Project 2 Report

School of Life Science ZhouBaichuan 2300012301

1. Introduction

In this project, we will explore the design of a branch predictor (BP) in a computer system. Specifically, we will introduce how to add a branch predictor module in the configuration script. We also provide a tutorial to add a new branch predictor in gem5 . You will evaluate how different types of branch predictors can impact the system performance. Further, you will try to implement various BPs by yourselves. Specifically, you will first implement a simple BP following the tutorial. Then, you are required to read a paper that first describes the perceptron BP. You need to summarize the paper and implement the perceptron BP. Next, you are required to compare your BP's performance with other BPs that gem5 has provided.

2. Experiment Setup

2.1 Environment

- **gem5 Version:** download from https://github.com/ChaseLab-PKU/archlab-gem5
- **Host System:** Ubuntu 22.04 from WSL2
- **Hardware configuration:** CPU: i7-13700HX, GPU: 4060 Laptop, DRAM: 16GB DDR4
- **Simulator Mode:** syscall emulation (SE) mode

2.2 System Configuration

- **CPU Model:** Simple CPU / Minor CPU / O3CPU
- **Memory:** 2GB DDR3 / 2GB DDR4 / 2GB DDR5
- **Other Parameters:** clock frequency: 2GHz

2.3 Benchmarks/Programs

- **Tested Programs:**
  - `shell_sort`
  - `spfa`
  - `branch_optimized`

2.4 Simulation Commands

- use `testlab2.sh` to automately run the simulations

3. Results

3.1 Codes

1. BranchPredictor.py

```
class Lab2PerceptronBP(BranchPredictor):
    type = "Lab2PerceptronBP"
    cxx_class = "gem5::branch_prediction::Lab2PerceptronBP"
    cxx_header = "cpu/pred/lab2_bp.hh"

    numPerceptrons = Param.Unsigned(1024, "Number of perceptrons (table entries)")
    historyLength = Param.Unsigned(32, "Number of global history bits")
    weightBits = Param.Unsigned(8, "Bits per weight (including sign)")
    threshold = Param.Int(60, "Training threshold (θ)")
    initWeight = Param.Int(0, "Initial value for all weights")
```

2. lab2_bp.hh

```
#ifndef __CPU_PRED_LAB2_BP_HH__
#define __CPU_PRED_LAB2_BP_HH__

#include "base/types.hh"
#include "cpu/pred/bpred_unit.hh"
#include "params/Lab2PerceptronBP.hh"

namespace gem5 {
namespace branch_prediction {

struct Perceptron {
    std::vector<int> weights; // include w0..wh
};



class Lab2PerceptronBP : public BPredUnit {

private:
  unsigned historyLength;
  unsigned weightBits;
  unsigned numPerceptrons;
  int threshold;
  int initWeight;
  int weightMax, weightMin;

  std::vector<Perceptron> perceptronTable;
  std::vector<bool> globalHistory; // size = historyLength

  inline int saturatingAdd(int weight, int value);
  inline int predictPerceptron(const Perceptron &p);
  inline unsigned getPerceptronIndex(Addr branch_addr);

  struct BPhistory {
    unsigned perceptronIndex;
    std::vector<bool> globalHistory; // size = historyLength
    int outputY;
    bool prediction;
  };
```

```
  public:
    Lab2PerceptronBP(const Lab2PerceptronBPParams &params);

    bool lookup(ThreadID tid, Addr branch_addr, void *&bp_history) override;
    void updateHistories(ThreadID tid, Addr pc, bool uncond, bool taken,
                         Addr target, void *&bp_history) override;
    void update(ThreadID tid, Addr pc, bool taken, void *&bp_history,
                bool squashed, const StaticInstPtr &inst, Addr target) override;
    void squash(ThreadID tid, void *&bp_history) override;



  };
  } // namespace branch_prediction
  } // namespace gem5

  #endif // __CPU_PRED_LAB2_BP_HH__
```

3. lab2_bp.cc

```
#include "cpu/pred/lab2_bp.hh"

namespace gem5 {
namespace branch_prediction {
Lab2PerceptronBP::Lab2PerceptronBP(const Lab2PerceptronBPParams &params)
    : BPredUnit(params),
    numPerceptrons(params.numPerceptrons),
    historyLength(params.historyLength),
    weightBits(params.weightBits),
    threshold(params.threshold),
    initWeight(params.initWeight) {
      weightMax = (1 << (weightBits - 1)) - 1;
      weightMin = -(1 << (weightBits - 1));

      globalHistory.assign(historyLength, false);

      perceptronTable.resize(numPerceptrons);
      for (unsigned i = 0; i < numPerceptrons; ++i) {
          perceptronTable[i].weights.resize(historyLength + 1, initWeight);
      }
    }

inline
int
Lab2PerceptronBP::saturatingAdd(int weight, int value) {
    int result = weight + value;
    if (result > weightMax) {
        return weightMax;
    } else if (result < weightMin) {
        return weightMin;
    } else {
        return result;
    }
}

inline
```

```
int
Lab2PerceptronBP::predictPerceptron(const Perceptron &p) {
    int y = p.weights[0]; // bias weight
    for (unsigned i = 0; i < historyLength; ++i) {
        if (globalHistory[i]) {
            y += p.weights[i + 1];
        } else {
            y -= p.weights[i + 1];
        }
    }
    return y;
}

inline
unsigned
Lab2PerceptronBP::getPerceptronIndex(Addr branch_addr) {
    return (branch_addr >> 2) % numPerceptrons;
}

bool Lab2PerceptronBP::lookup(ThreadID tid, Addr branch_addr,
                              void *&bp_history) {
  unsigned p_index = getPerceptronIndex(branch_addr);
  Perceptron &p = perceptronTable[p_index];
  int y = predictPerceptron(p);
  bool prediction = (y >= 0);

  BPhistory *history = new BPhistory;
  history->perceptronIndex = p_index;
  history->globalHistory = globalHistory;
  history->outputY = y;
  history->prediction = prediction;
  bp_history = (void *)history;

  return prediction;

}
void Lab2PerceptronBP::updateHistories(ThreadID tid, Addr pc, bool uncond,
                                       bool taken, Addr target,
                                       void *&bp_history) {
  assert(uncond || bp_history);

  if(uncond) {
      BPhistory *history = new BPhistory;
      history->perceptronIndex = getPerceptronIndex(pc);
      history->globalHistory = globalHistory;
      history->outputY = threshold + 1; // unused
      history->prediction = true; // unused
      bp_history = (void *)history;
  }

  // Update global history
  globalHistory.insert(globalHistory.begin(), taken);
  if (globalHistory.size() > historyLength) {
      globalHistory.pop_back();
  }

}
void Lab2PerceptronBP::update(ThreadID tid, Addr pc, bool taken,
                              void *&bp_history, bool squashed,
                              const StaticInstPtr &inst, Addr target) {
```

```
    assert(bp_history);
    BPhistory *history = static_cast<BPhistory *>(bp_history);
    Perceptron &p = perceptronTable[history->perceptronIndex];

    // Train the perceptron if the prediction was incorrect or
    // the output y is within the threshold
    int t = taken ? 1 : -1;

    if (squashed) {
      globalHistory = history->globalHistory;
      // delete history;
      // bp_history = nullptr;
      return;
    }

    if (history->prediction != taken || abs(history->outputY) <= threshold) {
      // Update bias weight
      p.weights[0] = saturatingAdd(p.weights[0], t);

      // Update other weights
      for (unsigned i = 0; i < historyLength; ++i) {
        int h = history->globalHistory[i] ? 1 : -1;
        p.weights[i + 1] = saturatingAdd(p.weights[i + 1], t * h);
      }
    }

    delete history;
    bp_history = nullptr;
  }
  void Lab2PerceptronBP::squash(ThreadID tid, void *&bp_history) {

    assert(bp_history);
    BPhistory *history = static_cast<BPhistory *>(bp_history);
    globalHistory = history->globalHistory;
    delete history;
    bp_history = nullptr;

  }
  } // namespace branch_prediction
  } // namespace gem5
```

4. branch_optimizer.c

raw_code

```
  /*   START position of editable area    */

  int min(int x, int y) {
      return y ^ ((x ^ y) & -(x < y));
  }

  int max(int x, int y) {
      return x ^ ((x ^ y) & -(x < y));
  }

  int abs(int x) {
      int const mask = x >> (sizeof(int) * 8 - 1);
```

```
    return (x + mask) ^ mask;
}

void statistics(WeatherData data[], int n) {

    for(int i = 0; i < n; i++) {
        if(data[i].temperature < -70 || data[i].temperature > 60)
            continue;

        if(data[i].windSpeed < 0 || data[i].windSpeed > 150)
            continue;

        if(data[i].humidity < 0 || data[i].humidity > 100)
            continue;

        if(data[i].day < 1 || data[i].day > 31)
            continue;

        if(data[i].month > 12 || data[i].month < 1)
            continue;

        if(data[i].year < 1900 || data[i].year > 2025)
            continue;

        valid_data_count++;

        if(data[i].year >= 2012 && data[i].year < 2024)
            year_count++;
        if(data[i].month > 6 && data[i].month < 8)
            month_count++;
        if(data[i].day == 15)
            day_count++;

        switch(data[i].city) {
            case NEW_YORK:
                city_count[0]++;
                break;
            case LOS_ANGELES:
                if(data[i].condition == RAINY)
                    los_rainy_days++;
                city_count[1]++;
                break;
            case CHICAGO:
                city_count[2]++;
                break;
            case HOUSTON:
                city_count[3]++;
                break;
            case MIAMI:
                city_count[4]++;
                break;
            default:
                break;
        }

        switch(data[i].condition) {
            case SUNNY:
                condition_count[0]++;
                break;
            case CLOUDY:
```

```
                condition_count[1]++;
                break;
            case RAINY:
                condition_count[2]++;
                break;
            case STORMY:
                condition_count[3]++;
                break;
            case SNOWY:
                condition_count[4]++;
                break;
            default:
                break;
        }

        if(data[i].windSpeed > 28.4)
            wind_scale_count[11]++;
        else if(data[i].windSpeed > 24.4)
            wind_scale_count[10]++;
        else if(data[i].windSpeed > 20.7)
            wind_scale_count[9]++;
        else if(data[i].windSpeed > 17.1)
            wind_scale_count[8]++;
        else if(data[i].windSpeed > 13.8)
            wind_scale_count[7]++;
        else if(data[i].windSpeed > 10.7)
            wind_scale_count[6]++;
        else if(data[i].windSpeed > 7.9)
            wind_scale_count[5]++;
        else if(data[i].windSpeed > 5.4)
            wind_scale_count[4]++;
        else if(data[i].windSpeed > 3.3)
            wind_scale_count[3]++;
        else if(data[i].windSpeed > 1.5)
            wind_scale_count[2]++;
        else if(data[i].windSpeed > 0.2)
            wind_scale_count[1]++;
        else
            wind_scale_count[0]++;

        max_temp = max(max_temp, data[i].temperature);
        min_temp = min(min_temp, data[i].temperature);

        max_abs_temp = max(max_abs_temp, abs(data[i].temperature));

        if(data[i].humidity < 50)
            low_humidity_count++;

        if(data[i].temperature > 30)
            high_temperature_count++;
        data[i].is_scanned = 1;
    }
}

/*    END position of editable area     */
```

optimized_code

```
*   START position of editable area    */
```

```
    Start position of suitable area    */

* 优化实现：使用局部累加器、合并验证检查、索引化城市处理、阈值数组判断风速档位 */

oid statistics(WeatherData data[], int n) {
   // 局部计数器（避免频繁访问 volatile 全局）
   int loc_los_rainy = 0;
   int loc_condition_count[5] = {0}; // SUNNY, CLOUDY, RAINY, STORMY, SNOWY
   int loc_city_count[5] = {0}; // NEW_YORK..MIAMI
   int loc_wind_scale_count[12] = {0}; // 0..11
   int loc_max_abs_temp = 0;
   float loc_max_temp = -100.0f;
   float loc_min_temp = 100.0f;

   int loc_low_humidity_count = 0;
   int loc_high_temperature_count = 0;
   int loc_valid_data_count = 0;
   int loc_year_count = 0;
   int loc_month_count = 0;
   int loc_day_count = 0;

   // 风速阈值（升序），用于确定档位（index 0 表示 <=0.2）
   const float wind_thresholds[11] = {
       0.2f, 1.5f, 3.3f, 5.4f, 7.9f, 10.7f, 13.8f, 17.1f, 20.7f, 24.4f, 28.4f
   };

   for (int i = 0; i < n; ++i) {
       WeatherData *d = &data[i];

       // 将多次边界检查合并为一个无短路位运算表达式（保证所有比较都被计算，减少分支数量与短路带来的预测不稳定）
       int temp_out = (d->temperature < -70.0f) | (d->temperature > 60.0f);
       int wind_out = (d->windSpeed < 0.0f) | (d->windSpeed > 150.0f);
       int hum_out = (d->humidity < 0.0f) | (d->humidity > 100.0f);
       int day_out = (d->day < 1) | (d->day > 31);
       int month_out = (d->month < 1) | (d->month > 12);
       int year_out = (d->year < 1900) | (d->year > 2025);

       if (temp_out | wind_out | hum_out | day_out | month_out | year_out)
           continue; // 单一分支决定数据合法性

       // 数据合法，使用局部计数器累加
       loc_valid_data_count++;

       // 简洁的布尔算术，分支更可预测
       loc_year_count += (d->year >= 2012 && d->year < 2024);
       loc_month_count += (d->month > 6 && d->month < 8);
       loc_day_count += (d->day == 15);

       // 城市处理：用索引方式减少 switch 的分支抖动
       int cidx = ((int)d->city) - 1; // NEW_YORK=1 -> 0
       if ((unsigned)cidx < 5u) {
           loc_city_count[cidx]++;
           // Los Angeles rainy days: cidx==1 corresponds to LOS_ANGELES
           loc_los_rainy += (cidx == 1 && d->condition == RAINY);
       }

       // condition -> 索引映射（小范围判断，分支局部且可预测）
       int cond_idx = -1;
       if (d->condition == SUNNY) cond_idx = 0;
       else if (d->condition == CLOUDY) cond_idx = 1;
       else if (d->condition == RAINY) cond_idx = 2;
```

```
        else if (d->condition == STORMY) cond_idx = 3;
        else if (d->condition == SNOWY) cond_idx = 4;


        if (cond_idx >= 0)
            loc_condition_count[cond_idx]++;

        // 风速分档：通过阈值数组和小循环计算档位，避免长串 else-if
        int widx = 0;
        // 依次比较，通常会在较低索引处提前跳出（对常见风速友好）
        while (widx < 11 && d->windSpeed > wind_thresholds[widx]) ++widx;
        loc_wind_scale_count[widx]++;

        // 温度最大/最小更新（直接比较）
        float t = d->temperature;
        if (t > loc_max_temp) loc_max_temp = t;
        if (t < loc_min_temp) loc_min_temp = t;

        // 最大绝对温度（以整数温度为基准，与原实现语义一致）
        int ti = (int)t;
        int abs_ti = abs(ti);
        if (abs_ti > loc_max_abs_temp) loc_max_abs_temp = abs_ti;

        // 湿度、极端高温计数
        loc_low_humidity_count += (d->humidity < 50.0f);
        loc_high_temperature_count += (t > 30.0f);

        d->is_scanned = 1;
    }

    // 最后一次性写回 volatile 全局变量，减少内存写压力
    los_rainy_days = loc_los_rainy;
    for (int i = 0; i < 5; ++i) condition_count[i] = loc_condition_count[i];
    for (int i = 0; i < 5; ++i) city_count[i] = loc_city_count[i];
    for (int i = 0; i < 12; ++i) wind_scale_count[i] = loc_wind_scale_count[i];

    max_abs_temp = loc_max_abs_temp;
    max_temp = loc_max_temp;
    min_temp = loc_min_temp;

    low_humidity_count = loc_low_humidity_count;
    high_temperature_count = loc_high_temperature_count;
    valid_data_count = loc_valid_data_count;
    year_count = loc_year_count;
    month_count = loc_month_count;
    day_count = loc_day_count;


*    END position of editable area     */
```

3.2 Paper Review

Summary of *Dynamic Branch Prediction with Perceptrons*

Authors: Daniel A. Jiménez and Calvin Lin
Institution: The University of Texas at Austin

Motivation

Modern superscalar processors rely on *branch prediction* to sustain instruction-level parallelism. Traditional two-level adaptive predictors (e.g., **gshare**, **bi-mode**) use tables of 2-bit saturating counters but face key limitations:

- **Aliasing:** Different branches may share table entries and interfere with one another.
- **Limited history length:** Hardware cost grows exponentially with history length, restricting long-range correlation tracking.

The authors propose using a **perceptron**—a simple neural network—to improve prediction accuracy by enabling *longer branch histories* with only *linear* hardware growth.

Challenges

1. **Hardware efficiency:** Neural methods are typically computation-heavy; implementation must remain feasible for hardware.
2. **Latency:** Prediction (a dot product operation) must complete within one or two CPU cycles.
3. **Training cost:** Online weight updates after every branch must be simple and fast.
4. **Learning limitation:** Perceptrons can only perfectly learn *linearly separable* branch behaviors.
5. **Context switching:** Predictor performance may degrade when processes share hardware resources.

Design Overview

The **Perceptron Branch Predictor** replaces the traditional pattern history table with a *table of perceptrons*.

- Each perceptron corresponds to one static branch and stores a vector of integer weights.
- Inputs are bits from the **global branch history register** (`taken = +1` , `not taken = −1`).
- The perceptron computes a weighted sum:
  $$y = w_0 + \sum_i x_i w_i$$
  If $y \ge 0$: predict **taken**; otherwise, **not taken**.
- After each branch, weights are updated using a simple rule based on the outcome and a tunable **threshold** θ.

Hardware and Performance Features

- **Linear hardware scaling** with history length → enables use of 60+ history bits.
- **Integer arithmetic** allows efficient one-cycle or two-cycle prediction.
- **Performance:** On SPEC 2000 integer benchmarks, the perceptron predictor reduces mispredictions by **~10.1%** over gshare under a 4 KB hardware budget.
- Works best for **linearly separable branches** and those requiring **long history correlations**.

- A **hybrid gshare + perceptron** design offers additional robustness under context switching.

Key Results and Insights

- Perceptrons successfully capture long-range correlations that conventional predictors miss.
- Despite their simplicity, they perform well even with limited integer weights.
- The predictor is complementary to existing approaches:
  - Better on *linearly separable* branches.
  - Slightly worse on *nonlinear* (inseparable) branches.
- Provides an inherent **confidence measure** based on output magnitude, useful for speculative execution.

Conclusion

The perceptron branch predictor is the first successful application of a neural network in dynamic branch prediction.
It:

- Outperforms traditional predictors at comparable hardware budgets.
- Scales efficiently with history length.
- Opens new directions for **hybrid predictors**, **compiler-assisted classification**, and **confidence-guided speculation**.

This work marks a foundational step in integrating *machine learning* into processor microarchitecture.

3.3 Results and statistics

Fig 1. IPC and accuracy between different executables and BPs

	A	B	C	D	E	F	G	H	I	J	K
1		LocalBP		BiModeBP		TournamentBP		SimpleBP		PerceptronBP	
2	executable	shell_sort	spfa	shell_sort	spfa	shell_sort	spfa	shell_sort	spfa	shell_sort	spfa
3	IPC	0.080996	0.091588	0.085154	0.093261	0.08516	0.092961	0.05876	0.075918	0.085533	0.094672
4	mispredicted	2259895	1265585	1635468	1196489	1637366	1195477	6476373	2566043	1600897	1117263
5	commited	11441421	5668471	11441421	5668471	11441421	5668471	11441421	5668471	11441421	5668471
6	accuracy	0.802481	0.776733	0.857057	0.788922	0.856891	0.789101	0.433954	0.547313	0.860079	0.802899

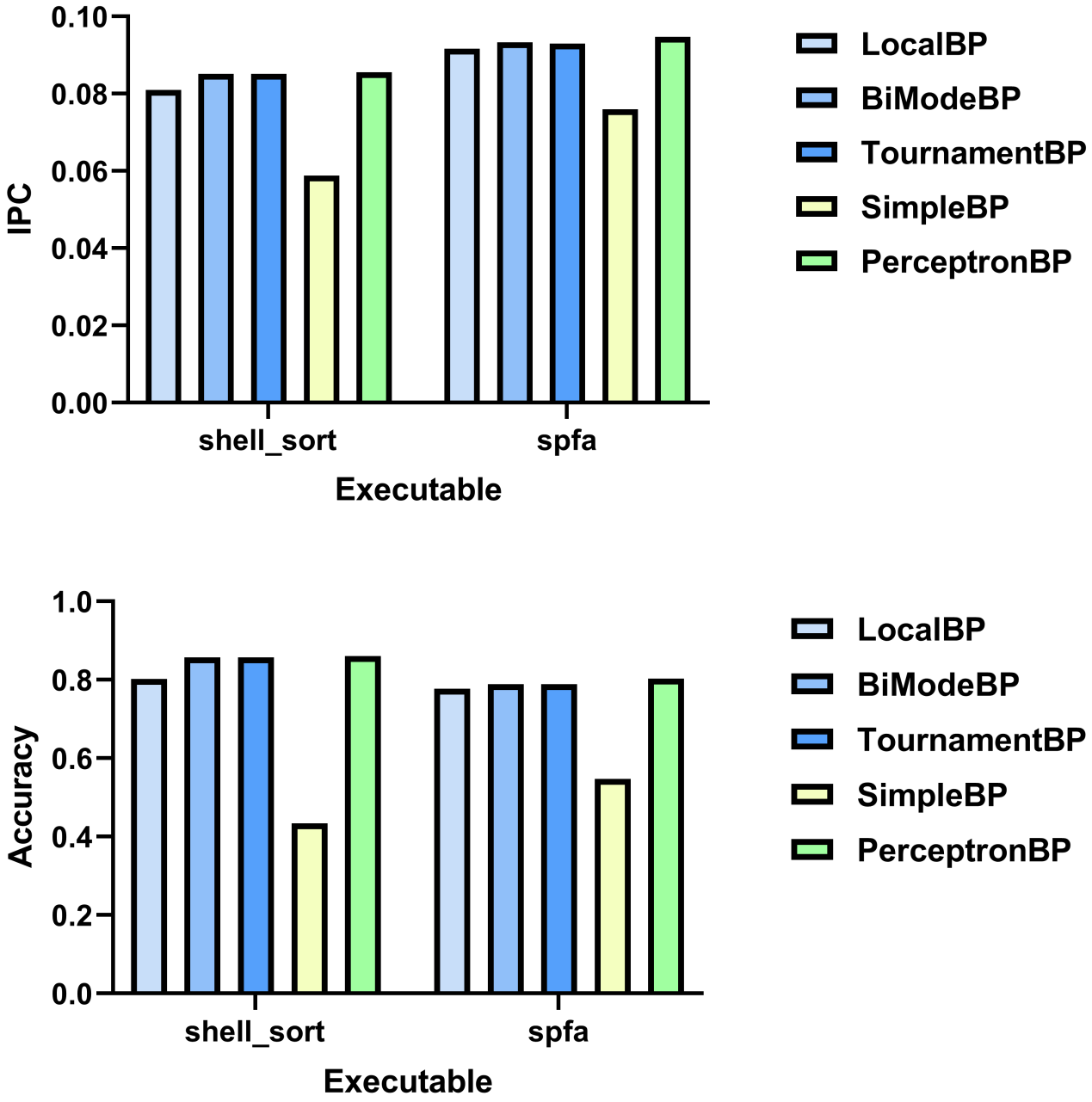
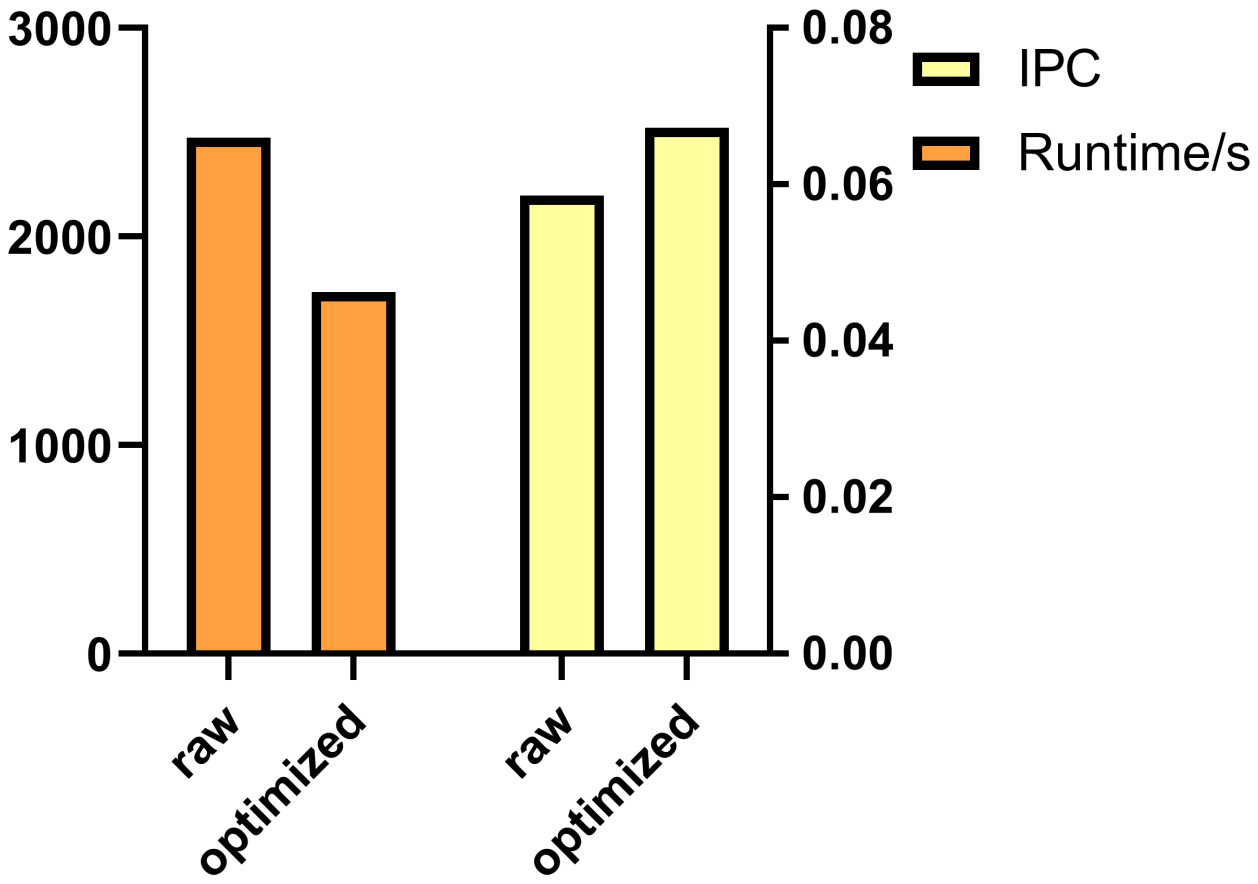


Fig 2. Performance of `branch_optimize` before and after optimize

	raw	optimized
Runtime/s	2473.34	1733.28
IPC	0.058562	0.067233



4. Appendix

- **Codes and Raw Outputs:** Recources can be found at https://github.com/ZhouBC123/Autumn2025_PKU_Archlab.