _k) : k{_k} {}

  uptr<Move<ESolutionKP>> apply(ESolutionKP& se) override {
    se.first[k] = !se.first[k];                          // reverts element 'k'
    return uptr<Move<ESolutionKP>>(new MoveBitFlip{k});  // returns reverse move
  }
};

uptr<Move<ESolutionKP>> makeMoveBitFlip(int k) {
  return uptr<Move<ESolutionKP>>(new MoveBitFlip{k});
}
// gerador de movimentos aleatórios do tipo BitFlip
```

(continues on next page)

(continued from previous page)

```
uptr<Move<ESolutionKP>> fRandomFlip(sref<ProblemContext> pKP,
                                    const ESolutionKP& se) {
  int k = ::rand() % pKP->n;  // sorteia um item entre [0..n-1]
  // cria um "movimento" de flip, na posição k, com operação 'fApplyFlip'
  return uptr<Move<ESolutionKP>>(makeMoveBitFlip(k));
}

// Estrutura de Vizinhança para BitFlip (NS)
// FNS<ESolutionKP> nsFlip{fRandomFlip};

// ================== EVERYTHING TOGETHER ====================

class OptFrameDemoKP {
 public:
  FConstructive<std::vector<bool>, ProblemContext> randomConstructive;
  FEvaluator<ESolutionKP, MAXIMIZE, ProblemContext> evalKP;
  FNS<ESolutionKP, ProblemContext> nsFlip;

  explicit OptFrameDemoKP(sref<ProblemContext> p)
      : randomConstructive{p, frandom},
        evalKP{FEvaluator<ESolutionKP, MAXIMIZE, ProblemContext>{p, fevaluate}},
        nsFlip{FNS<ESolutionKP, ProblemContext>{p, fRandomFlip}} {}
};
```

mainKP-fcore-ex.cpp

File 'mainKP-fcore-ex.cpp' located in 'demo/02_QuickstartKP_SA/'

```
// mainKP-fcore-ex.cpp

// C++
#include <iostream>

//
#include "KP-fcore-ex.hpp"  // implementação da mochila

// import everything on main()
using namespace std;        // NOLINT
using namespace optframe;   // NOLINT
using namespace scannerpp;  // NOLINT
// using namespace KP_fcore;

int main(int argc, char** argv) {std::cout << "======== Carregando Problema ========" <<␣
↪std::endl;
// semente pseudo-aleatória fixa em zero
srand(time(NULL));

std::string sinstance = "knapsack-example.txt";
File f{sinstance};

if (!f.isOpen()) {
  std::cerr << "Problema '" << sinstance << "' não encontrado no diretório!"
            << std::endl;
```

(continues on next page)

(continued from previous page)

```
  return 1;
}

Scanner scanner{std::move(f)};

sref<ProblemContext> pKP{new ProblemContext{}};
pKP->load(scanner);
std::cout << "número de elementos na mochila:" << pKP->n << std::endl;

OptFrameDemoKP demo{pKP};
std::cout << "======== Testa Construtivo Aleatório ========" << std::endl;
// invoca método 'generateSolution' do FCore 'FConstructive' para construtivo
// aleatório
std::vector<bool> sol = *demo.randomConstructive.generateSolution(0.0);
// imprime solução inicial
std::cout << sol << std::endl;
//
std::cout << "======== Testa Avaliador ========" << std::endl;
// avalia solução inicial e cria um par 'ESolution'
ESolutionKP esol(sol, demo.evalKP.evaluate(sol));
// imprime avaliação da solução inicial
esol.second.print();

std::cout << "======== Executa Simulated Annealing ========" << std::endl;
// Especifica um gerador aleatório para o Simulated Annealing
RandGen rg;
//
// Cria objeto da classe 'InitialSearch' (parecido com 'construtivoAleatório')
BasicInitialSearch<ESolutionKP> initRand(demo.randomConstructive, demo.evalKP);
// Instancia um Simulated Annealing com alpha=98%, iterações na temp = 100,
// temperatura inicial = 99999
BasicSimulatedAnnealing<ESolutionKP> sa{
    demo.evalKP, initRand, demo.nsFlip, 0.98, 100, 99999, rg};
// executa o SA e coleta o 'status' de saída
// passa um 'Criterio de Parada' por tempo (= 10 segundos)
auto searchOut = sa.search(
    StopCriteria<ESolutionKP::second_type>{10.0});  // 10.0 seconds max
// pega melhor solução do método SA
ESolutionKP melhor = *searchOut.best;  //*sa.getBestSolution();
std::cout << "======== Imprime melhor solução do SA ========" << std::endl;
// imprime representação da melhor solução
cout << melhor.first << endl;
// imprime avaliação da melhor solução
melhor.second.print();


std::cout << "======== Fim da Execução ========" << std::endl;
return 0;
} // main
```

knapsack-example.txt

```
File 'knapsack-example.txt' located in 'demo/02_QuickstartKP_SA/'
```

```
5
10
1 1 1 5 5
1 2 3 7 8
```

To compile it (generates binary *app_KP*):

```
g++ -g -O3 --std=c++20 -I../../include  mainKP-fcore-ex.cpp -o app_KP
```

CHAPTER

FOUR

QUICK START (CONTINUED)

We expand the knowledge from Quick Start where the user has learned how to Install OptFrame, and how to compile and test a Simulated Annealing metaheuristic for the classic 0-1 Knapsack Problem (01KP).

We demonstrate how to update this code for other metaheuristics, for the classic Traveling Salesman Problem (TSP).

4.1 Second example: Traveling Salesman Problem, Local Search and Random Keys Genetic Algorithm

At this point, we assume the reader is familiarized with the Traveling Salesman Problem... we intend to expand this section in the future with figures and more motivation (for now, sorry, and let's move on).

4.1.1 TSP Solution definition

We define a TSP solution as a permutations of N cities being visited by a Traveling Salesman. In this representation, each city is represented as a number $0..N-1$, being a solution a vector of N integers (example: [0,2,3,1] means that solution starts from city 0, then follows to city 2, then city 3, then city 1, and finally comes back to city 0). Objective is to find a route that minimizes distance between the N visited cities.

We may define ESolutionTSP as a pair, containing a solution and its objective value (double).

```
#pragma once

// C++
#include <algorithm>
#include <functional>
#include <iostream>
#include <vector>
//
#include <OptFCore/FCore.hpp>
#include <OptFrame/Core.hpp>
#include <OptFrame/Heuristics/Heuristics.hpp>  // many metaheuristics here...
#include <OptFrame/Scanner++/Scanner.hpp>
#include <OptFrame/Util/Matrix.hpp>
#include <OptFrame/printable/printable.hpp>

using namespace std;         // NOLINT
using namespace optframe;    // NOLINT
using namespace scannerpp;   // NOLINT
```

(continues on next page)

(continued from previous page)

```

// define TSP solution type as 'vector<int>', using 'double' as evaluation type
using ESolutionTSP =
    std::pair<std::vector<int>,  // first part of search space element: solution
                                 // (representation)
              Evaluation<int>    // second part of search space element:
                                 // evaluation (objective value)
              >;
```

4.1.2 Problem Data

We read a matrix of distances between pairs of cities (considering Euclidean distance), and store in a structure named ProblemContext. Note that we round the result to int, just to allow precise value calculation (but one may use float or double, and then manage the floating-point errors). Do not forget to #include helpers from optframe namespace, such as Matrix and Scanner.

```
// TSP problem context and data reads
class ProblemContext {
 public:
  int n{0};           // number of clients
  Matrix<int> dist;  // distance matrix (Euclidean)
  // load data from Scanner
  void load(Scanner& scanner) {
    n = *scanner.nextInt();    // reads number of clients
    dist = Matrix<int>(n, n);  // initializes n x n matrix
    //
    vector<double> xvalues(n);
    vector<double> yvalues(n);
    //
    for (int i = 0; i < n; i++) {
      scanner.next();
      xvalues[i] = *scanner.nextDouble();  // reads x
      yvalues[i] = *scanner.nextDouble();  // reads y
    }
    // calculate distance values, for every client pair (i,j)
    for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
        dist(i, j) = (int)::round(distance(xvalues.at(i), yvalues.at(i),
                                           xvalues.at(j), yvalues.at(j)));
  }
  // euclidean distance (double as return)
  double distance(double x1, double y1, double x2, double y2) {
    return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
  }
};
// Create TSP Problem Context
// ProblemContext pTSP;
```

Finally, we declare a global object `ProblemContext pTSP` (for simplicity), that includes all data.

4.1.3 Evaluation

Objective calculation can be done by using Functional Core class FEvaluator, which is a huge simplification from OptFrame Core components Evaluator (for single objectives) and GeneralEvaluator (for single and multiple objectives).

```
Evaluation<int> fevaluate(sref<ProblemContext> pTSP,
                          const std::vector<int>& s) {
  int f = 0;
  for (int i = 0; i < int(pTSP->n) - 1; i++) f += pTSP->dist(s[i], s[i + 1]);
  f += pTSP->dist(s[int(pTSP->n) - 1], s[0]);
  return Evaluation<int>{f};
}

// Evaluate
// FEvaluator<ESolutionTSP, MinOrMax::MINIMIZE> eval{fevaluate};
```

4.2 Search methods based on neighborhoods

The next components will depend on the type of search method used, we start with neighborhood-based search techniques.

4.2.1 Random constructive

In a similar manner with Knapsack example (on Quickstart part 1), we define a random solution generator.

```
std::vector<int> frandom(sref<ProblemContext> pTSP) {
  vector<int> v(pTSP->n, -1);  // get information from context
  for (int i = 0; i < (int)v.size(); i++) v[i] = i;
  std::random_shuffle(v.begin(), v.end());
  return v;
}

// Generate random solution
// FConstructive<std::vector<int>, ProblemContext> crand{frandom};
```

4.2.2 First move operator: Swap

We start with a Move operator capable of exchanging two elements from a given TSP solution.

```
std::pair<int, int> fApplySwap(sref<ProblemContext>,
                               const std::pair<int, int>& moveData,
                               ESolutionTSP& se) {
  int i = moveData.first;
  int j = moveData.second;
  // perform swap of clients i and j
  int aux = se.first[j];
  se.first[j] = se.first[i];
  se.first[i] = aux;
  return std::pair<int, int>(j, i);  // return a reverse move ('undo' move)s
```

(continues on next page)

(continued from previous page)

```
}

// Swap move
using MoveSwap = FMoveP<std::pair<int, int>, ESolutionTSP, ProblemContext>;

uptr<Move<ESolutionTSP>> makeMoveSwap(sref<ProblemContext> p, int i, int j) {
  return uptr<Move<ESolutionTSP>>(new MoveSwap{p, make_pair(i, j), fApplySwap});
}
```

We have four types of neighborhood definitions in OptFrame (NS, NSFind, NSSeq and NSEnum), but two major are NS and NSSeq.

NS works well for random neighborhoods, with no intent of explicitly visiting all possible neighbors (also useful for continuous or infinite neighborhoods). Swap move and NS definition can be seen below.

```
uptr<Move<ESolutionTSP>> fRandomSwap(sref<ProblemContext> pTSP,
                                     const ESolutionTSP& se) {
  int i = rand() % pTSP->n;
  int j = i;
  while (j <= i) {
    i = rand() % pTSP->n;
    j = rand() % pTSP->n;
  }
  return uptr<Move<ESolutionTSP>>(makeMoveSwap(pTSP, i, j));
}

// Swap move (NS)
// FNS<ESolutionTSP> nsswap{fRandomSwap};
```

We now define the NSSeq neighborhood with a explicit iterator definition, that requires four operations: first (initializes first valid move), next (skips to next valid move), current (returns current move) and isDone (indicates if no move exists).

```
auto make_nsseq(sref<ProblemContext> p) {
  sref<FNSSeq<std::pair<int, int>, ESolutionTSP, ProblemContext>> nsseq2{
      new FNSSeq<std::pair<int, int>, ESolutionTSP, ProblemContext>{
          p,
          [](sref<ProblemContext> pTSP,
             const ESolutionTSP& se) -> uptr<Move<ESolutionTSP>> {
            int i = rand() % pTSP->n;
            int j = i;
            while (j <= i) {
              i = rand() % pTSP->n;
              j = rand() % pTSP->n;
            }
            return uptr<Move<ESolutionTSP>>(makeMoveSwap(pTSP, i, j));
          },
          // iterator initialization (fGenerator)
          [](sref<ProblemContext> pTSP, const ESolutionTSP& se)
              -> std::pair<int, int> { return make_pair(-1, -1); },
          [](sref<ProblemContext> pTSP, std::pair<int, int>& p) -> void {
            // void (*fFirst)(IMS&),                    // iterator.first()
            p.first = 0;
            p.second = 1;
```

(continues on next page)

(continued from previous page)

```
        },
        [](sref<ProblemContext> pTSP, std::pair<int, int>& p) -> void {
          // void (*fNext)(IMS&),                    // iterator.next()
          if (p.second < (pTSP->n - 1))
            p.second++;
          else {
            p.first++;
            p.second = p.first + 1;
          }
        },
        [](sref<ProblemContext> pTSP, std::pair<int, int>& p) -> bool {
          // bool (*fIsDone)(IMS&),                  //
          // iterator.isDone()
          return p.first >= pTSP->n - 1;
        },
        [](sref<ProblemContext> pTSP,
           std::pair<int, int>& p) -> uptr<Move<ESolutionTSP>> {
          // uptr<Move<XES>> (*fCurrent)(IMS&)       //
          // iterator.current()
          return uptr<Move<ESolutionTSP>>(
              makeMoveSwap(pTSP, p.first, p.second));
        }  // FNSSeq
    }};
  return nsseq2;
}

class OptFrameDemoTSP {
 public:
  sref<ProblemContext> pTSP;
  sref<FConstructive<std::vector<int>, ProblemContext>> randomConstructive;
  sref<FEvaluator<ESolutionTSP, MINIMIZE, ProblemContext>> eval;
  sref<FNS<ESolutionTSP, ProblemContext>> nsSwap;
  sref<FNSSeq<std::pair<int, int>, ESolutionTSP, ProblemContext>> nsseqSwap;
  sptr<DecoderRandomKeys<ESolutionTSP, double>> decoder;

  explicit OptFrameDemoTSP(sref<ProblemContext> p)
      : pTSP{p},
        randomConstructive{
            new FConstructive<std::vector<int>, ProblemContext>{p, frandom}},
        eval{new FEvaluator<ESolutionTSP, MINIMIZE, ProblemContext>{p,
                                                                    fevaluate}},
        nsSwap{new FNS<ESolutionTSP, ProblemContext>{p, fRandomSwap}},
        nsseqSwap{make_nsseq(p)}

  {}
};

//
```

Hint: According to groundbreaking ideas from Variable Neighborhood Search community, the user should create multiple neighborhoods, with different ideas in each one, in order to better explore the solution space.

At this point, we quickly demonstrate how novel C++20 features, such as Coroutines, have improved OptFrame in latest versions (named FxCore library). We note that all four iterator operations (first, next, current and isDone) are made available quite naturally with a single coroutine generator that executes co_yield for each available move.

```
// THIS IS ONLY AVAILABLE WITH OptFrame FxCore and C++20 -fcoroutines
// Swap move (NSSeq) - with "Fancy" iterator (coroutines)
using NSSeqSwapFancy = FxNSSeqFancy<
  ESolutionTSP,
  [](const ESolutionTSP& se) -> uptr<Move<ESolutionTSP>> {
     int i = rand() % pTSP.n;
     int j = i;
     while (j <= i) {
        i = rand() % pTSP.n;
        j = rand() % pTSP.n;
     }
     return uptr<Move<ESolutionTSP>>(new MoveSwap{ make_pair(i, j) });
  },
  [](const ESolutionTSP& se) -> Generator<Move<ESolutionTSP>*> {
     for (int i = 0; i < int(pTSP.n) - 1; i++)
        for (int j = i + 1; j < pTSP.n; j++)
           co_yield new MoveSwap{ make_pair(i, j) }; // implicit unique_ptr requirements
  }>;
```

We will explore more of these bleeding-edge tools in an advanced topic.

4.2.3 Complete Example for TSP Components

For simplicity, we separate the main TSP components in a file named `TSP-fcore.hpp`.

Hint: This example could be made in a single file, to be even simpler. However, we recommend users to have a clear separation for the header declaration of *FCore components* (on `TSP-fcore.hpp`) from the `main()` entrypoint depending on method used, e.g., `mainTSP-fcore-ils.cpp` or `mainTSP-fcore-brkga.cpp`.

TSP-fcore.hpp

`File 'TSP-fcore.hpp' located in 'demo/03_QuickstartTSP_VNS_BRKGA/'`

```
#pragma once

// C++
#include <algorithm>
#include <functional>
#include <iostream>
#include <vector>
//
#include <OptFCore/FCore.hpp>
#include <OptFrame/Core.hpp>
#include <OptFrame/Heuristics/Heuristics.hpp>  // many metaheuristics here...
#include <OptFrame/Scanner++/Scanner.hpp>
#include <OptFrame/Util/Matrix.hpp>
#include <OptFrame/printable/printable.hpp>

```

(continues on next page)

(continued from previous page)

```cpp
using namespace std;         // NOLINT
using namespace optframe;    // NOLINT
using namespace scannerpp;   // NOLINT

// define TSP solution type as 'vector<int>', using 'double' as evaluation type
using ESolutionTSP =
    std::pair<std::vector<int>,  // first part of search space element: solution
                                 // (representation)
              Evaluation<int>    // second part of search space element:
                                 // evaluation (objective value)
              >;

// TSP problem context and data reads
class ProblemContext {
 public:
  int n{0};           // number of clients
  Matrix<int> dist;  // distance matrix (Euclidean)
  // load data from Scanner
  void load(Scanner& scanner) {
    n = *scanner.nextInt();    // reads number of clients
    dist = Matrix<int>(n, n);  // initializes n x n matrix
    //
    vector<double> xvalues(n);
    vector<double> yvalues(n);
    //
    for (int i = 0; i < n; i++) {
      scanner.next();
      xvalues[i] = *scanner.nextDouble();  // reads x
      yvalues[i] = *scanner.nextDouble();  // reads y
    }
    // calculate distance values, for every client pair (i,j)
    for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
        dist(i, j) = (int)::round(distance(xvalues.at(i), yvalues.at(i),
                                           xvalues.at(j), yvalues.at(j)));
  }
  // euclidean distance (double as return)
  double distance(double x1, double y1, double x2, double y2) {
    return sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
  }
};
// Create TSP Problem Context
// ProblemContext pTSP;

Evaluation<int> fevaluate(sref<ProblemContext> pTSP,
                          const std::vector<int>& s) {
  int f = 0;
  for (int i = 0; i < int(pTSP->n) - 1; i++) f += pTSP->dist(s[i], s[i + 1]);
  f += pTSP->dist(s[int(pTSP->n) - 1], s[0]);
  return Evaluation<int>{f};
}

```

(continues on next page)

(continued from previous page)

```
// Evaluate
// FEvaluator<ESolutionTSP, MinOrMax::MINIMIZE> eval{fevaluate};
std::vector<int> frandom(sref<ProblemContext> pTSP) {
  vector<int> v(pTSP->n, -1);  // get information from context
  for (int i = 0; i < (int)v.size(); i++) v[i] = i;
  std::random_shuffle(v.begin(), v.end());
  return v;
}

// Generate random solution
// FConstructive<std::vector<int>, ProblemContext> crand{frandom};

std::pair<int, int> fApplySwap(sref<ProblemContext>,
                               const std::pair<int, int>& moveData,
                               ESolutionTSP& se) {
  int i = moveData.first;
  int j = moveData.second;
  // perform swap of clients i and j
  int aux = se.first[j];
  se.first[j] = se.first[i];
  se.first[i] = aux;
  return std::pair<int, int>(j, i);  // return a reverse move ('undo' move)s
}

// Swap move
using MoveSwap = FMoveP<std::pair<int, int>, ESolutionTSP, ProblemContext>;

uptr<Move<ESolutionTSP>> makeMoveSwap(sref<ProblemContext> p, int i, int j) {
  return uptr<Move<ESolutionTSP>>(new MoveSwap{p, make_pair(i, j), fApplySwap});
}
uptr<Move<ESolutionTSP>> fRandomSwap(sref<ProblemContext> pTSP,
                                     const ESolutionTSP& se) {
  int i = rand() % pTSP->n;
  int j = i;
  while (j <= i) {
    i = rand() % pTSP->n;
    j = rand() % pTSP->n;
  }
  return uptr<Move<ESolutionTSP>>(makeMoveSwap(pTSP, i, j));
}

// Swap move (NS)
// FNS<ESolutionTSP> nsswap{fRandomSwap};

auto make_nsseq(sref<ProblemContext> p) {
  sref<FNSSeq<std::pair<int, int>, ESolutionTSP, ProblemContext>> nsseq2{
      new FNSSeq<std::pair<int, int>, ESolutionTSP, ProblemContext>{
          p,
          [](sref<ProblemContext> pTSP,
             const ESolutionTSP& se) -> uptr<Move<ESolutionTSP>> {
            int i = rand() % pTSP->n;
            int j = i;
```

(continues on next page)

(continued from previous page)

```
            while (j <= i) {
              i = rand() % pTSP->n;
              j = rand() % pTSP->n;
            }
            return uptr<Move<ESolutionTSP>>(makeMoveSwap(pTSP, i, j));
          },
          // iterator initialization (fGenerator)
          [](sref<ProblemContext> pTSP, const ESolutionTSP& se)
              -> std::pair<int, int> { return make_pair(-1, -1); },
          [](sref<ProblemContext> pTSP, std::pair<int, int>& p) -> void {
            // void (*fFirst)(IMS&),                    // iterator.first()
            p.first = 0;
            p.second = 1;
          },
          [](sref<ProblemContext> pTSP, std::pair<int, int>& p) -> void {
            // void (*fNext)(IMS&),                     // iterator.next()
            if (p.second < (pTSP->n - 1))
              p.second++;
            else {
              p.first++;
              p.second = p.first + 1;
            }
          },
          [](sref<ProblemContext> pTSP, std::pair<int, int>& p) -> bool {
            // bool (*fIsDone)(IMS&),                   //
            // iterator.isDone()
            return p.first >= pTSP->n - 1;
          },
          [](sref<ProblemContext> pTSP,
             std::pair<int, int>& p) -> uptr<Move<ESolutionTSP>> {
            // uptr<Move<XES>> (*fCurrent)(IMS&)        //
            // iterator.current()
            return uptr<Move<ESolutionTSP>>(
                makeMoveSwap(pTSP, p.first, p.second));
          }  // FNSSeq
      }};
  return nsseq2;
}

class OptFrameDemoTSP {
 public:
  sref<ProblemContext> pTSP;
  sref<FConstructive<std::vector<int>, ProblemContext>> randomConstructive;
  sref<FEvaluator<ESolutionTSP, MINIMIZE, ProblemContext>> eval;
  sref<FNS<ESolutionTSP, ProblemContext>> nsSwap;
  sref<FNSSeq<std::pair<int, int>, ESolutionTSP, ProblemContext>> nsseqSwap;
  sptr<DecoderRandomKeys<ESolutionTSP, double>> decoder;

  explicit OptFrameDemoTSP(sref<ProblemContext> p)
      : pTSP{p},
        randomConstructive{
            new FConstructive<std::vector<int>, ProblemContext>{p, frandom}},
```

(continues on next page)

(continued from previous page)

```
        eval{new FEvaluator<ESolutionTSP, MINIMIZE, ProblemContext>{p,
                                                                    fevaluate}},
        nsSwap{new FNS<ESolutionTSP, ProblemContext>{p, fRandomSwap}},
        nsseqSwap{make_nsseq(p)}

  {}
};

//
```

4.2.4 Exploring with neighborhood exploration

OptFrame supports several strategies for neighborhood exploration, such as: First Improvement, Best Improvement, Random Selection and Multi Improvement. We can also combine several local search strategies in a multiple strategy called Variable Neighborhood Descent (VND).

```
int main() {
  srand(0);  // using system random (weak... just an example!)

  // load data into problem context 'pTSP'
  Scanner scanner{"5\n1 10 10\n2 20 20\n3 30 30\n4 40 40\n5 50 50\n"};
  sref<ProblemContext> pTSP{new ProblemContext{}};
  pTSP->load(scanner);
  std::cout << pTSP->dist << std::endl;

  OptFrameDemoTSP demo{pTSP};

  // evaluator
  // TSPEval ev;
  //
  // create simple solution
  // TSPRandom crand;
  //
  std::vector<int> sol = *demo.randomConstructive->generateSolution(0);
  std::cout << sol << std::endl;

  // evaluation value and store on ESolution pair
  ESolutionTSP esol(sol, demo.eval->evaluate(sol));
  esol.second.print();  // print evaluation

  // swap 0 with 1
  MoveSwap move{demo.pTSP, make_pair(0, 1), fApplySwap};
  move.print();

  // NSSwap nsswap;
  // move for solution 'esol'
  auto m1 = demo.nsSwap->randomMove(esol);
  m1->print();

  std::cout << std::endl;
  std::cout << "begin listing NSSeqSwapFancy" << std::endl;
```

(continues on next page)

(continued from previous page)

```
  //
  auto it1 = demo.nsseqSwap->getIterator(esol);
  for (it1->first(); !it1->isDone(); it1->next()) it1->current()->print();
  std::cout << "end listing NSSeqSwapFancy" << std::endl;

  // Random number generator
  RandGen rg;                       // stack version
  sref<RandGen> rg2{new RandGen};  // heap version (safely shared)
  // testing simulated annealing
  BasicInitialSearch<ESolutionTSP> initRand(demo.randomConstructive, demo.eval);

  vsref<LocalSearch<ESolutionTSP>> ns_list;
  Evaluator<ESolutionTSP::first_type, ESolutionTSP::second_type>* ev2 =
      new FEvaluator<ESolutionTSP, MinOrMax::MINIMIZE, ProblemContext>{
          pTSP, fevaluate};
  GeneralEvaluator<ESolutionTSP>* gev2 = (GeneralEvaluator<ESolutionTSP>*)ev2;
  sref<GeneralEvaluator<ESolutionTSP>> eval2(gev2);
  ns_list.push_back(new BestImprovement<ESolutionTSP>(eval2, demo.nsseqSwap));

  VariableNeighborhoodDescent<ESolutionTSP> VND(demo.eval, ns_list);
  // VND.setVerbose();
```

If one wants to build a complete metaheuristic, the Iterated Local Search (ILS) or a Variable Neighborhood Search (VNS). The ILS is based on general perturbation concept, so we will use the concept of Levels of Perturbation, that are increased when stuck in a poor quality local optimum. We adopt a perturbation strategy that tries to escape at level p by applying p+2 random moves, e.g., at level 0, 2 random moves are applied, and so on.

```
//
ILSLPerturbationLPlus2<ESolutionTSP> pert(demo.eval, demo.nsSwap, rg2);

IteratedLocalSearchLevels<ESolutionTSP> ils(demo.eval, initRand, VND, pert, 10,
                                            5);
// ils.setVerbose();

std::cout << "will start ILS for 3 seconds" << std::endl;

auto status = ils.search(
    StopCriteria<ESolutionTSP::second_type>{3.0});  // 3.0 seconds max
ESolutionTSP best = *status.best;
// best solution value
best.second.print();
std::cout << "solution: " << best.first << std::endl;

std::cout << "FINISHED" << std::endl;
return 0;
}
```

4.2.5 Complete Example for ILS

We provide the main file for TSP ILS `mainTSP-fcore-ils.cpp`.

mainTSP-fcore-ils.cpp

`File 'mainTSP-fcore-ils.cpp' located in 'demo/03_QuickstartTSP_VNS_BRKGA/'`

```cpp
// SPDX-License-Identifier:  MIT OR LGPL-3.0-or-later
// Copyright (C) 2007-2022 - OptFrame developers
// https://github.com/optframe/optframe

// OptFrame Demo TSP - Iterated Local Search

// C++
#include <iostream>
//
#include "TSP-fcore.hpp"
// implementation of TSP

#include <OptFrame/Core.hpp>
#include <OptFrame/Heuristics/Heuristics.hpp>  // many metaheuristics here...
#include <OptFrame/Heuristics/ILS/IteratedLocalSearchLevels.hpp>
#include <OptFrame/Heuristics/LocalSearches/BestImprovement.hpp>
#include <OptFrame/Heuristics/LocalSearches/VariableNeighborhoodDescent.hpp>
#include <OptFrame/LocalSearch.hpp>

// import everything on main()
using namespace std;         // NOLINT
using namespace optframe;    // NOLINT
using namespace scannerpp;   // NOLINT
// using namespace TSP_fcore;
int main() {
  srand(0);  // using system random (weak... just an example!)

  // load data into problem context 'pTSP'
  Scanner scanner{"5\n1 10 10\n2 20 20\n3 30 30\n4 40 40\n5 50 50\n"};
  sref<ProblemContext> pTSP{new ProblemContext{}};
  pTSP->load(scanner);
  std::cout << pTSP->dist << std::endl;

  OptFrameDemoTSP demo{pTSP};

  // evaluator
  // TSPEval ev;
  //
  // create simple solution
  // TSPRandom crand;
  //
  std::vector<int> sol = *demo.randomConstructive->generateSolution(0);
  std::cout << sol << std::endl;

  // evaluation value and store on ESolution pair
  ESolutionTSP esol(sol, demo.eval->evaluate(sol));
```

(continues on next page)

(continued from previous page)

```
  esol.second.print();  // print evaluation

  // swap 0 with 1
  MoveSwap move{demo.pTSP, make_pair(0, 1), fApplySwap};
  move.print();

  // NSSwap nsswap;
  // move for solution 'esol'
  auto m1 = demo.nsSwap->randomMove(esol);
  m1->print();

  std::cout << std::endl;
  std::cout << "begin listing NSSeqSwapFancy" << std::endl;
  //
  auto it1 = demo.nsseqSwap->getIterator(esol);
  for (it1->first(); !it1->isDone(); it1->next()) it1->current()->print();
  std::cout << "end listing NSSeqSwapFancy" << std::endl;

  // Random number generator
  RandGen rg;                       // stack version
  sref<RandGen> rg2{new RandGen};  // heap version (safely shared)
  // testing simulated annealing
  BasicInitialSearch<ESolutionTSP> initRand(demo.randomConstructive, demo.eval);

  vsref<LocalSearch<ESolutionTSP>> ns_list;
  Evaluator<ESolutionTSP::first_type, ESolutionTSP::second_type>* ev2 =
      new FEvaluator<ESolutionTSP, MinOrMax::MINIMIZE, ProblemContext>{
          pTSP, fevaluate};
  GeneralEvaluator<ESolutionTSP>* gev2 = (GeneralEvaluator<ESolutionTSP>*)ev2;
  sref<GeneralEvaluator<ESolutionTSP>> eval2(gev2);
  ns_list.push_back(new BestImprovement<ESolutionTSP>(eval2, demo.nsseqSwap));

  VariableNeighborhoodDescent<ESolutionTSP> VND(demo.eval, ns_list);
  // VND.setVerbose();//
ILSLPerturbationLPlus2<ESolutionTSP> pert(demo.eval, demo.nsSwap, rg2);

IteratedLocalSearchLevels<ESolutionTSP> ils(demo.eval, initRand, VND, pert, 10,
                                            5);
// ils.setVerbose();

std::cout << "will start ILS for 3 seconds" << std::endl;

auto status = ils.search(
    StopCriteria<ESolutionTSP::second_type>{3.0});  // 3.0 seconds max
ESolutionTSP best = *status.best;
// best solution value
best.second.print();
std::cout << "solution: " << best.first << std::endl;

std::cout << "FINISHED" << std::endl;
return 0;
}
```

4.3 Search methods based on random keys

We finish with the Biased Random Key Genetic Algorithm (BRKGA), a simple metaheuristic inspired by classic Genetic Algorithm, using the solution representation of n Random Keys, which are $[0,1]^n$ float values.

4.3.1 Random key generation

The BRKGA requires an initial solution generator, which is in this case, n random [0,1] floats. This can be done automatically by the method (since its trivial do generate n [0,1] random numbers), but we choose to demonstrate manually (by inheriting from OptFrame Core class Initial Population).

This is good to tune the degree of randomness (number of random digits) and also the random function used.

```
class MyRandomKeysInitEPop
    : public InitialEPopulation<
          std::pair<std::vector<double>, Evaluation<int>>> {
  using RSK = std::vector<double>;

 private:
  int sz;
  sref<RandGen> rg;

 public:
  explicit MyRandomKeysInitEPop(int size, sref<RandGen> _rg = new RandGen)
      : sz{size}, rg{_rg} {}

  // copy constructor
  // MyRandomKeysInitEPop(const MyRandomKeysInitEPop& self)
  //     : sz{self.sz}, rg{self.rg} {}

  // this generator cannot evaluate solutions
  bool canEvaluate() const override { return false; }

  VEPopulation<std::pair<RSK, Evaluation<int>>> generateEPopulation(
      unsigned populationSize, double timelimit) override {
    VEPopulation<std::pair<RSK, Evaluation<int>>> pop;

    for (unsigned i = 0; i < populationSize; i++) {
      vector<double> vd(sz);
      for (int j = 0; j < sz; j++) vd[j] = (rg->rand() % 100000) / 100000.0;
      // assert(!this->canEvaluate());
      std::pair<RSK, Evaluation<int>> ind{vd, Evaluation<int>{}};
      pop.push_back(ind);
    }

    return pop;
  }
};
```

4.3.2 BRKGA decoding

BRKGA also requires a decoder function, that maps this array of random keys into a permutation.

This can be easily done with Functional Core class FDecodeRK, and an interesting approach based on sorting the keys, related to a predefined indexing of each key.

```
pair<Evaluation<int>, vector<int>> fDecodeEval(
    sref<Evaluator<typename ESolutionTSP::first_type,
                   typename ESolutionTSP::second_type, ESolutionTSP>>
        eval,
    const vector<double>& rk) {
  vector<pair<double, int>> v(rk.size());
  //
  for (unsigned i = 0; i < v.size(); i++) v[i] = pair<double, int>(rk[i], i);

  sort(v.begin(), v.end(),
       [](const pair<double, int>& i, const pair<double, int>& j) -> bool {
         return i.first < j.first;
       });

  // R = vector<int>
  vector<int> p(v.size());
  for (unsigned i = 0; i < v.size(); i++) p[i] = v[i].second;

  /*
  // ========== CHECKER ========
  vector<bool> vb(v.size(), false);
  for (unsigned i = 0; i < p.size(); i++)
     vb[p[i]] = true;
  for (unsigned i = 0; i < vb.size(); i++) {
     if (!vb[i]) {
        std::cout << "ERROR rk:" << rk << std::endl;
        std::cout << "ERROR v:" << v << std::endl;
        std::cout << "ERROR p:" << p << std::endl;
        std::cout << "ERROR vb:" << vb << std::endl;
     }
     assert(vb[i]);
  }
  // ===== end CHECKER =====
*/

  Evaluation<int> e = eval->evaluate(p);
  return make_pair(e, p);
}

// evaluator random keys (for TSP)
// FDecoderEvalRK<std::pair<std::vector<int>, Evaluation<int>>, double>
// decoder{fDecode};
```

4.3.3 BRKGA with TSP

We are ready to build a TSP instance with 3 cities with coordinates (10,10), (20,20) and (30,30), and invoke a BRKGA to solve it.

The parameters of BRKGA are: decoding function, initial solution generator, population size, number of iterations, also rates for mutation (randomness), elite (best solutions), preference for elite solutions, and finally, a random generation method.

```
int main() {
  sref<RandGen> rg = new RandGen;  // avoids weird windows OS interactions

  // load data into problem context 'pTSP'
  Scanner scanner{"5\n1 10 10\n2 20 20\n3 30 30\n4 40 40\n5 50 50\n"};
  sref<ProblemContext> pTSP{new ProblemContext{}};
  pTSP->load(scanner);
  std::cout << pTSP->dist << std::endl;

  OptFrameDemoTSP demo{pTSP};
  // setup decoder function
  demo.decoder = sptr<DecoderRandomKeys<ESolutionTSP, double>>{
      new FDecoderEvalRK<std::pair<std::vector<int>, Evaluation<int>>, double>{
          demo.eval, fDecodeEval}};

  // Parameters BRKGA
  // (C1): Evaluator<S, XEv>& _evaluator, int key_size, unsigned numGen,
  // unsigned _popSize, double fracTOP, double fracBOT, double _probElitism) :

  sref<DecoderRandomKeys<ESolutionTSP, double>> _decoder = demo.decoder;
  sref<InitialEPopulation<std::pair<vector<double>, ESolutionTSP::second_type>>>
      _initPop = new MyRandomKeysInitEPop(pTSP->n);  // passing key_size

  // eprk, pTSP.n, 1000, 30, 0.4, 0.3, 0.6
  BRKGA<ESolutionTSP, double> brkga(
      _decoder, MyRandomKeysInitEPop(pTSP->n, rg),  // key_size = pTSP.n
      30, 1000, 0.4, 0.3, 0.6, rg);

  auto searchOut = brkga.search(3.0);  // 3.0 seconds max
  ESolutionTSP best = *searchOut.best;
  // best solution value
  best.second.print();
  std::cout << "solution: " << best.first << std::endl;

  std::cout << "FINISHED" << std::endl;
  return 0;
}
```

The result from searchOut can be split in two parts, an error code and the returned solution (the same as in Simulated Annealing or any other OptFrame search method).

4.3.4 Complete Example for BRKGA

We provide the main file for TSP BRKGA `mainTSP-fcore-brkga.cpp`.

mainTSP-fcore-brkga.cpp

`File 'mainTSP-fcore-brkga.cpp' located in 'demo/03_QuickstartTSP_VNS_BRKGA/'`

```cpp
// SPDX-License-Identifier:  MIT OR LGPL-3.0-or-later
// Copyright (C) 2007-2022 - OptFrame developers
// https://github.com/optframe/optframe

// OptFrame Demo TSP - BRKGA

// C++
#include <iostream>
//
#include "TSP-fcore.hpp"
// implementation of TSP
//
#include <OptFrame/Core.hpp>
#include <OptFrame/Heuristics/EA/RK/BRKGA.hpp>
#include <OptFrame/Heuristics/Heuristics.hpp>  // many metaheuristics here...
#include <OptFrame/InitialPopulation.hpp>
#include <OptFrame/LocalSearch.hpp>

// import everything on main()
using namespace std;         // NOLINT
using namespace optframe;    // NOLINT
using namespace scannerpp;   // NOLINT
// using namespace TSP_fcore;

class MyRandomKeysInitEPop
    : public InitialEPopulation<
          std::pair<std::vector<double>, Evaluation<int>>> {
  using RSK = std::vector<double>;

 private:
  int sz;
  sref<RandGen> rg;

 public:
  explicit MyRandomKeysInitEPop(int size, sref<RandGen> _rg = new RandGen)
      : sz{size}, rg{_rg} {}

  // copy constructor
  // MyRandomKeysInitEPop(const MyRandomKeysInitEPop& self)
  //     : sz{self.sz}, rg{self.rg} {}

  // this generator cannot evaluate solutions
  bool canEvaluate() const override { return false; }

  VEPopulation<std::pair<RSK, Evaluation<int>>> generateEPopulation(
      unsigned populationSize, double timelimit) override {
```

(continues on next page)

(continued from previous page)

```
    VEPopulation<std::pair<RSK, Evaluation<int>>> pop;

    for (unsigned i = 0; i < populationSize; i++) {
      vector<double> vd(sz);
      for (int j = 0; j < sz; j++) vd[j] = (rg->rand() % 100000) / 100000.0;
      // assert(!this->canEvaluate());
      std::pair<RSK, Evaluation<int>> ind{vd, Evaluation<int>{}};
      pop.push_back(ind);
    }

    return pop;
  }
};


pair<Evaluation<int>, vector<int>> fDecodeEval(
    sref<Evaluator<typename ESolutionTSP::first_type,
                   typename ESolutionTSP::second_type, ESolutionTSP>>
        eval,
    const vector<double>& rk) {
  vector<pair<double, int>> v(rk.size());
  //
  for (unsigned i = 0; i < v.size(); i++) v[i] = pair<double, int>(rk[i], i);

  sort(v.begin(), v.end(),
       [](const pair<double, int>& i, const pair<double, int>& j) -> bool {
         return i.first < j.first;
       });

  // R = vector<int>
  vector<int> p(v.size());
  for (unsigned i = 0; i < v.size(); i++) p[i] = v[i].second;

  /*
  // ========== CHECKER ========
  vector<bool> vb(v.size(), false);
  for (unsigned i = 0; i < p.size(); i++)
     vb[p[i]] = true;
  for (unsigned i = 0; i < vb.size(); i++) {
     if (!vb[i]) {
        std::cout << "ERROR rk:" << rk << std::endl;
        std::cout << "ERROR v:" << v << std::endl;
        std::cout << "ERROR p:" << p << std::endl;
        std::cout << "ERROR vb:" << vb << std::endl;
     }
     assert(vb[i]);
  }
  // ===== end CHECKER =====
*/

  Evaluation<int> e = eval->evaluate(p);
  return make_pair(e, p);
```

(continues on next page)

(continued from previous page)

```
}

// evaluator random keys (for TSP)
// FDecoderEvalRK<std::pair<std::vector<int>, Evaluation<int>>, double>
// decoder{fDecode};
int main() {
  sref<RandGen> rg = new RandGen;  // avoids weird windows OS interactions

  // load data into problem context 'pTSP'
  Scanner scanner{"5\n1 10 10\n2 20 20\n3 30 30\n4 40 40\n5 50 50\n"};
  sref<ProblemContext> pTSP{new ProblemContext{}};
  pTSP->load(scanner);
  std::cout << pTSP->dist << std::endl;

  OptFrameDemoTSP demo{pTSP};
  // setup decoder function
  demo.decoder = sptr<DecoderRandomKeys<ESolutionTSP, double>>{
      new FDecoderEvalRK<std::pair<std::vector<int>, Evaluation<int>>, double>{
          demo.eval, fDecodeEval}};

  // Parameters BRKGA
  // (C1): Evaluator<S, XEv>& _evaluator, int key_size, unsigned numGen,
  // unsigned _popSize, double fracTOP, double fracBOT, double _probElitism) :

  sref<DecoderRandomKeys<ESolutionTSP, double>> _decoder = demo.decoder;
  sref<InitialEPopulation<std::pair<vector<double>, ESolutionTSP::second_type>>>
      _initPop = new MyRandomKeysInitEPop(pTSP->n);  // passing key_size

  // eprk, pTSP.n, 1000, 30, 0.4, 0.3, 0.6
  BRKGA<ESolutionTSP, double> brkga(
      _decoder, MyRandomKeysInitEPop(pTSP->n, rg),  // key_size = pTSP.n
      30, 1000, 0.4, 0.3, 0.6, rg);

  auto searchOut = brkga.search(3.0);  // 3.0 seconds max
  ESolutionTSP best = *searchOut.best;
  // best solution value
  best.second.print();
  std::cout << "solution: " << best.first << std::endl;

  std::cout << "FINISHED" << std::endl;
  return 0;
}
```

4.4 More Examples

For a other examples, see folder Examples/FCore-BRKGA and execute `bazel build ...`

> **Warning:** Feel free to check folder `OptFrame/Examples` for other examples on FCore and OptFrame Classic.

CHAPTER

FIVE

CONCEPTS

OptFrame is a framework for optimization, specially focused on heuristics and metaheuristics. Some works were published in literature with theoretical background, please visit OptFrame GitHub repository.

OptFrame supports many types of optimization strategies, such as mono and multi objective, and several different metaheuristic classes, such as trajectory and population based. Each of these techniques have different theories behind it, but OptFrame manages to put all of them together in a concise framework.

In this sense, we define a Search (or Exploration) Space in OptFrame as a pair composed by a Solution (or Representation) Space (called **XSolution**) and an Evaluation (or Objective) Space (called **XEvaluation**). Together, they form a **XESolution** space, also denoted by the pair **(XSolution,XEvaluation)**.

There are several types of search spaces, depending on the solution strategy employed. The **XESSolution** enforces that the Evaluation Space is of single-objective kind **XSEvaluation**, thus enforcing a *total order* over the *objType* behind the evaluation type.

Hint: By default, `XSEvaluation::objType` defaults to 64-bit floating point, known as *double* type. This, and other things can be changed (for example, to *int* or *long long*) if necessary!

So, **XESSolution** denotes the pair **(XSolution, XSEvaluation)**.

Hint: Most CXX Concepts in OptFrame begin with letter **X**, to not confuse with standard classes.

On the other hand, the **XEMSolution** space denotes the pair **(XSolution, XMEvaluation)**, where **XMEvaluation** requires multiple elements of *total order* to exist, like $2^{XEvaluation}$. This is very useful in multi-objective problems and it does not constrain the type of every individual **XEvaluation** element, as long as they individually form a *total order* (for example, one may be an *int*, while the other is a *double*).

Hint: On practice, basic implementations such as the *MultiEvaluation* assume a single type for `XSEvaluation::objType`, defaulting to 64-bit floating point, known as *double* type. This makes it simpler to code multi-objective metaheuristics, but does not exclude the possibility of adopting different types as a tuple, if necessary.

Some spaces contain a Power Set of other spaces, such as the **X2Solution**, which is equivalent to a set of solutions of type $2^{XSolution}$, and the **X2ESolution**, which is a set of **XESolution** pairs, like $2^{XESolution}$.

Hint: One can regard a **X2Solution** as a *population of individuals*, in Genetic Algorithm notation, and a **X2ESolution** as a *population of individuals together with their fitness values*.

Finally, a **X2EMSolution** denotes a set of **EMSolution** spaces, like $2^{XEMSolution}$.

Hint: One can regard a **X2EMSolution** as a *Pareto Set together with a Pareto Front*, in multi objective literature notation.

Using this process, one may define even more complex spaces, if necessary. For more details, check C++ file *BaseConcepts.hpp* (this requires C++20 CXX Concepts).

5.1 Domains

Besides the concepts presented before, OptFrame also supports Default Domains, that are practical implementations of the spaces described before. An interesting example is for BRKGA metaheuristic, that only allows individuals/solutions to be represented as a vector of 01-real numbers, called *random-keys*. So, in order to detect if a component is really compatible with other, it is necessary to check if those are operating under the same domain. By default, the Default Domain of a component does not appear in its name, only appearing (with angle backet notation) if the component is operating outside its default domain.

Function *getNamedDomain* on file *Domain.hpp* shows all available default domains in OptFrame:

```
template <typename X>
constexpr static std::string_view getNamedDomain() {
  if constexpr (is_rkf64<X>::value) return "<XRKf64>";
  if constexpr (is_rkf32<X>::value)
    return "<XRKf32>";
  else if constexpr (is_XRKf64Ef64<X>::value)
    return "<XRKf64Ef64>";
  else if constexpr (is_X2RKf64Ef64<X>::value)
    return "<X2RKf64Ef64>";
  else if constexpr (is_XESf64<X>::value)
    return "<XESf64>";
  else if constexpr (is_XESf32<X>::value)
    return "<XESf32>";
  else if constexpr (is_XESi64<X>::value)
    return "<XESi64>";
  else if constexpr (is_X2ESf64<X>::value)
    return "<X2ESf64>";
  else if constexpr (is_XMESf64<X>::value)
    return "<XMESf64>";
  else if constexpr (is_X2MESf64<X>::value)
    return "<X2MESf64>";
  else if constexpr (XESolution<X>)
    return "<XES>";
  else if constexpr (is_X2S<X>::value)
    return "<X2S>";
  else if constexpr (XSolution<X>)
    return "<XS>";
  return "";
}
```

Briefly, these are the supported default domains:

- "<XS>": only solution is supported (no evaluation)
- "<X2S>": only multi solutions (or population) is supported (no evaluation)

- "<XES>": both solution and evaluation is supported
- "<XESf64>": solution and evaluation with 64-bit floating-point is supported
- "<XESf32>": solution and evaluation with 32-bit floating-point is supported
- "<XESi64>": solution and evaluation with 64-bit integer is supported
- "<X2ESf64>": population of XESf64 elements
- "<XMESf64>": both solution and multi evaluation with 64-bit floating-point is supported
- "<X2MESf64>": population of XMESf64 elements
- "<XRKf32>": only solution of random-keys with 32-bit floating-point (no evaluation)
- "<XRKf64Ef64>": only solution of random-keys with 64-bit floating-point and also 64-bit evaluation
- "<X2RKf64Ef64>": only population of XRKf64Ef64 elements
- "<XRKf64EMi32>": only solution of random-keys with 64-bit floating-point and multiple 32-bit integer evaluation
- and so on...

This can (CERTAINLY) look confusing, so don't worry too much about it! It is not common to have these, specially for common mono objective optimization problems.

In the rare occasions where such domain appears in the name of a component, it will certainly help you observing the coherent interactions of different spaces/domain in a single unified framework. This is the greatest power of OptFrame!

CHAPTER

SIX

FCORE LIBRARY

FCore stands for OptFrame Functional Core, which is a simplified version of OptFrame Classic. FCore was introduced in OptFrame v4.1, with functional-style abstractions for mostly used OptFrame components. It is fully compatible with OptFrame classic, so users are able to interchange components (including all available metaheuristics).

6.1 Design Principles

The FCore library has the following design principles:

- 1) User will write the least possible number of lines of code for its components
- 2) User will never explicitly use class inheritance, while still being able to explore fundamental code optimization strategies
- 3) User will be able to execute any existing (meta)-heuristic, only using FCore

Principle (1) is **accomplished**, and it also allows us to break compatibility in future releases (as long as it simplifies the FCore) Principle (2) is **partially accomplished**, as Move costs are still not available (and considered fundamental) Principle (3) is **partially accomplished**, since GRASP, GA and other metaheuristics are still not accessible (without class inheritance)

FCore assumes largely supported compilers, at least for Windows and Linux, and for this reason, C++17 is the current standard.

6.2 Extended FCore: the FxCore

Some interesting programming techniques can help to fulfill Principle (1), but they require C++20 and newer/experimental features. For this reason, we provide FxCore library, as an Extended version of FCore. We also assume that users of FxCore are familiar with FCore, and for this reason, we may allow greater number of configurations for Principle (2). Users should be able to use the same number of metaheuristics, for both FCore and FxCore, regarding Principle (3).

6.3 Read more

Danger: This section is incomplete!

Project is released under LGPLv3 copyleft license.

CHAPTER

SEVEN

DEBUGGING

OptFrame provides several search methods that are ready to use, including classic metaheuristics. Users may feel the need for debug messages during experimentation, and for this reason, we provide a complete debug/messaging system.

Hint: Advanced users are recommended to write their own search methods, based on existing templates/examples. This allows having more control over parameters and execution flow, so as introducing state-of-the-art high performance techniques. See section "Custom Methods" for more information.

7.1 Logs/Message System

Every OptFrame component (including search methods) inherits from `Component` class (see file `Component.hpp`). This allows automatic support for two types of logs: *user logs* and *machine logs*.

7.1.1 User Logs

User logs are commonly used for understanding the methods, so as tracking possible warning and errors. There are five different levels (error, warning, information, verbose, debug):

- error: execution errors to log stream (this is different from exception messaging on `cerr`, which is typically disabled by `-fno-exceptions`)
- warning: execution warnings to log stream
- information: typical informative messages (DEFAULT log level)
- verbose: excessive messaging (typically for debug purposes)
- debug: same as verbose, but completely disabled/removed when `NDEBUG` flag is active (good for performance)

Typically, *information* level is enabled, presenting restricted (but useful) search-related information. Examples are: best solution improvements (only value is presented); initial/final results of search method.

Hint: When designing new methods, avoid printing complete solutions, only print its value (or values, when multi-objective). The reason is that solution structures tend to get bigger and complex to read, being mostly useless for debug purposes. For recording solutions, one may easily customize event callbacks (see section "Custom Callbacks").

To activate verbose/debug logs recursively (from a component into all its sub-components), use method `setVerboseR()`.

```
auto mySearchMethod = ...;    // maybe a Simulated Annealing?
mySearchMethod.setVerboseR(); // sets verbose/debug level to all sub-components
// check that debug level is activated (will print 'debug=1')
std::cout << "debug=" << mySearchMethod.debug << std::endl;
```

To activate a specific log level, to some specific component, just use method `setMessageLevel(...)`:

```
auto mySearchMethod = ...;    // maybe a Simulated Annealing?
mySearchMethod.setMessageLevel(LogLevel::Silent); // no logs
// check that debug level is inactive (will print 'debug=0')
std::cout << "debug=" << mySearchMethod.debug << std::endl;
```

On the other hand, one may want to completely disable all logs for all sub-components (maybe some script execution?):

```
auto mySearchMethod = ...;    // maybe a Simulated Annealing?
mySearchMethod.setSilentR();  // no logs for component and all sub-components
// check that debug level is inactive (will print 'debug=0')
std::cout << "debug=" << mySearchMethod.debug << std::endl;
```

User may also choose the stream for logs (DEFAULT is `std::cout`):

```
auto mySearchMethod = ...;          // maybe a Simulated Annealing?
mySearchMethod.setLogR(&std::cout); // log stream for component and all sub-components
```

Hint: Fine-grained components should not provide information logs, otherwise log system could be quickly overloaded. User is welcome to implement its own logs (using stream `(*Component::logdata)`) during development of its components.

Warning: Not all search components are currently designed to support all suitable message types and custom log stream. Some messages may currently leak through `std::cout` or `std::cerr` (feel free to open an Issue or PR in github).

7.1.2 Machine Logs

An interesting feature of OptFrame components is support for structured (context-aware) machine logs. Typical machine logs are: list of best values and iteration/time when it was found; launch configuration of search method.

User may push machine logs through standard streams (such as `std::cout`) or specific files. However, it is useful to add some *context* to the logging, typically *plain text* or *json*.

Warning: Machine logs are disabled by default.

```
auto mySearchMethod = ...;          // maybe a Simulated Annealing?
std::ofstream fout("output.txt");   // creates 'output.txt' file
// send 'ctxt' semantic stream outputs to designated file
optframe::ctxt.setStream(fout);     // redirects 'ctxt' plain text stream to file
// sets machine logs through plain text stream
mySearchMethod.mlog = &optframe::ctxt;
```

Hint: User can also select json context stream `optframe::cjson` as output. There are also plans to support CSV format as `optframe::ccsv`, but we need some help on that...

Warning: Context log streams depend on each implementation of the search methods/components. Currenty, few methods support several context standards, so *plain text* should be preferred (at first!).

7.2 Custom Callbacks

There are two interesting callbacks to customize search methods:

- onBest: activated when a new *best* is found
- onIncumbent: activated when *incument* has changed

We intentionally avoid the word "solution" here, since "best" may be some "best solution" or "best Pareto Set" for multi-objective. This is the same for "incumbent", as it may be "incumbent solution", "incumbent population" or something else.

Here's an example for *onBest*, for some single-objective trajectory-based method:

```
auto mySearchMethod = ...;          // maybe a Simulated Annealing?
mySearchMethod.onBest = [](auto& self)
{
    // logging solution to user logs
    (*self.logdata) << "My method has improved best solution! Print solution:\t" << self.
↪best->first << std::endl;
    // logging solution to machine logs
    (*self.mlog) << "solution\t" << self.best->first << std::endl;
    // fine-tuning stop criteria according to some solution characteristic (or objective␣
↪value)
    return self.best->second.evaluation() > 9500.0; // halts if less or equals to 9500.0␣
↪(minimization problem)
};
```

Callbacks allow search to be stopped (thus also acting as a custom stop criteria): returning false should halt execution.

Danger: This section is incomplete.

7.3 Custom Methods

User may implement a new search method by inheriting from `GlobalSearch` class and implementing `search` method.

Danger: This section is incomplete.

CHAPTER

EIGHT

ADVANCED

Some advanced tricks and optimization techniques.

Danger: This section is incomplete!

8.1 Explicit XESolution

Instead of using standard `std::pair` to glue solution and evaluation types, one can explictly declare a class. This is useful to embed user methods and other problem-specific operations.

```
// explicit declaration of class to represent XESolution pair
class MyESolution1
{
public:
    using first_type = std::vector<bool>;
    using second_type = Evaluation<double>;
    first_type first;   // some XSolution type
    second_type second; // some XEvaluation type
    // other specific methods ...
}

// declaration of XESolution pair via inheritance over IESolution
class MyESolution2 : public IESolution<std::vector<bool>, Evaluation<double>>
{
    // other specific methods ...
}
```

8.2 Local Search

OptFrame has three main types of neighborhoods: `NS`, `NSSeq` and `NSEnum`.

Advanced local search, ...

8.3 Move Cost

Hint: On OptFrame Classic (not FCore) it's possible to avoid the creation of an undo move. Currently, all existing examples on OptFrame Classic use the undo feature anyway, since move undo is typically much faster than a complete copy of the solution structure.

8.4 Multi-Objective

For multiple objectives, ...

CHAPTER

NINE

ADVANCED - CHECKMODULE

We expand the knowledge from Quick Start where the user has learned how to Install OptFrame, and how to compile and test metaheuristics for classic 0-1 Knapsack Problem (01KP) and Traveling Salesman Problem (TSP).

We now demonstrate how to quickly test the code and provide better move evaluations.

9.1 Multiple ways to evaluate a move

Warning: At this point, we assume the reader is familiarized with the Traveling Salesman Problem, and is also familiarized with basic workings of OptFrame and OptFrame Functional Core.

We recall the example for the TSP, where a move operation called MoveSwap is proposed.

```
std::pair<int, int> fApplySwap(sref<ProblemContext>,
                               const std::pair<int, int>& moveData,
                               ESolutionTSP& se) {
  int i = moveData.first;
  int j = moveData.second;
  // perform swap of clients i and j
  int aux = se.first[j];
  se.first[j] = se.first[i];
  se.first[i] = aux;
  return std::pair<int, int>(j, i);  // return a reverse move ('undo' move)s
}

// Swap move
using MoveSwap = FMoveP<std::pair<int, int>, ESolutionTSP, ProblemContext>;

uptr<Move<ESolutionTSP>> makeMoveSwap(sref<ProblemContext> p, int i, int j) {
  return uptr<Move<ESolutionTSP>>(new MoveSwap{p, make_pair(i, j), fApplySwap});
}
```

9.1.1 Basic move cost calculation (revCost)

Typically, any basic move implementation will require at least one extra evaluation (on objective space) and two move applications (called revCost):

1. First apply move it to current solution, becoming a neighbor solution (prevent solution copy)
2. generating an Undo Move
3. evaluating the cost at the neighbor solution
4. applying the Undo Move and come back to original solution
5. update objective value (by numeric copy)

```
class MoveSwap : public Move<ESolutionTSP> {
 public:
  int i, j;

  MoveSwap(int _i, int _j) : i{_i}, j{_j} {}

  bool canBeApplied(const ESolutionTSP& se) override {
    return (::abs(i - j) >= 2) && (i >= 1) && (j >= 1);
  }

  uptr<Move<ESolutionTSP>> apply(ESolutionTSP& se) override {
    // perform swap of clients i and j
    int aux = se.first[j];
    se.first[j] = se.first[i];
    se.first[i] = aux;
    return uptr<Move<ESolutionTSP>>(
        new MoveSwap{j, i});  // return a reverse move ('undo' move)s
  }

  bool operator==(const Move<ESolutionTSP>& other) const override {
    auto& fmove = (MoveSwap&)other;
    return (i == fmove.i) && (j == fmove.j);
  }
};

uptr<Move<ESolutionTSP>> makeMoveSwap(int i, int j) {
  return uptr<Move<ESolutionTSP>>(new MoveSwap{i, j});
}
```

Important: We have included the filter `canBeApplied` to limit the scope of valid moves, so as an `operator==` to check for equality between moves.

9.1.2 Faster move cost calculation (fasterCost)

Step 3 from revCost is typically costly, what may be prevented by directly updating objective value at the same time the move is applied to solution. This can be done by adopting the `applyUpdate` operation on Move implementation. Usually, this can be done by either:

1. Precomputing the cost value before move apply, and then apply it
2. Apply the move and then directly compute the updated objective value

```
class MoveSwap : public Move<ESolutionTSP> {
 public:
  sref<ProblemContext> pTSP;
  int i, j;

  MoveSwap(sref<ProblemContext> _p, int _i, int _j) : pTSP{_p}, i{_i}, j{_j} {}

  bool canBeApplied(const ESolutionTSP& se) override {
    return (::abs(i - j) >= 2) && (i >= 1) && (j >= 1);
  }

  uptr<Move<ESolutionTSP>> applyUpdate(ESolutionTSP& se) override {
    // input cannot be outdated
    assert(!se.second.isOutdated());
    auto& s = se.first;
    int diff =
        -pTSP->dist(s[i - 1], s[i]) - pTSP->dist(s[i], s[(i + 1) % pTSP->n]) -
        pTSP->dist(s[j - 1], s[j]) - pTSP->dist(s[j], s[(j + 1) % pTSP->n]);
    diff += pTSP->dist(s[i - 1], s[j]) +
            pTSP->dist(s[j], s[(i + 1) % pTSP->n]) +
            pTSP->dist(s[j - 1], s[i]) + pTSP->dist(s[i], s[(j + 1) % pTSP->n]);
    // solution swap
    auto rev = this->apply(se);
    se.second.setObjFunction(se.second.evaluation() + diff);
    return rev;
  }

  uptr<Move<ESolutionTSP>> apply(ESolutionTSP& se) override {
    // perform swap of clients i and j
    int aux = se.first[j];
    se.first[j] = se.first[i];
    se.first[i] = aux;
    return uptr<Move<ESolutionTSP>>(
        new MoveSwap{pTSP, j, i});  // return a reverse move ('undo' move)s
  }

  bool operator==(const Move<ESolutionTSP>& other) const override {
    auto& fmove = (MoveSwap&)other;
    return (i == fmove.i) && (j == fmove.j);
  }
};

uptr<Move<ESolutionTSP>> makeMoveSwap(sref<ProblemContext> p, int i, int j) {
  return uptr<Move<ESolutionTSP>>(new MoveSwap{p, i, j});
}
```

9.1.3 Constant cost calculation (cost)

Finally, an even faster approach is not directly compute cost of move, without even applying it to the solution. The method `cost()` may optionally return a calculated cost, if user wants to use that.

We observe that it is important to have some precise numerics here, such as `Evaluation<int>`, instead of `Evaluation<double>`. Cost recalculation may impose calculation errors (but if user wants to do it anyway, OptFrame will not forbid it).

```
class MoveSwap : public Move<ESolutionTSP> {
 public:
  sref<ProblemContext> pTSP;
  int i, j;

  MoveSwap(sref<ProblemContext> _p, int _i, int _j) : pTSP{_p}, i{_i}, j{_j} {}

  bool canBeApplied(const ESolutionTSP& se) override {
    return (::abs(i - j) >= 2) && (i >= 1) && (j >= 1);
  }

  uptr<Move<ESolutionTSP>> applyUpdate(ESolutionTSP& se) override {
    // input cannot be outdated
    assert(!se.second.isOutdated());
    auto& s = se.first;
    int diff =
        -pTSP->dist(s[i - 1], s[i]) - pTSP->dist(s[i], s[(i + 1) % pTSP->n]) -
        pTSP->dist(s[j - 1], s[j]) - pTSP->dist(s[j], s[(j + 1) % pTSP->n]);
    diff += pTSP->dist(s[i - 1], s[j]) +
            pTSP->dist(s[j], s[(i + 1) % pTSP->n]) +
            pTSP->dist(s[j - 1], s[i]) + pTSP->dist(s[i], s[(j + 1) % pTSP->n]);
    // solution swap
    auto rev = this->apply(se);
    se.second.setObjFunction(se.second.evaluation() + diff);
    return rev;
  }

  uptr<Move<ESolutionTSP>> apply(ESolutionTSP& se) override {
    // perform swap of clients i and j
    int aux = se.first[j];
    se.first[j] = se.first[i];
    se.first[i] = aux;
    return uptr<Move<ESolutionTSP>>(
        new MoveSwap{pTSP, j, i});  // return a reverse move ('undo' move)s
  }

  virtual op<Evaluation<int>> cost(const ESolutionTSP& se,
                                   bool allowEstimated) override {
    assert(!se.second.isOutdated());
    auto& s = se.first;
    int diff =
        -pTSP->dist(s[i - 1], s[i]) - pTSP->dist(s[i], s[(i + 1) % pTSP->n]) -
        pTSP->dist(s[j - 1], s[j]) - pTSP->dist(s[j], s[(j + 1) % pTSP->n]);
    diff += pTSP->dist(s[i - 1], s[j]) +
            pTSP->dist(s[j], s[(i + 1) % pTSP->n]) +
```

(continues on next page)

(continued from previous page)

```
            pTSP->dist(s[j - 1], s[i]) + pTSP->dist(s[i], s[(j + 1) % pTSP->n]);
    return std::make_optional(Evaluation<int>(diff));
  }

  bool operator==(const Move<ESolutionTSP>& other) const override {
    auto& fmove = (MoveSwap&)other;
    return (i == fmove.i) && (j == fmove.j);
  }
};

uptr<Move<ESolutionTSP>> makeMoveSwap(sref<ProblemContext> p, int i, int j) {
  return uptr<Move<ESolutionTSP>>(new MoveSwap{p, i, j});
}
```

9.2 Using the CheckCommand to test structures

One can easily check for the integrity of the structures, by creating an instance of CheckCommand.

First, create it, then add all available components (NS, NSSeq, Evaluator, Constructive, ...). Finally, invoke it passing two parameters: randomized pressure and iterative pressure.

```
// SPDX-License-Identifier: LGPL-3.0-or-later OR MIT
// Copyright (C) 2007-2022 - OptFrame - https://github.com/optframe/optframe

#include <iostream>
//
#include <OptFrame/Core.hpp>
#include <OptFrame/Heuristics/Heuristics.hpp>  // many metaheuristics here...
#include <OptFrame/Heuristics/ILS/IteratedLocalSearchLevels.hpp>
#include <OptFrame/Heuristics/LocalSearches/BestImprovement.hpp>
#include <OptFrame/Heuristics/LocalSearches/VariableNeighborhoodDescent.hpp>
#include <OptFrame/LocalSearch.hpp>
#include <OptFrame/Util/CheckCommand.hpp>
//
#include "TSP-fcore.hpp"  // NOLINT
// implementation of TSP

// import everything on main()
using namespace std;
using namespace optframe;
using namespace scannerpp;
// using namespace TSP_fcore;
int main() {
  std::stringstream ss;
  srand(1000000);
  int N = 50;
  ss << N << std::endl;
  for (unsigned i = 0; i < N; i++)
    ss << i << "\t" << rand() % 1000 << "\t" << rand() % 1000 << std::endl;
  // Scanner scanner{ std::string(instance5) };
  Scanner scanner{ss.str()};
```

(continues on next page)

(continued from previous page)

```
  sref<ProblemContext> pTSP{new ProblemContext{}};
  pTSP->load(scanner);

  // REQUIRE(pTSP.n == 5);
  assert(pTSP->n == 50);

  // set random seed for std::random_shuffle
  srand(1000000);

  FConstructive<std::vector<int>, ProblemContext> crand{pTSP, frandom};
  FEvaluator<ESolutionTSP, MinOrMax::MINIMIZE, ProblemContext> eval{pTSP,
                                                                    fevaluate};
  FNS<ESolutionTSP, ProblemContext> nsswap{pTSP, fRandomSwap};
  sref<FNSSeq<std::pair<int, int>, ESolutionTSP, ProblemContext>> nsseq2{
      make_nsseq(pTSP)};

  sref<InitialSearch<ESolutionTSP>> initRand{
      new BasicInitialSearch<ESolutionTSP>(crand, eval)};

  CheckCommand<ESolutionTSP> check(false);  // verbose
  //
  check.addEvaluator(eval);
  check.add(initRand);
  check.addNS(nsswap);     // NS
  check.addNSSeq(nsseq2);  // NSSeq

  // bool run(int iterMax, int nSolNSSeq)
  check.run(100, 10);
  //
  return 0;
}
```

An example of output can be seen here:

Solution Time to clone a solution

---

title #tests avg(ms) std(ms)
Solution 59502 0.0009 0.0014

---

all - 0.0009 -

|Constructive|=1 testing construction of initial solution

#id title #tests avg(ms) std(ms)
#0 OptFrame:InitialSearch 100 0.0181 0.0334

---

all * - 0.0181 -

|Evaluators|=1 testing full evaluate(s) of a solution

#id title #tests avg(ms) std(ms)
#0 Direction:MIN 59602 0.0013 0.0008

---

all * - 0.0013 -

|NS|=2 testing time of move apply(s) [apply_no_evaluation]

#id title #tests avg(ms) std(ms)
#0 OptFrame:FNS 41048 0.0008 0.0015
#1 OptFrame:FNSSeq 18454 0.0007 0.0008

---

all * - 0.0008 -

|NS|=2 testing time of cost based on move apply(s) [revCost]

#id title #tests avg(ms) std(ms)
#0 OptFrame:FNS 20524 0.0054 0.0033
#1 OptFrame:FNSSeq 9227 0.0051 0.0018

---

all * - 0.0053 -

|NS|=2 testing time of cost based on move apply(e, s) [fasterCost]

#id title #tests avg(ms) std(ms)
#0 OptFrame:FNS 20524 0.0020 0.0019
#1 OptFrame:FNSSeq 9227 0.0019 0.0011

---

all * - 0.0020 -

|NS|=2 testing time of real cost based on move apply(e, s) - forcing allowCosts to False [realFasterCost]

#id title #tests avg(ms) std(ms)
#0 OptFrame:FNS 20524 0.0015 0.0029
#1 OptFrame:FNSSeq 9227 0.0013 0.0006

---

all * - 0.0014 -

|NS|=2 testing time of move cost()

#id title #tests avg(ms) std(ms)
#0 OptFrame:FNS 20524 0.0008 0.0006
#1 OptFrame:FNSSeq 9227 0.0008 0.0007

---

all * - 0.0008 -

Note that cost() version is faster than others, for both neighborhoods (NS and NSSeq), as expected.

9.2.1 Error codes on CheckCommand

CheckCommand may launch several errors, according to the table below:

```
static const int CMERR_EV_BETTERTHAN = 2001;
static const int CMERR_EV_BETTEREQUALS = 2002;
static const int CMERR_MOVE_EQUALS = 3001;
static const int CMERR_MOVE_HASREVERSE = 3002;
static const int CMERR_MOVE_REVREV_VALUE = 3004;
static const int CMERR_MOVE_REVSIMPLE = 3005;
static const int CMERR_MOVE_REVFASTER = 3006;
static const int CMERR_MOVE_REALREVFASTER = 3008;
static const int CMERR_MOVE_COST = 3007;
```

Feel free to file an Issue on OptFrame Project in order to discuss how to handle them.

In the future, we intend to expand this section.

9.2.2 More Tricks on CheckCommand

For a other examples, see folder Examples/FCore-BRKGA and execute `bazel build ...`

> **Warning:** Feel free to check folder `OptFrame/Examples` for other examples on FCore and OptFrame Classic.

CHAPTER

TEN

HEURISTICS MODULES

The Heuristics package from OptFrame provides several implementations of classic heuristics and metaheuristics.

To build a metaheuristic, one needs to pass problem-specific components, either directly in C++ constructor or by using a component builder string syntax. For every proposed metaheuristic, there are one or more component builders, so feel free to explore them. On extended implementations of OptFrame, such as OptFrame Python (a.k.a, pyoptframe), component builders are mandatory.

Please see:

10.1 Simulated Annealing metaheuristic

10.1.1 General Concepts and Theory

Simulated Annealing is a popular metaheuristic, proposed by Kirkpatrick et al. in 1983:

```
S. Kirkpatrick, D.C. Gellat, and M.P. Vecchi. Optimization by Simulated
Annealing. Science, 220:671-680, 1983.
```

To learn more, we recommend the following classic books:

- Handbook of Metaheuristics
- Metaheuristics: from Design to Implementation

In Portuguese, we recommend the textbook by prof. Marcone Jamilson Freitas Souza.

10.1.2 Simulated Annealing pseudocode

1. Classic Definition in Literature

Parameters

- $f(.)$: objective function (minimization)
- $\mathcal{N}(.)$: neighborhood structure
- α: cooling factor
- $SAmax$: max number of iterations per temperature
- Ti: initial temperature
- $\xi(.)$: random number generator

- $stop(time = \infty, target = -\infty)$: general stop criteria for timelimit and target quality
- s: initial solution

Pseudocode for Classic Version in Literature

10.1.3 BasicSimulatedAnnealing

1. Main Definition

Parameters

- $stop(.)$: reference to StopCriteria component
- $g(.)$: shared reference to GeneralEvaluator component
- $initsol()$: shared reference to InitialSearch component
- $\mathcal{N}_k(.)$: list of shared references to NS component
- α: cooling factor (double)
- $SAmax$: max number of iterations per temperature (int)
- Ti: initial temperature (double)
- $\xi(.)$: shared reference to RandGen component

Pseudocode for Main Version

SearchStatus return codes

There are return codes being currently used: $NO\_SOLUTION$, $EARLY\_STOP$ and NO_REPORT. The return $EARLY\_STOP$ will trigger warnings.

Primary and Secondary search spaces

BasicSimulatedAnnealing is a *trajectory-based single objective global search* method:

- The primary search space (best type) XSH is *XESSolution*, where its base type XES is also *XESSolution*.
- The secondary search space (incumbent type) XSH2 is *XESSolution*, where its base type XES2 is also *XESSolution*.

This occurs since BasicSimulatedAnnealing inherits from SingleObjSearch, that constraints its *XESolution* space for single objective *XESSolution*, and also ITrajectory, that requires XSH=XSH2.

To better understand these notations, see Concepts

Algorithm 1 BasicSimulatedAnnealing

procedure BasicSimulatedAnnealing($stop(.)$, $g(.)$, $initsol()$, $\mathcal{N}_k(.)$, α, $SAmax$, Ti, $\xi(.)$)
  $\langle s, e\rangle \leftarrow initsol(stop)$
  if $\nexists \langle s, e\rangle$ **then**
    return $NO\_SOLUTION, \langle\rangle$
  end if
  $\langle s^*, e^*\rangle \leftarrow \langle s, e\rangle$
  $T \leftarrow Ti$
  $iterT \leftarrow 0$
  while $T \geq 0.0001$ **and not** $stop(e^*)$ **do**
    $j \leftarrow \xi^{\mathbb{Z}}(0, k-1)$
    $m \leftarrow \mathcal{N}_j^{ANY}(\langle s, e\rangle)$
    if $\nexists m$ **then**
      return $EARLY\_STOP, \langle s^*, e^*\rangle$
    end if
    $\langle s_1, e_1\rangle \leftarrow \langle s, e\rangle$
    $\langle s_1', e_1^\circ\rangle, \bar{m} \leftarrow m \oplus \langle s_1, e_1\rangle$
    $\langle s_1', e_1'\rangle \leftarrow g(\langle s_1', e_1^\circ\rangle)$
    if $g_<(e_1', e_1)$ **then**
      $\langle s, e\rangle \leftarrow \langle s_1', e_1'\rangle$
      if $g_<(e, e^*)$ **then**
        $\langle s^*, e^*\rangle \leftarrow \langle s, e\rangle$
      end if
    else
      $x \leftarrow \xi^{\mathbb{R}}(0, 1)$
      $\Delta \leftarrow |e_1' - e|$
      if $x < e^{\frac{-\Delta}{T}}$ **then**
        $\langle s, e\rangle \leftarrow \langle s_1', e_1'\rangle$
      end if
    end if
    if $iterT < SAmax$ **then**
      $iterT \leftarrow iterT + 1$
    else
      $iterT \leftarrow 0$
      $T \leftarrow \alpha \cdot T$
    end if
  end while
  return $NO\_REPORT, \langle s^*, e^*\rangle$
end procedure

Primary ComponentBuilder string syntax

One may build BasicSimulatedAnnealing on C++ by using its constructors from *BasicSimulatedAnnealing.hpp* header file.

It belongs to **SA** family and its Component Builder inherits from *GlobalSearchBuilder*, so a common way to find it (e.g. in OptFrame Python), is to use:

```
your_problem.engine.list_builders(":BasicSA")
```

The component builder string identifier for BasicSimulatedAnnealing is:

```
"OptFrame:ComponentBuilder:GlobalSearch:SA:BasicSA"
```

Expected arguments are:

```
OptFrame:ComponentBuilder:GlobalSearch:SA:BasicSA |params|=6
    param 0 => OptFrame:GeneralEvaluator:Evaluator : evaluation function
    param 1 => OptFrame:InitialSearch : constructive heuristic
    param 2 => OptFrame:NS[] : list of NS
    param 3 => OptFrame:double : cooling factor
    param 4 => OptFrame:int : number of iterations for each temperature
    param 5 => OptFrame:double : initial temperature
```

The **Default Domain** for BasicSimulatedAnnealing component is "<XESf64>" (single solutions on search space with 64 bits floating-point on objective space), as inherited from *GlobalSearch* and *SingleObjSearch*.

Example of string syntax

A simple example could be:

```
"OptFrame:GeneralEvaluator:Evaluator 0 OptFrame:InitialSearch 0 OptFrame:NS[] 0 0.98␣
↪1000 999999"
```

See Examples folder for real examples on C++ and OptFrame Python examples for using component builder string syntax.

1. Helpers

Simulated Annealing family includes a special method to estimate the initial temperature `estimateInitialTemperature`.

This method is found in textbook by prof. Marcone Jamilson Freitas Souza (In Portuguese).

3. Extended Versions and Callbacks

One may build extended versions of BasicSimulatedAnnealing, by configuring its callbacks and using alternative component builders.

SearchContext

BasicSimulatedAnnealing defines a SearchContext called SearchContextSA, with the following data:

- *BasicSimulatedAnnealing<XES>& self`*: reference to self (to get parameters)
- *double T*: current temperature
- *int iterT*: current iteration (per temperature)

Must double check these in the future (unstable to use):

- *std::optional<XES>& best*: reference to best solution, if exists
- *std::optional<XES>& incumbent*: reference to incumbent solution, if exists

BasicSimulatedAnnealing allows manipulation of its SearchContextSA in callbacks, in order to change/personalize its search behavior.

Pseudocode for Extended Version

The pseudocode below details the extension possibilities on BasicSimulatedAnnealing.

Callbacks

There are four **generic callbacks** available on extended versions of simulated annealing:

- *onBest*: from GlobalSearch
- *onIncumbent*: from ITrajectory
- *onLoop*: from ILoop
- *onBeforeLoop*: from ILoop

The *onBest* and *onIncumbent* are generic callbacks that work on current solution. The *onLoop* and *onBeforeLoop* from ILoop can be better explored as *specific callbacks*.

The are four **specific callbacks** implemented: *onBestCtx*, *onIncumbentCtx*, *onLoopCtx* and *onBeforeLoopCtx*.

By overriding *onLoopCtx* and *onBeforeLoopCtx* one may manipulate SearchContextSA, for example, to implement alternative cooling schemes for Simulated Annealing.

Algorithm 2 BasicSimulatedAnnealingCallbacks

procedure BasicSimulatedAnnealingCallbacks($stop(.)$, $g(.)$, $initsol()$, $\mathcal{N}_k(.)$, Ti, $\xi(.)$, $onBest(.)$, $onIncumbent(.)$, $onLoop(.)$, $onBeforeLoop(.)$)

$\langle s, e\rangle \leftarrow initsol(stop)$
if $\nexists \langle s, e\rangle$ **then**
return $NO\_SOLUTION, \langle\rangle$
end if
$onIncumbent(\langle s, e\rangle)$
$\langle s^*, e^*\rangle \leftarrow \langle s, e\rangle$
$onBest(\langle s^*, e^*\rangle)$
$context.T \leftarrow Ti$
$context.iterT \leftarrow 0$
while $onLoop(context, stop)$ **do**
$j \leftarrow \xi^{\mathbb{Z}}(0, k-1)$
$m \leftarrow \mathcal{N}_j^{ANY}(\langle s, e\rangle)$
if $\nexists m$ **then**
return $EARLY\_STOP, \langle s^*, e^*\rangle$
end if
$\langle s_1, e_1\rangle \leftarrow \langle s, e\rangle$
$\langle s'_1, e^\circ_1\rangle, \bar{m} \leftarrow m \oplus \langle s_1, e_1\rangle$
$\langle s'_1, e'_1\rangle \leftarrow g(\langle s'_1, e^\circ_1\rangle)$
if $g_<(e'_1, e_1)$ **then**
$\langle s, e\rangle \leftarrow \langle s'_1, e'_1\rangle$
$onIncumbent(\langle s, e\rangle)$
if $g_<(e, e^*)$ **then**
$\langle s^*, e^*\rangle \leftarrow \langle s, e\rangle$
$onBest(\langle s^*, e^*\rangle)$
end if
else
$x \leftarrow \xi^{\mathbb{R}}(0, 1)$
$\Delta \leftarrow |e'_1 - e|$
if $x < e^{\frac{-\Delta}{T}}$ **then**
$\langle s, e\rangle \leftarrow \langle s'_1, e'_1\rangle$
$onIncumbent(\langle s, e\rangle)$
end if
end if
$context \leftarrow onBeforeLoop(context)$
end while
return $NO\_REPORT, \langle s^*, e^*\rangle$

end procedure

Alternative Parameters

Some possibilities may appear only in C++ constructors, such as passing a single neighborhood instead of a list.

Important: The **searchBy** method inherited from *GlobalSearch* allows directly passing a primary XESolution element, thus *overriding the initsol() component*.

> **Warning:** This section is still incomplete!

> **Danger:** This section is incomplete!

CHAPTER

ELEVEN

LICENSE

Project is released under LGPLv3 copyleft license.

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search