ICCAD Contest Platform

The official contest platform for Problem C: Microarchitecture Design Space Exploration of ICCAD Contest.

The problem formulation is illustrated in Problem C.

The platform is inspired from Bayesmark.

Instructions for using the platform

Installation

We recommend you use Python 3.5+ for your environment.

Install the latest version of the platform to your machines.

```
$ pip3 install iccad_contest
```

or

```
$ pip3 install iccad-contest==0.0.4
```

Download the example cases, i.e., **Problem C_cases_0622.zip**, from the official ICCAD Contest problem website.

Get started with a simple case of the random search

Unzip example cases downloaded following Installation, and you can receive files containing example cases and configurations.

```
Problem\ C\ data
├── gp-configs.json
├── gp-optimizer.py
├── lr-offline-optimizer.py
├── lr-online-optimizer.py
├── random-search-optimizer.py
└── random-search-w-early-stopping-optimizer.py
```

```
$ cd Problem\ C\ data
```

Receive the help menu

```
$ python3 random-search-optimizer.py -h
```

```
usage: random-search-optimizer.py [-h] [-o OUTPUT_PATH] [-u UUID]
                                  [-s SOLUTION_SETTINGS] [-q
NUM_OF_QUERIES]

ICCAD'22 Contest Platform - solutions evaluation

optional arguments:
  -h, --help            show this help message and exit
  -o OUTPUT_PATH, --output-path OUTPUT_PATH
                        contest output path specification
  -u UUID, --uuid UUID  universally unique identifier (UUID) specification
  -s SOLUTION_SETTINGS, --solution-settings SOLUTION_SETTINGS
                        solution submission specification
  -q NUM_OF_QUERIES, --num-of-queries NUM_OF_QUERIES
                        the number of queries specification
```

`--output-path`: specifies a directory path to store all output information.

`--uuid`: specifies an universally unique identifier (UUID) as a random seed.

`--solution-settings`: if your implementation contains some `<dict>`-like mapping settings, you need to organize these settings as a JSON file. You specify the file path with `--solution-settings` (please see Get started with a simple case of building a Gaussian process regression model).

`--num-of-queries`: specify how many times to issue an access to the dataset.

Notice that you should specify `--num-of-queries` for your submission. `--num-of-queries` is related to tuning your model with good convergence. More `--num-of-queries` can incur more overall running time (ORT).

The only parameter related to your quality of solution results is `--num-of-queries`. A good algorithm can find better Pareto frontiers with less `--num-of-queries`, i.e., your solution uses less time cost.

A simple case to start the random search

```
$ python3 random-search-optimizer.py -o output -u
"00ef538e88634ddd9810d034b748c24d" -q 10
```

The command specifies `output` as the directory path to store all generated information. It uses `00ef538e88634ddd9810d034b748c24d` as a random seed. It issues 10 times to access the dataset. The results are generated to `output` directory. You can check the summary reports for the random search in `output/summary`, and log information in `output/log`.

We compare the average distance to reference set (ADRS) and the overall running time (ORT) for your submissions with a set of pre-determined UUIDs.

A better solution pertains **lower ADRS and ORT**. Please refer it to Evaluation of your submission.

Get started with a simple case of building a simple linear regression model

We can build a simple model to do the exploration.

The online algorithm can dynamically update the simple model through exploration.

We can experimence an online algorithm with a simple linear regression model.

```
$ python3 lr-online-optimizer.py -o output -u
"00ef538e88634ddd9810d034b748c24d" -q 10
```

Since the codes `lr-online-optimizer.py` is only for demonstration, so no advanced optimization techniques are utilized.

The offline algorithm cannot update the simple model through exploration, i.e., it is frozen. However, it is often trained on a larger initial dataset, and we can evaluate the model's accuracy using collected data. So the initialization algorithm could be critical for such algorithm design.

We can also experimence an offline algorithm with a simple linear regression model.

```
$ python3 lr-offline-optimizer.py -o output -u
"00ef538e88634ddd9810d034b748c24d" -q 2
```

In the offline algorithm, `-q` or `--num-of-queries` is set to 2 since it often sweeps the design space and conducts only one exploration. More `-q` or `--num-of-queries` does no good help improve the ADRS except for degrading ORT.

Get started with a simple case of building a Gaussian process regression model

The example can be executed with

```
$ python3 gp-optimizer.py -o output -s gp-configs.json -u
"00ef538e88634ddd9810d034b748c24d" -q 10
```

In the command, `-s` or `--solution-settings` should be specified. The application program interface (API) of `GaussianProcessRegressorOptimizer` is defined shown below,

```
class GaussianProcessRegressorOptimizer(AbstractOptimizer):
    primary_import = "iccad_contest"

    def __init__(self, design_space, optimizer, random_state):
        ...
```

`optimizer`, `random_state`, etc. are corporated into the initialization function, and they can be specified through a JSON file, i.e., `gp-configs.json`.

```
{
    "optimizer": "fmin_l_bfgs_b",
    "random_state": 2022
}
```

Such design will provide you with a convenient coding and algorithm tuning experience, i.e., if your solution contains many hyper-parameters, you can specify a set of those hyper-parameters with a single JSON file. Be careful that you need to specify which JSON file to use via the experiment command.

Get started with a simple case of the random search when applying the early stopping criterion.

```
$ python3 random-search-w-early-stopping-optimizer.py -o output -u
"00ef538e88634ddd9810d034b748c24d" -q 100
```

You can set `self.early_stopping = True` in *random-search-w-early-stopping-optimizer.py* to early stop your optimizer. Hence, the ORT is determined by your early stopping criterion if your early stopping condition is satisfied instead of `--num-of-queries`.

Prepare your submission

Refer to the random search example, i.e., `random-search-optimizer.py`.

```
"""
    random search optimizer performs random search.
    the code is only for example demonstration.
    ```
        python random-search-optimizer.py \
            -o [your experiment outputs directory] \
            -q [the number of your queries]
    ```
    you can specify more options to test your optimizer. please use
    ```
        python random-search-optimizer.py -h
    ```
    to check.
"""


import numpy as np
from iccad_contest.abstract_optimizer import AbstractOptimizer
from iccad_contest.design_space_exploration import experiment
```

```
class RandomSearchOptimizer(AbstractOptimizer):
    primary_import = "iccad_contest"

    def __init__(self, design_space):
        """
            build a wrapper class for an optimizer.

            parameters
            ----------
            design_space: <class "MicroarchitectureDesignSpace">
        """
        AbstractOptimizer.__init__(self, design_space)
        self.n_suggestions = 1

    def suggest(self):
        """
            get a suggestion from the optimizer.

            returns
            -------
            next_guess: <list> of <list>
                list of `self.n_suggestions` suggestion(s).
                each suggestion is a microarchitecture embedding.
        """
        x_guess = np.random.choice(
            range(1, self.design_space.size + 1),
            size=self.n_suggestions
        )
        return [
            self.design_space.vec_to_microarchitecture_embedding(
                self.design_space.idx_to_vec(_x_guess)
            ) for _x_guess in x_guess
        ]

    def observe(self, x, y):
        """
            send an observation of a suggestion back to the optimizer.

            parameters
            ----------
            x: <list> of <list>
                the output of `suggest`.
            y: <list> of <list>
                corresponding values where each `x` is mapped to.
        """
        pass


if __name__ == "__main__":
    experiment(RandomSearchOptimizer)
```

You need to implement `suggest`, and `observe` functions after you inherit `AbstractOptimizer` from `iccad_contest.abstract_optimizer`. `suggest`, as the function name shows intuitively, is to provide a suggestion. `observe` takes action after your optimizer sees the suggestion and corresponding objective values. You can train a surrogate model, adjust your optimization strategy, etc. `x` and `y` are `<list>` in `observe`. `x` corresponds to your previous suggestions, and `y` are the objective values corresponding to `x`. **The operations of how each element of `x` is mapped to each element `y` are hidden by the platform, so you do not need to care about the details.** Nevertheless, you need to ensure that each element of `x` is valid, i.e., each element of `x` is in the design space. An 'out of the design space' error prompts if an element of `x` is not in the design space, and the platform will fall back to generate random suggestions in that case. The design space exploration engine will frequently call your `suggest`, and `observe` to find better microarchitectures. If you have organized some hyper-parameters to a JSON file and passed the JSON file to the platform, you can check Get started with a simple case of building a Gaussian process regression model for more information about how to use a JSON file to organize your codes. The submission includes

- your implemented Python scripts,
- the execution commands,
- related JSON configurations leveraged by your algorithm, if any, and
- dependency packages list for supporting your algorithm execution, if any.

Microarchitecture Design Space

Definition of the design space

The design space is extracted from RISC-V BOOM. We will release dataset step by step. The current dataset only includes one design set called `sub-design-1`, and it is defined using a sheet with `xlsx` format. The sheet is located at `iccad_contest/contest-dataset`. The sheet comprises two sub sheets, i.e., `Microarchitecture Design Space` and `Components`. In `Microarchitecture Design Space`, each component is chosen by an index, which is mapped to a kind of structure defined in `Components`. The number at the end column of `Microarchitecture Design Space` is the size of the design space. `Components` holds different definitions of each component. Within each component, `idx` denotes the index of a kind of structure with respect to the component. A member `design_space` defined in `AbstractOptimizer` specifies the microarchitecture design space. `design_space.descriptions` define the descriptions of parameterized components for the microarchitecture, and `design_space.components_mappings` define each component's structure parameters mappings. For more information about the data structure, please refer to `MicroarchitectureDesignSpace` defined in `iccad_contest/functions/design_space.py`.

Some data structures are helpful that you might frequently leverage.

```
descriptions = {
    "sub-design-1": {
        "Fetch": [1],
        "Decoder": [1],
        "ISU": [1, 2, 3],
        "IFU": [1, 2, 3],
        ...
    }
}
```

```
components_mappings = {
    "Fetch": {
        "description": ["FetchWidth"],
        "1": [4]
    },
    "Decoder": {
        "description": ["DecodeWidth"],
        "1": [1]
    },
    "ISU": {
        "description": [
            "MEM_INST.DispatchWidth", "MEM_INST.IssueWidth"
            "MEM_INST.NumEntries", "INT_INST.DispatchWidth",
            "INT_INST.IssueWidth", "INT_INST.NumEntries",
            "FP_INST.DispatchWidth", "FP_INST.IssueWidth",
            "FP_INST.NumEntries"
        ],
        "1": [1, 1, 8, 1, 1, 8, 1, 1, 8],
        "2": [1, 1, 6, 1, 1, 6, 1, 1, 6],
        "3": [1, 1, 10, 1, 1, 12, 1, 1, 12]
    },
    "IFU": {
        "description": ["BranchTag", "FetchBufferEntries",
"FetchTargetQueue"]
        "1": [8, 8, 16],
        "2": [6, 6, 14],
        "3": [10, 12, 20]
    },
    ...
}
```

`descriptions` and `components_mappings` help to define the design space. We index each design with one integer. The first design is indexed with 1. The second is indexed with 2. The same logic goes for others.

Some functions are helpful that you might frequently leverage.

```
def idx_to_vec(self, idx):
    ...
```

`idx_to_vec` transforms an index to an index vector, then used to specify a combination defined in `Microarchitecture Design Space`.

```
def vec_to_microarchitecture_embedding(self, vec):
    ...
```

`vec_to_microarchitecture_embedding` transforms an index vector to a microarchitecture embedding, i.e., a feature vector with each element defines a structure of a component for the processor.

If you want to operate with the design space using functions that are not supported by the contest platform, you need to implement them yourself.

Dataset

The released dataset is a two-dimensional matrix with the shape of (number of designs for `sub-design-1`, *CONCAT*(`microarchitecture embedding`, `performance`, `power`, `area`, `time of the VLSI flow`)).

The dataset is a **dummy**, helping you familiarize yourself with the platform.

The dummy dataset is in `iccad_contest/contest-dataset/contest.csv`

`microarchitecture embedding`: a vector concatenated from structure representations of all components.

`performance`: a scalar value denotes the performance of the design.

`power`: a scalar value denotes the power of the design.

`area`: a scalar value denotes the area of the design.

`time of the VLSI flow`: a scalar value denotes the time cost spent to evaluate accurate performance, power, and area values via the VLSI flow.

We will evaluate your submission with real dataset, measured with commonly-used benchmarks and commercial electronic design automation (EDA) tools. Notice that we will organize, and announce another platform, allowing you to submit your solutions with a fixed frequency, and you can see your results, i.e., ADRS and ORT, on a real dataset. You can modify and tune your solutions with respect to the released results of your previous submission.

Evaluation of your submission

We compare the average distance to reference set (ADRS) and the overall running time (ORT) for your submissions with a set of pre-determined UUIDs.

For a fair comparison of ADRS, the platform automatically normalized the performance, power, and area values, and the ADRS is calculated based on such manipulated values. Hence, it is normal to see some negative normalized PPA values. As Dataset introduces, we will evaluate your solution with the actual dataset.

The final score is calculated based on your optimizer's average ADRS and ORT on a pre-determined series of hidden UUIDs. The calculation is via

$$\text{score} = \frac{1}{\text{ADRS}} \cdot \begin{cases} \alpha - \dfrac{\text{ORT} - \theta}{\theta}, \text{ORT} \geq \theta \\ \alpha + \left|\dfrac{\text{ORT} - \theta}{\theta}\right|, \text{ORT} < \theta \end{cases},$$

where α is an ORT score baseline, equal to 6, and θ is a pre-defined ORT budget, equivalent to 2625000. The constant is set to align with the de facto microarchitecture design space exploration flow. It is worth noting that if your ORT is six times larger than θ, then your final score will be negative. Hence, a better solution has lower ADRS and ORT as much as possible.

Notice

The ICCAD Contest includes alpha, beta, and final test submissions. We will release a dataset each round, and you need to use the released dataset to test your implementations. At the final stage, we have hidden cases to test your implementations, and we compare the results of all contestants based on hidden cases.

- ~~Alpha test submission 2022/06/13 17:00:00 (GTM+8)~~
- Beta test submission 2022/07/22 17:00:00 (GTM+8)
- Final submission 2022/08/30 17:00:00 (GTM+8)

Contact

Please contact **cad.contest.iccad@gmail.com** or DAMO Academy for any question, comment, or bug report.