Using HyFlex and irace for Automatic Configuration of Hyper-heuristics: Detailed Instructions

Weiyao Meng

In this challenge, your goal is to enhance the performance of provided hyper-heuristics for solving travelling salesman problems (TSP) by leveraging the flexibility and capabilities of HyFlex and irace.

HyFlex (Hyper-heuristics Flexible framework) [BCH+09] is a Java object-oriented benchmark framework for the design and analysis of hyper-heuristics. Hyper-heuristics can be seamlessly designed and implemented within HyFlex using its provided heuristic algorithms and functionalities, and rigorously tested across the predefined problem domains within HyFlex.

Some heuristic algorithms provided in HyFlex involve two parameters, i.e., '*intensity of mutation*' and the '*depth of search*' (denoted as '*i*' and '*d*' with $0 \leq [i, d] \leq 1$). These two parameters control the behaviour of the related search operators.

Notably, the configuration settings of these parameters can be intelligently managed through irace [LIDLC+16], a tool specialised in automatic parameter configuration.

To initiate this challenge, we provide irace configuration scenario files tailored for the automatic configuration of 'i' and 'd' parameters within the provided hyper-heuristics.

This instruction provides step-by-step guidance on how to utilise the provided resources effectively. You will learn how to:

- Set up HyFlex and irace.
- Execute the provided hyper-heuristics within HyFlex.
- Use irace to automatically configure the parameters of the provided hyper-heuristics.

Contents

Download

Not this one. Everything they need is already in the folder.

Download the materials from here: https://github.com/weiyaomeng

The provided resources include:

- project-hyflex: the source code of the provided example hyper-heuristics and the HyFlex framework (`chesc-fixed-no-ps.jar`).
- project-irace: the files of an example configuration scenario of using irace.

Setup

Setting up HyFlex

HyFlex can either be run using Java from the command line or by importing the files into a Java IDE. For simplicity, we use the Eclipse in this tutorial.

1. **Installing Java:** https://www.java.com/en/download/help/download_options.html
2. **Installing Eclipse:** https://www.eclipse.org/downloads/
3. **Importing resources in 'project-hyflex' into Eclipse**
   - In Eclipse, navigate to **File** → **Import** → **General** → **Existing Projects into Workspace**.
4. **Adding HyFlex JAR file to classpath**
   - In Eclipse, right-click the project folder in the **Project Explorer** on the left side of the screen.
   - Select **Build Path** → **Add External Archives**.
   - Navigate to the location of the HyFlex JAR file and select it.
   - Click **Open** to add the JAR file to your project's classpath.
5. **Verifying HyFlex is set up correctly**
   - In Eclipse, find the example program `examples.example.ExampleRun1.java` in your project.
   - Compile and run `ExampleRun1`.

If HyFlex is setup properly, you will observe the expected output in the console:

```
Algorithm: Example Hyper Heuristic One
Problem instance: class travelingSalesmanProblem.TSP
Time limit set to: 60 seconds
Search ...
```

Setting up irace

Follow the **Setup** instructions in https://lopez-ibanez.eu/sigevo-summer-school-2023/.
For simplicity, we use the Rstudio console for the rest of the tutorial.

Part 1: Executing Hyper-heuristics

When setting up HyFlex, you have attempted to run the `ExampleRun1` class. This class demonstrates the minimum that must be done to test a hyper-heuristic on a problem domain. At this point, it is advisable to study this class. Comments in the code explain each section in detail, but a summary of the steps is given here.

1. Create a Problem Domain object
2. Create a Hyper-Heuristic object
3. Load a problem instance
4. Set a time limit
5. Tell the Hyper-Heuristic which problem it is to solve, by giving it the reference to the Problem Domain object
6. Run the Hyper-Heuristic
7. Print the best solution found within the time limit

In `ExampleRun1`, you will see that we create a Hyper-Heuristic object of the `ExampleHyperHeuristic1` type. The `Example Hyper Heuristic One` algorithm is executed on a specific TSP instance for 60 seconds. Once the termination time is reached, the console will display the best solution value found.

At this point, it is advisable to study `ExampleHyperHeuristic1` next, to learn how to develop a hyper-heuristic strategy[1]. Pseudo-code is provided in the Appendix.

Part 2: Preparing Hyper-heuristics Runner for irace

To enable irace to execute the hyper-heuristics within HyFlex and configure parameters effectively, we need to ensure that the Runner is executable from the command line and capable of receiving parameters as arguments. Specifically, the Runner should be designed to accept parameters passed as command-line arguments, and we need to export the Runner as Runnable JAR.

Navigate to the 'examples.rn' package to find `RN`, which extends the `ExampleHyperHeuristic1` by allowing for the passage of 'i' and 'd' parameters.

Note: Several heuristics in HyFlex feature configurable parameters 'i' and 'd'. For simplicity, we only consider configuring these parameters for heuristics with such settings in the provided example. Specifically, for the TSP domain within HyFlex, three heuristics have configurable 'i' and three LLHs have configurable 'd' parameters.

The `RNRunner` executes `RN` and allows arguments (such as parameters, random seeds, instance ID and run time) to be passed from the command line. Running `RNRunner` without required arguments will result in an error message prompting the necessary arguments.

```
Error: -p and -t are required.
```

Refer to the class comments for detailed information on the required arguments and try the example run configuration. The Runner now returns the objective function value of the best-found solution.

To utilise `RNRunner` with irace, we have already exported it as a runnable JAR. The corresponding JAR file is located in the designated folder.

[1] Examples of more complex setup is available at HyFlex website.

Part 3: Preparing Configuration Scenario in irace

The irace User Guide provides a guide for setting up a basic execution of irace in Section 4.1. The template files provided within the irace package serve as the starting point for creating new configuration scenarios.

In this challenge, example configuration files are provided to configure the heuristics of the hyper-heuristics `RN`. These configuration files, along with `RNRunner`, are located in the "project-irace/irace-hyflex-rn" folder. The following sections provide explanations of how these files differ from the template files, showing the minimum that must be done.

1. Target runner

The 'target-runner' is the script that is invoked by irace. It refers to the program that irace runs multiple times with different sets of parameters to assess their effectiveness.

In the provided example, the 'target-runner.exe' facilitates the execution of `RNRunner` with specified parameter configurations and retrieves the resulting objective function value for irace analysis.

Comparison with the template target runner reveals the primary change in the `EXE` and `EXE_PARAMS` variables, specifying the command line for running `RNRunner`. Adjust these variables if you modify the command line for executing the Runner.

Currently, the `-t` argument is set to 10,000 (i.e., 10 seconds), indicating that each run of `RNHH` terminates after 10 seconds. Adapt this value as needed for your configuration.

2. Target algorithm parameters

The "parameters.txt" file describes the parameters, one per line, specifying the name, type, and range.

In this example, there are 6 parameters ($d1, d2, d3, i1, i2, i3$) to configure. Although both 'i' and 'd' take values from [0, 1], 'd' is discretised for efficiency. Therefore, 'd' is treated as a categorical parameter while 'i' is a real-valued parameter.

3. Initial configurations

Initial parameter values are set in "configurations.txt".

Currently, the values are all set to '0.2', consistent with the default setting in HyFlex.

4. Training instances

The training instances are specified in the "instance-train.txt" file, where each line contains the ID of a TSP instance. These IDs are passed to the target Runner as command-line arguments by irace.

Currently, we utilise the TSP instances 0, 1, 2, 3, 4, 5 as training instances. For a comprehensive list of all instances provided in HyFlex, please refer to the Appendix.

5. Configuration scenario

The 'scenario.txt' file contains essential settings for the parameter configuration process. Some key parameters you may need to adjust include directory paths, instance files, and configuration files.

An important setting is the tuning budget, which is specified by either `maxExperiments` or `maxTime`. In this example, `maxExperiments` is specified to limit the number of target Runner executions by irace. Specifically, it is set to 180, meaning that irace will terminate after completing 180 runs.

Part 4: Executing irace

The configuration scenario for the provided RNHH is now ready to use by irace. To execute the configuration scenario in the RStudio console, follow these steps:

1. Change the working directory to the 'irace-hyflex-rn' folder. If the folder's location is '/path/to/project-irace/irace-hyflex-rn', use the command:

```
setwd("/path/to/project-irace/irace-hyflex-rn/")
```

2. Load irace:

```
library("irace")
```

3. Run irace:

```
irace.cmdline("")
```

You'll see text output similar to Figure .

```
> setwd("Desktop/NATCOR-students/project-irace/irace-hyflex-example")
> library("irace")
> irace.cmdline()
#------------------------------------------------------------------------------
# irace: An implementation in R of (Elitist) Iterated Racing
# Version: 3.5.6863679
# Copyright (C) 2010-2020
# Manuel Lopez-Ibanez     <manuel.lopez-ibanez@manchester.ac.uk>
# Jeremie Dubois-Lacoste
# Leslie Perez Caceres    <leslie.perez.caceres@ulb.ac.be>
#
# This is free software, and you are welcome to redistribute it under certain
# conditions.  See the GNU General Public License for details. There is NO
# WARRANTY; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
#
# irace builds upon previous code from the race package:
#     race: Racing methods for the selection of the best
#     Copyright (C) 2003 Mauro Birattari
#------------------------------------------------------------------------------
# installed at: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/irace
# called with:
== irace == WARNING: A default scenario file '/Users/weiyao/Desktop/NATCOR-students/project-irace/irace-hyflex-example/
scenario.txt' has been found and will be read.
# Read 1 configuration(s) from file '/Users/weiyao/Desktop/NATCOR-students/project-irace/irace-hyflex-example/
configurations.txt'
# 2024-03-28 12:07:45 GMT: Initialization
```

Upon completion, you will see an output like Figure .

```
# 2024-03-28 12:38:15 GMT: Stopped because budget is exhausted
# Iteration: 5
# nbIterations: 4
# experimentsUsedSoFar: 180
# timeUsed: 0
# remainingBudget: 0
# currentBudget: 48
# number of elites: 4
# nbConfigurations: 8
# Total CPU user time: 1976.417, CPU sys time: 16.538, Wall-clock time: 1830.083
# Best configurations (first number is the configuration ID; listed from best to worst according to the sum of ranks):
    d1  d2  d3     i1     i2     i3
20 0.8 0.8 0.4 0.0482 0.7421 0.8498
17 0.8 0.8 0.4 0.1861 0.4202 0.6090
16 0.8 0.8   0 0.5194 0.6204 0.5239
5  0.8 0.8 0.4 0.3337 0.9966 0.6985
# Best configurations as commandlines (first number is the configuration ID; same order as above):
20  -d 0.8 0.8 0.4 -i 0.0482 0.7421 0.8498
17  -d 0.8 0.8 0.4 -i 0.1861 0.4202 0.609
16  -d 0.8 0.8 0 -i 0.5194 0.6204 0.5239
5   -d 0.8 0.8 0.4 -i 0.3337 0.9966 0.6985


# Testing of elite configurations: 1
# Testing iteration configurations: FALSE
# 2024-03-28 12:38:15 GMT: No test instances, skip testing
>
```

Congratulations! You've successfully configured parameters for RNHH. You can then try deploying these parameters in RNHH.

Next Steps

To complete this challenge, please utilise the flexibility and capabilities in HyFlex and irace to enhance the provided hyper-heuristics for obtaining better TSP solutions. Below are some guidance to assist your further exploration of these two packages.

More Example Hyper-heuristics

Alongside the basic random hyper-heuristics, we offer two additional hyper-heuristics and their corresponding Runner in the "project-hyflex" folder.

- Within the `examples.mcf` package, you'll find the source code[2] for a modified choice function hyper-heuristic [DÖB12], including `MCF` and its corresponding runner `MCFRunner`.
- For the purpose of this challenge, we present a simplified choice function hyper-heuristic in the `examples.scf` package. This includes the `SCF` and its corresponding runner `SCFRunner`.

Specifically, `SCFRunner` is designed to execute `SCF` and facilitate easy integration with irace, simplifying the preparation of the Runner for irace.

If you choose to automatically configure `MCF`, ensure a comprehensive understanding of the algorithm's implementation.

For those interested in enhancing the heuristic selection method and acceptance criteria, a thorough understanding of the algorithm's implementation is essential. You can draw inspiration from the implementation of `RN` or explore examples available on the HyFlex website.

Explore further in irace

The provided example scenario covers essential elements, but there are additional optional settings within the configuration scenario that have not been addressed. Feel free to delve deeper into the irace user guide, as it offers a wealth of features and can be applied to various configuration scenarios. Some of these include utilising testing instances, implementing parallelisation, and conducting multi-objective tuning. Additionally, the guide provides valuable insights into setting the tuning budget and more information on interpreting output and results.

[2] The original source code is available at here.

Appendix

Pseudocode for the ExampleHyperheuristic1

Algorithm 1 Pseudocode for the ExampleHyperheuristic1

```
1: s ← generateInitialSolution();
2: while termination_criterion_not_satisfied do
3:    h ← randomHeuristicSelection(heuristic_set);
4:    s′ ← applyHeuristic(h, s);
5:    accept ← naiveAcceptance(s, s′);
6:    if accept then
7:       s ← s′;
8:    else
9:       s ← s;
10:   end if
11:   if f(s′) < f(s_best) then
12:      s_best ← s′;
13:   end if
14: end while
15: return s_best;
```

Simplified choice function hyper-heuristics

¡TOADD¿Pseudocode and choice function

TSP Instances within HyFlex

¡TOADD¿

References

[BCH+09] Edmund K Burke, Tim Curtois, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Sanja Petrovic, José Antonio Vázquez-Rodríguez, et al. Hyflex: A flexible framework for the design and analysis of hyper-heuristics. In *Multidisciplinary International Scheduling Conference (MISTA 2009), Dublin, Ireland*, pages 790–797, 2009.

[DÖB12] John H Drake, Ender Özcan, and Edmund K Burke. An improved choice function heuristic selection for cross domain heuristic search. In *Parallel Problem Solving from Nature-PPSN XII: 12th International Conference, Taormina, Italy, September 1-5, 2012, Proceedings, Part II 12*, pages 307–316. Springer, 2012.

[LIDLC+16] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.