Adaptive Transfer of Genetic Knowledge in Evolutionary Optimization and Program Synthesis

Yifan He

supervised by Prof. Tetsuya Sakurai & Prof. Claus Aranha

University of Tsukuba

February 2023

Outline

Genetic Knowledge Transfer (GKT)

The process where one Evolutionary Algorithm (EA) is affected by the dynamics of another EA

Adaptive Transfer of Genetic Knowledge

Effective GKT among a sequence of many different tasks based on fitness reward and self-adaptation

Part I: Genetic Knowledge Transfer

Evolutionary Algorithm (EA)

Search algorithms based on the idea of natural evolution

- Solve difficult Optimization Problems (OPs)
- Applications other than OPs: composing music, generating programs, ...

Solving Many Distinct Tasks in A Sequence

Humans solve many distinct tasks and learn to improve themselves

- Learn knowledge from the past tasks
- Use the obtained knowledge to solve future tasks

The performance of an conventional EA is **NOT** influenced by the past tasks it has solved

Can EA learn and improve itself when solving a sequence of many distinct tasks like a human?

Evolutionary Algorithms with Multiple Tasks

Multi-Objective Optimization

Optimize more than one conflicting objectives in one run

- Maximize return and minimize risk in financial investment

Multi-Task Optimization

Optimize more than one related/similar tasks in one run

Genetic Programming with Multiple Tasks

Search computer programs to solve more than one tasks

- Several tasks in one run
- Several tasks in sequence

Multi-Task Optimization (MTO)

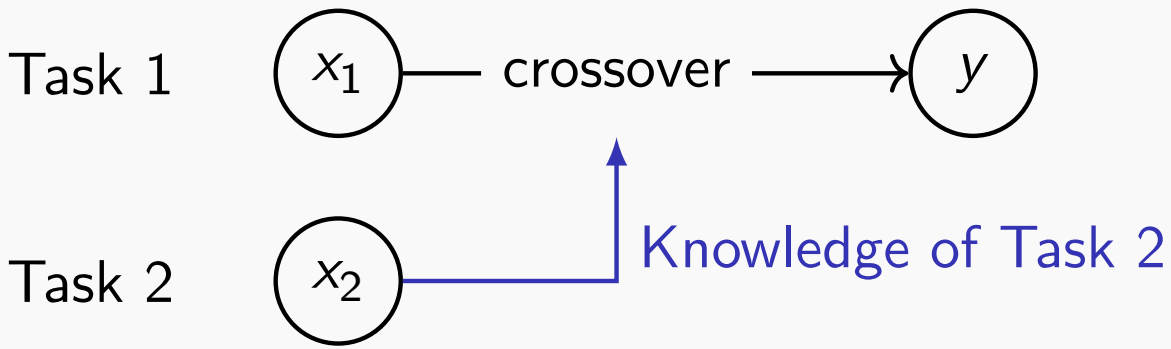
Optimize more than one related/similar tasks in one run by exploiting the common knowledge

- Task 1: optimize the structural design of a sedan car
- Task 2: optimize the structural design of a SUV

Multi-population Multi-Task Evolutionary Framework[1]

Every task is solved by a sub-population

- Solutions to the different tasks go to a crossover step (with a probability)



Genetic Programming (GP) with Multiple Tasks

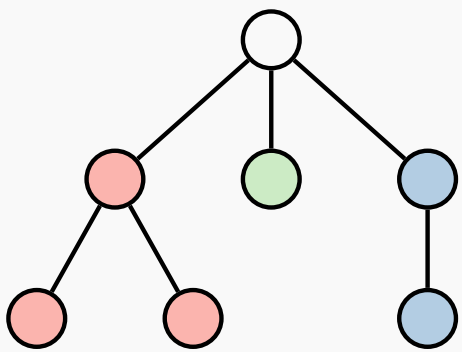
GP: EA that searches a population of trees

Using the subtrees from the solutions to one task to help the search of another task

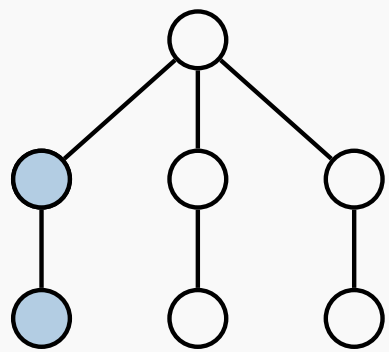
Reusing Extracted Knowledge in Genetic Programming[2]

1. Solve Task 1 and extract subtrees from good solutions
2. Randomly select a subtree and mutate the individual of Task 2

Task 1



Task 2



Definition of Genetic Knowledge Transfer (GKT)

The process where an EA is **affected by the dynamics** of another EA

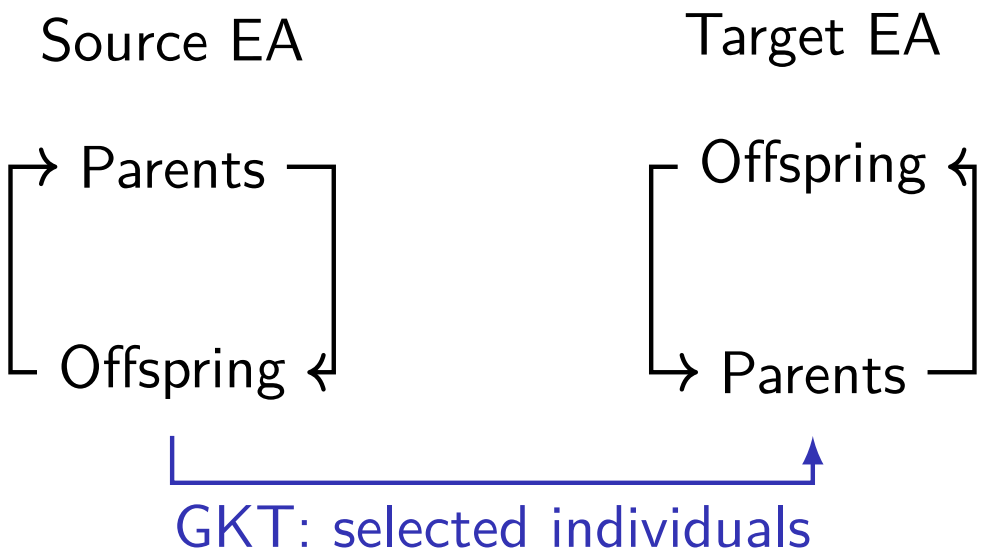
- Genetic knowledge = the dynamics of an EA (all individuals, parameters, . . . , during evolution)
- Transfer = the process to affect an EA

The entire dynamics of an EA is too much to use

- Representative: best individual, final solution/population
- Building blocks: sub-solutions of final solution, common sub-solutions
- High-level statistics: distribution of good variables

Naive Genetic Knowledge Transfer (NGKT)

Select individuals from the offspring of source EA (affecting EA) and use as parents in the target EA (affected EA)



NGKT is the common component of many literature methods

- NGKT in MTO
- NGKT+building block extraction in GP
- NGKT+statistics

Case Study I: Multi-Criteria Seismic History Matching

Overview of Case Study I

Research Question

How to aggregate different objective functions in Seismic History Matching 1)without assigning weights and 2)getting solutions good on all objectives?

Methodology

We apply the idea of Genetic Knowledge Transfer

1. We explain that Lexicase Selection is a variant of NGKT
2. We propose a novel method based on Lexicase Selection and Differential Evolution

Seismic History Matching (SHM)

Search a model x of subsurface by matching simulation results with historical observation regarding M metrics

$$\begin{aligned} \textit{minimize} \quad & ||sim_k(x) - obs_k||_{k=1}^{M} \\ s.t. \quad & x \in \mathbb{R}^n \end{aligned}$$

Example of SHM with two metrics

- Seismic data: a scan of the field using wave
- Well log data: a record of metric value in different time

Aggregation of multiple metrics

- Different types: matrix (seismic) and time series (well log)
- Different scales

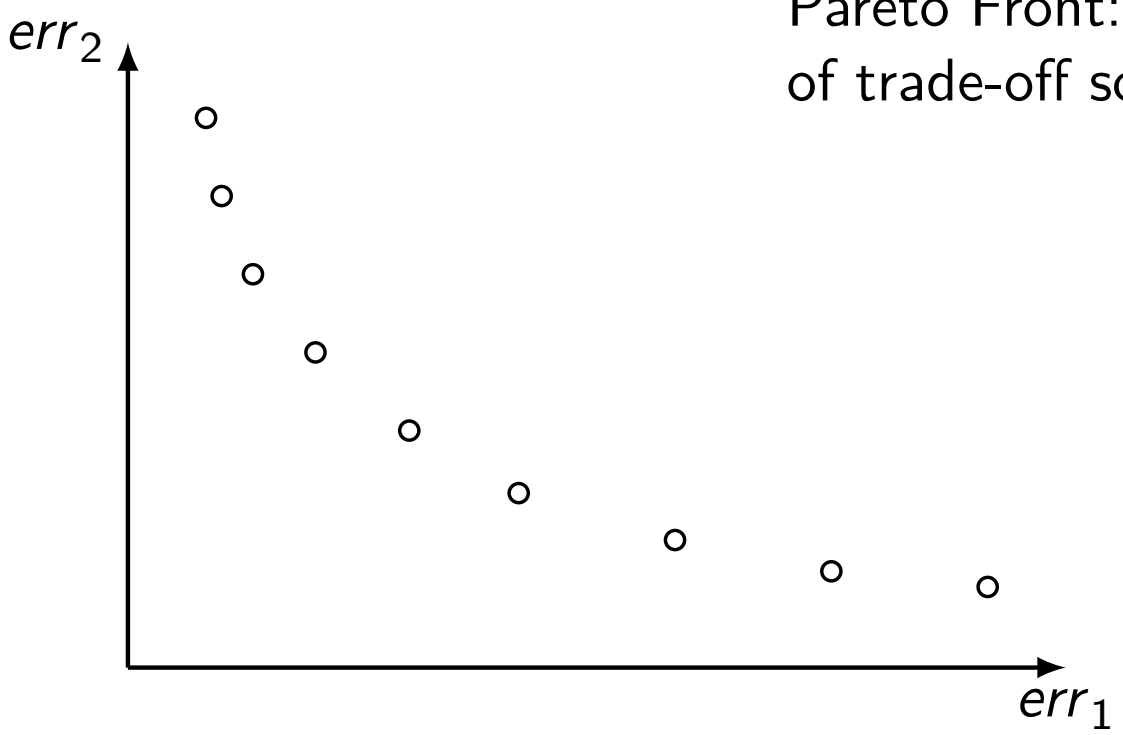
Seismic History Matching by Evolutionary Algorithms

Weighted sum aggregation

- Hard to decide weight

Multi-Objective Optimization

- Optimize conflicting objectives simultaneously
- Applications in SHM: NSGA-II[3], RVEA[4]



Pareto Front: best set of trade-off solutions

Research Objective of Case Study I

How to aggregate the objective functions in the SHM problems?

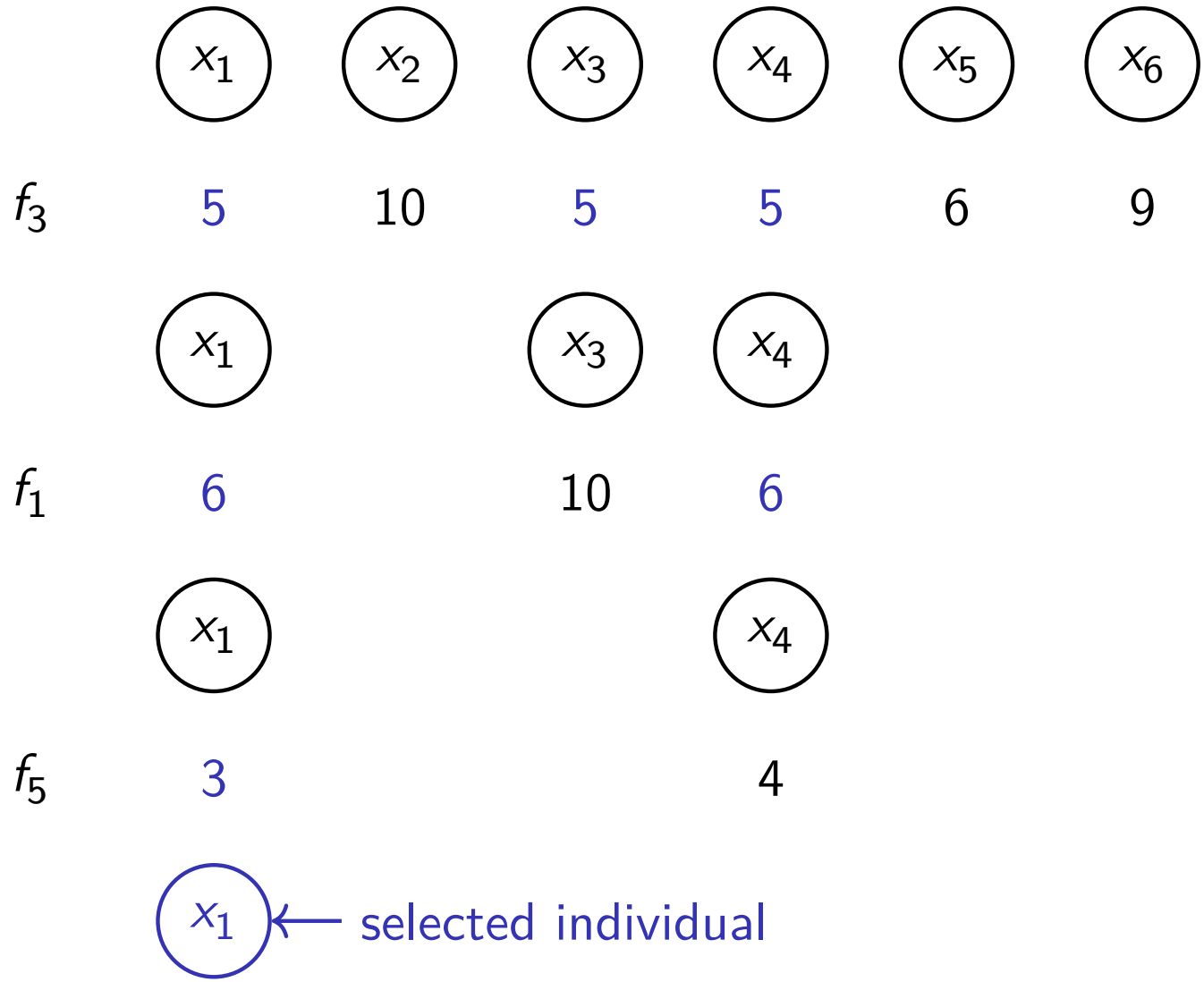
1. No weight assignment is needed
2. Solutions perform well on all objectives

For a SHM problem, there is only one true model in the real world

- All objective functions should share same optimal solution
- Our idea of GKT should apply to this problem

Lexicase Selection[5]

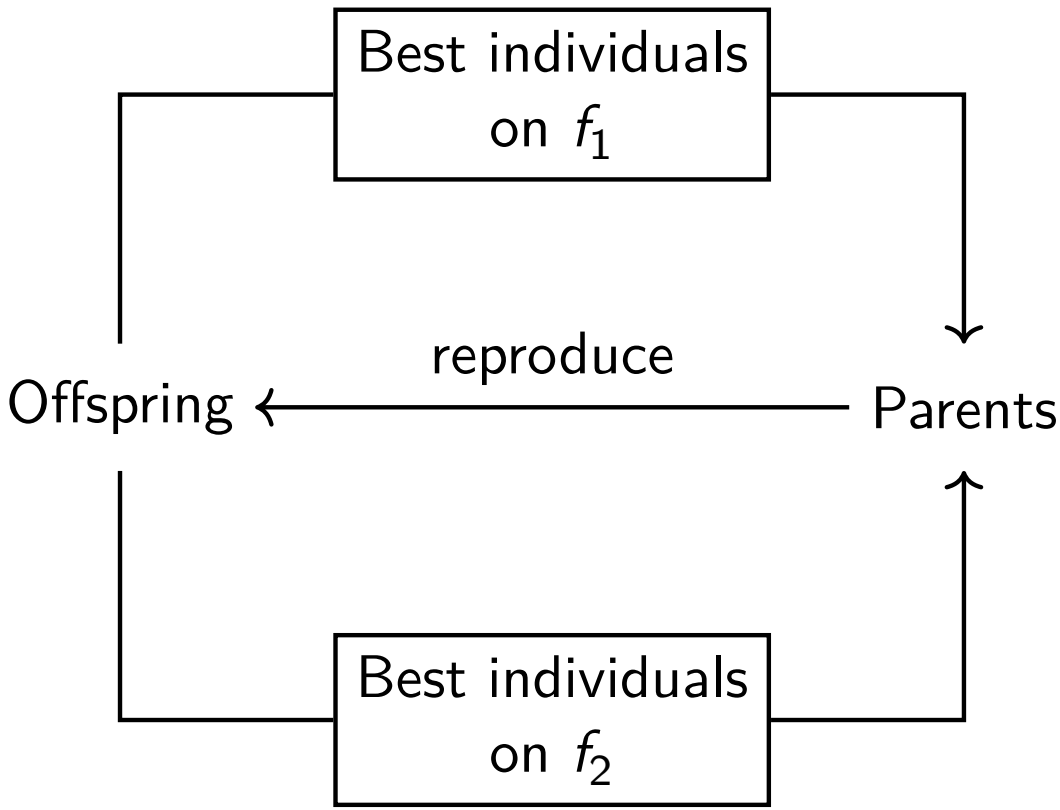
Filter the population based on shuffled list of objective functions



Genetic Knowledge Transfer in Lexicase Selection

Consider Lexicase Selection with 2-objective case

1. $[f_1, f_2] \rightarrow$ select individuals with best fitness on f_1
2. $[f_2, f_1] \rightarrow$ select individuals with best fitness on f_2



Proposed Differential Evolution with Lexicase Selection

Differential mutation

$$\mathbf{y} = \mathbf{x}_{lex} + F \cdot (\mathbf{x}_{r_1} - \mathbf{x}_{r_2})$$

- $\mathbf{x}_{lex}$: selected by Lexicase Selection
- $\mathbf{x}_{r_1}$, $\mathbf{x}_{r_2}$: selected by random selection
- F: scaling factor

Other modifications

- Omit the survival selection
- Introduce polynomial mutation after differential mutation

We call this method Lex-DE

Experimental Methods

Test problem: Volve dataset[6]

- Two seismic objectives: Seis-mean, Seis-spa
- Three well log objectives: P-F-12, P-F-14, P-F-15C

Methods to compare

- Lex-DE (proposed method)
- NSGA-II[3]: most common Multi-Objective method on SHM
- RVEA[4]: Multi-Objective method using decomposition

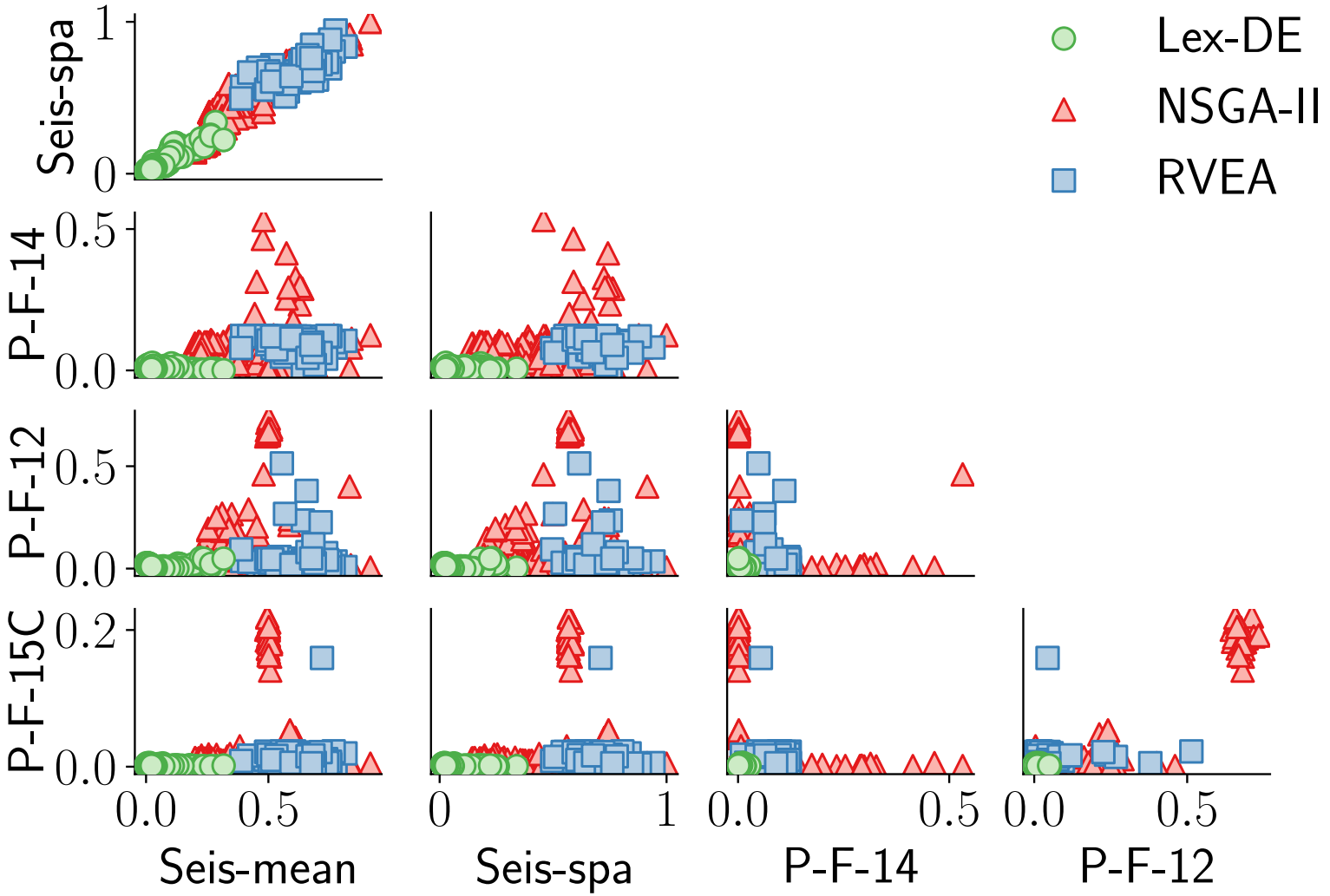
Parameters

- population size = 20, max generation = 100
  - SHM is heavy[a], small number of evaluation is common
  - E.g., SHM by RVEA[4]: pop. size = 20, max gen. = 75
- Five repetition

[a] It cost us about one week to run five repetitions of one method

Non-dominated Solutions of Best Runs in Objective Space

The objective values are scaled into $[0, 1]$ based on non-dominated solutions in all runs



Summary of Case Study I

Objective aggregation for SHM problems

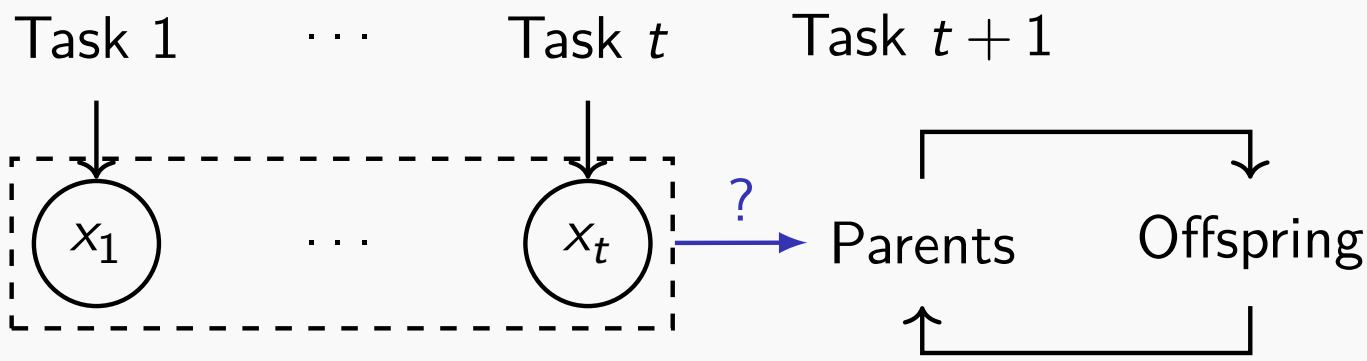
1. Applied the idea of Naive Genetic Knowledge Transfer (NGKT)
2. Lexicase Selection
   - A variant of NGKT between multiple tasks with only one population
3. Proposed Differential Evolution based on Lexicase Selection
   - Better performance than NSGA-II and RVEA
   - More concentrated solution set and closer to ground truth

Part II: Adaptive Transfer of Genetic Knowledge

Genetic Knowledge Transfer in Sequential Problem-solving

GKT between EA that solved past tasks and EA that is going to solve future tasks

NGKT among Many Distinct Tasks: An Example



Solved t (a large number) tasks that are not necessarily similar

- Final solution as genetic knowledge
- Not all genetic knowledge from the t tasks are helpful

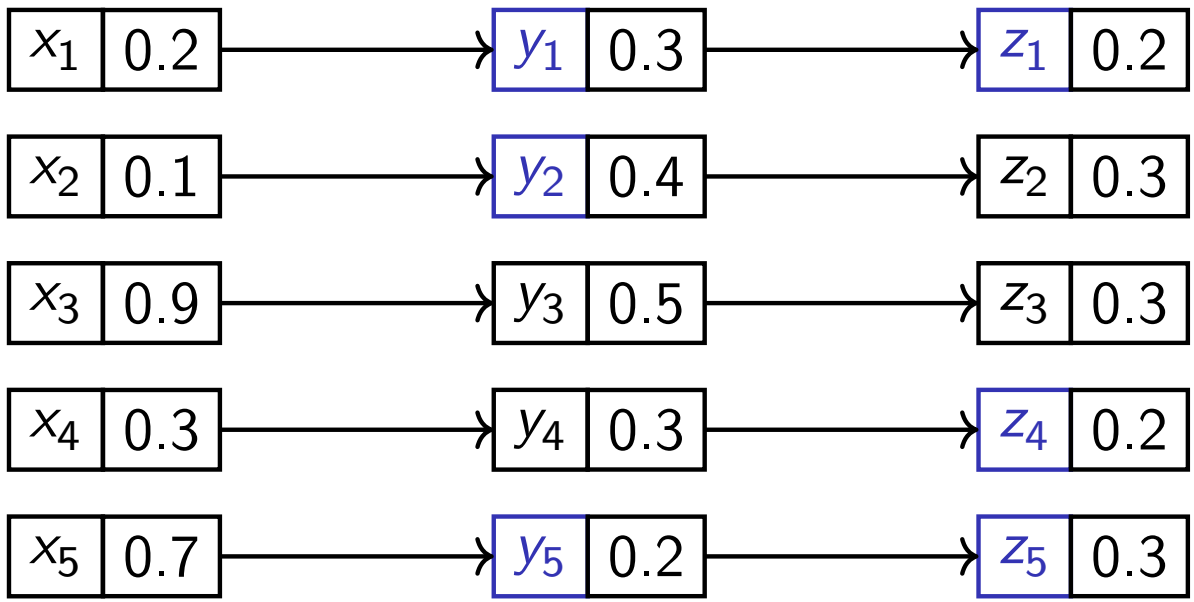
How can an EA automatically discover helpful genetic knowledge when it has solved too many distinct tasks?

- There are both helpful and unhelpful genetic knowledge

Self-adaptive Evolutionary Algorithms[7]

Self-adaptive EAs automatically find the best parameter

- E.g., best mutation rate of Genetic Algorithm from $[0, 1]$



Automatic Discovery of Helpful Genetic Knowledge

Find helpful genetic knowledge for Task $t+1$ from previous t tasks

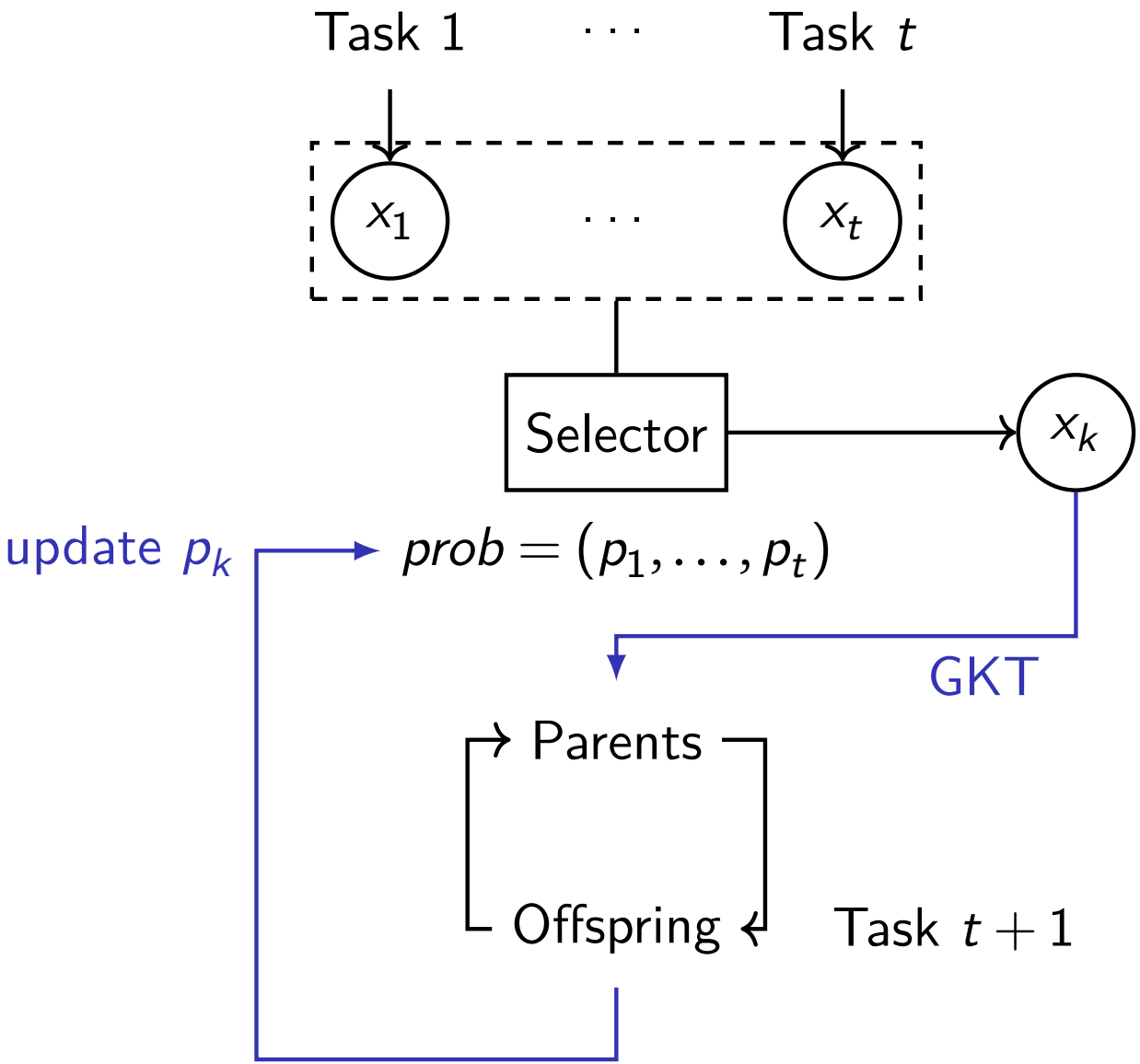
- Assume that we use final solutions as genetic knowledge
- $\rightarrow$ Find helpful solutions among t solutions

Discovering Helpful Genetic Knowledge by Self-adaptation

1. Select a solution s by probability
2. Crossover an individual in EA (that solves Task $t+1$) with the selected solution
3. If the child is better, increase the probability to select s

We call this idea the adaptive transfer of genetic knowledge

Adaptive Transfer of Genetic Knowledge



Case Study II: Knowledge-Driven Program Synthesis

Overview of Case Study II

Research Question

How can a Genetic Programming algorithm improve itself when solving a sequence of many distinct program synthesis problems?

Methodology

We apply the idea of adaptive transfer of genetic knowledge

1. We use subprograms as genetic knowledge
2. We propose a system that solves tasks, extracts subprograms, and reuses subprograms by adaptive transfer

Program Synthesis

Find a sequence of instructions $\mathbf{x}$ from the available set $\mathscr{I}$ that satisfies a set of I/O examples $\{in_k, out_k\}_{k=1}^{M}$

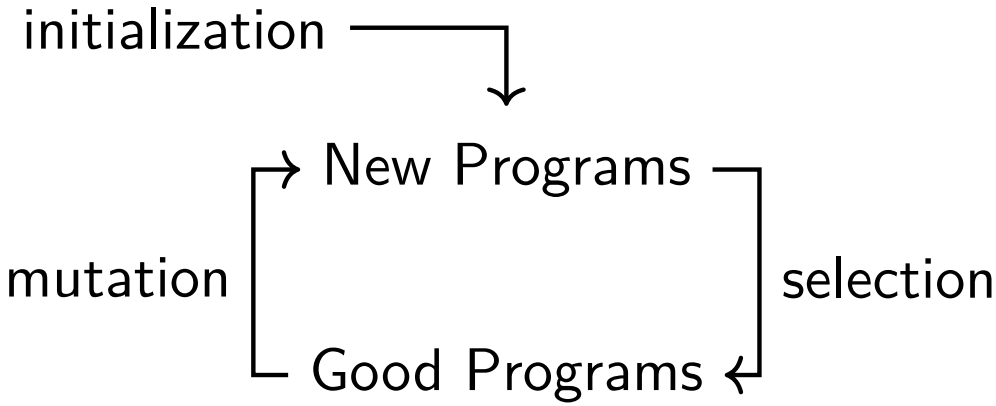
$$\begin{aligned} minimize \quad & \Sigma_{k=1}^{M} ||\mathbf{x}(in_k) - out_k|| \\ s.t. \quad & \mathbf{x} = (x_1, \ldots, x_n) \\ & x_i \in \mathscr{I} \end{aligned}$$

Example of Program Synthesis Problem

- Instruction set: { `x`, `y`, `add`, `sub`, `mult`, `div`, `1`, `0` }
- I/O examples
  - in=(1, 1), out=0
  - in=(2, 4), out=4
- Program: `mult(sub(x,y),sub(y,x))`

Genetic Programming (GP)

EAs that solve program synthesis problems



Applications of GP

- Neural Architecture Search
- Image Classification and Processing
- Feature Selection, Extraction, and Construction
- Trading Rule Extraction
- Evolvable Hardware (e.g., robotics, circuit)

Research Objective of Case Study II

Humans reuse the code fragments (subprograms) that they wrote in the past

- E.g., using libraries (`import numpy as np`)
- Allow humans to create more complex programs

Can GP do similar thing?

- Humans solve a lot of tasks
- Humans solve distinct tasks

How can a GP algorithm improve itself when solving a sequence of many distinct tasks through subprogram reuse?

Subprogram Reuse in Genetic Programming

Reuse in one task

- Automatically Defined Functions[8]

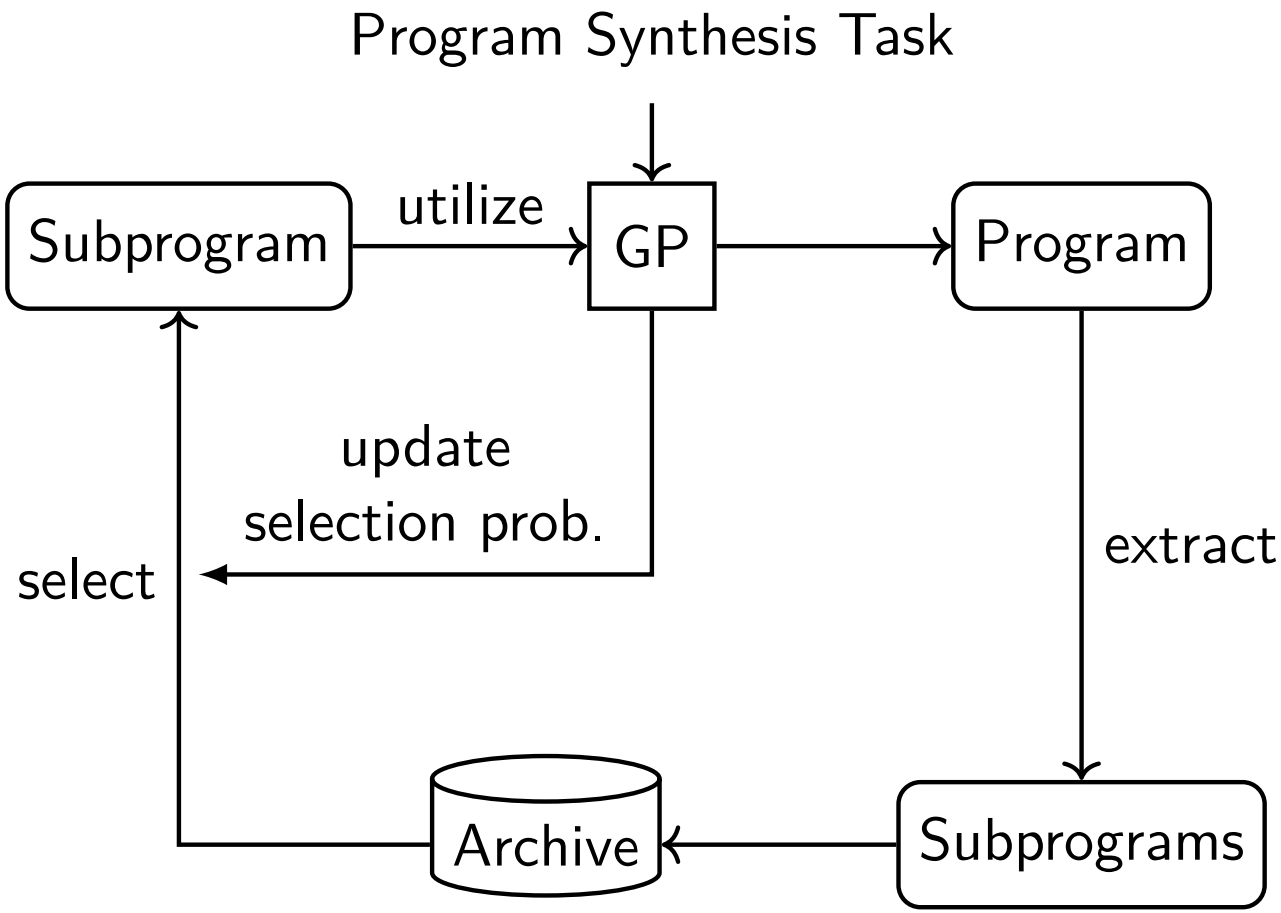
Reuse across multiple similar tasks

- Instruction set on several similar problems[9]
- Random subtrees from previous image classification task[2, 10]
- Common subtrees in two tasks[11]
- Final population of the last task[12]

Reuse across many distinct tasks

- Our method!

Proposed Knowledge-Driven Program Synthesis System



Components: Subprogram Extraction and Utilization

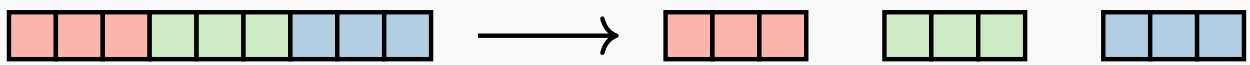
PushGP[13]

- A program is encoded as a list
- Invalid instructions in the program are ignored

Proposed Extraction Method: Even Partitioning (EP)

Given a program, how to get subprograms?

- Divide a program (list) into subprograms with equal length



Proposed Utilization Method: Replacement Mutation (RM)

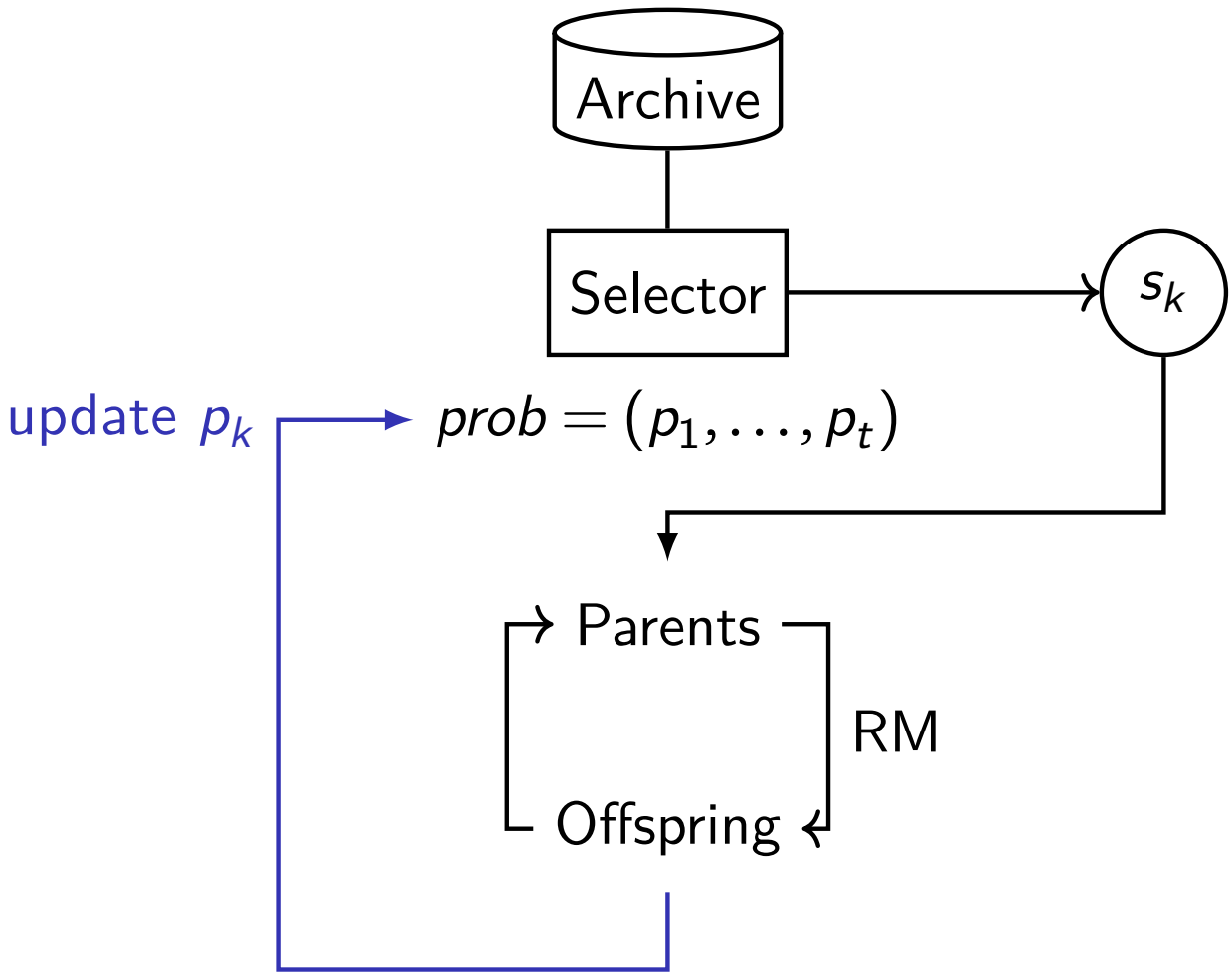
Given a subprogram, how to use it with PushGP?

- Replace a random part of the parent with the subprogram

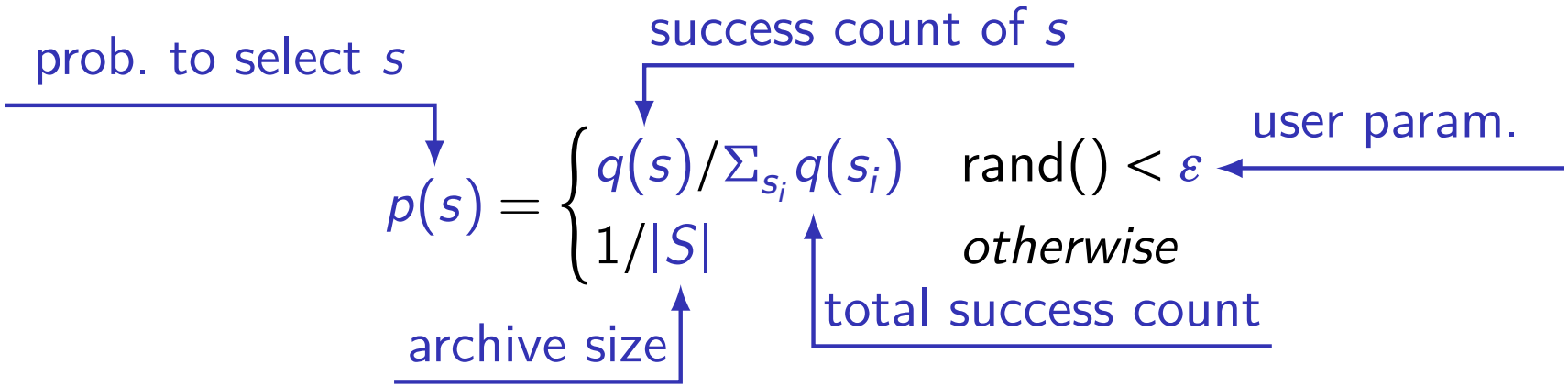
Components: Subprogram Selection

Given an archive of subprograms, how to select helpful ones?

- We use the idea of adaptive transfer of genetic knowledge



Components: Update Strategy of Selection Probability



Success count $q(s)$: if the child generated by RM with s satisfies more I/O examples than its parents, $q(s) \leftarrow q(s) + 1$

Intuition

1. Exploit subprograms with high success count by probability ε
2. Explore all subprograms by probability $1 - \varepsilon$

Demonstration of Subprogram Selection Strategy

Small or Large: given an integer n, print "small" if $n < 1000$ and "large" if $n \geq 2000$

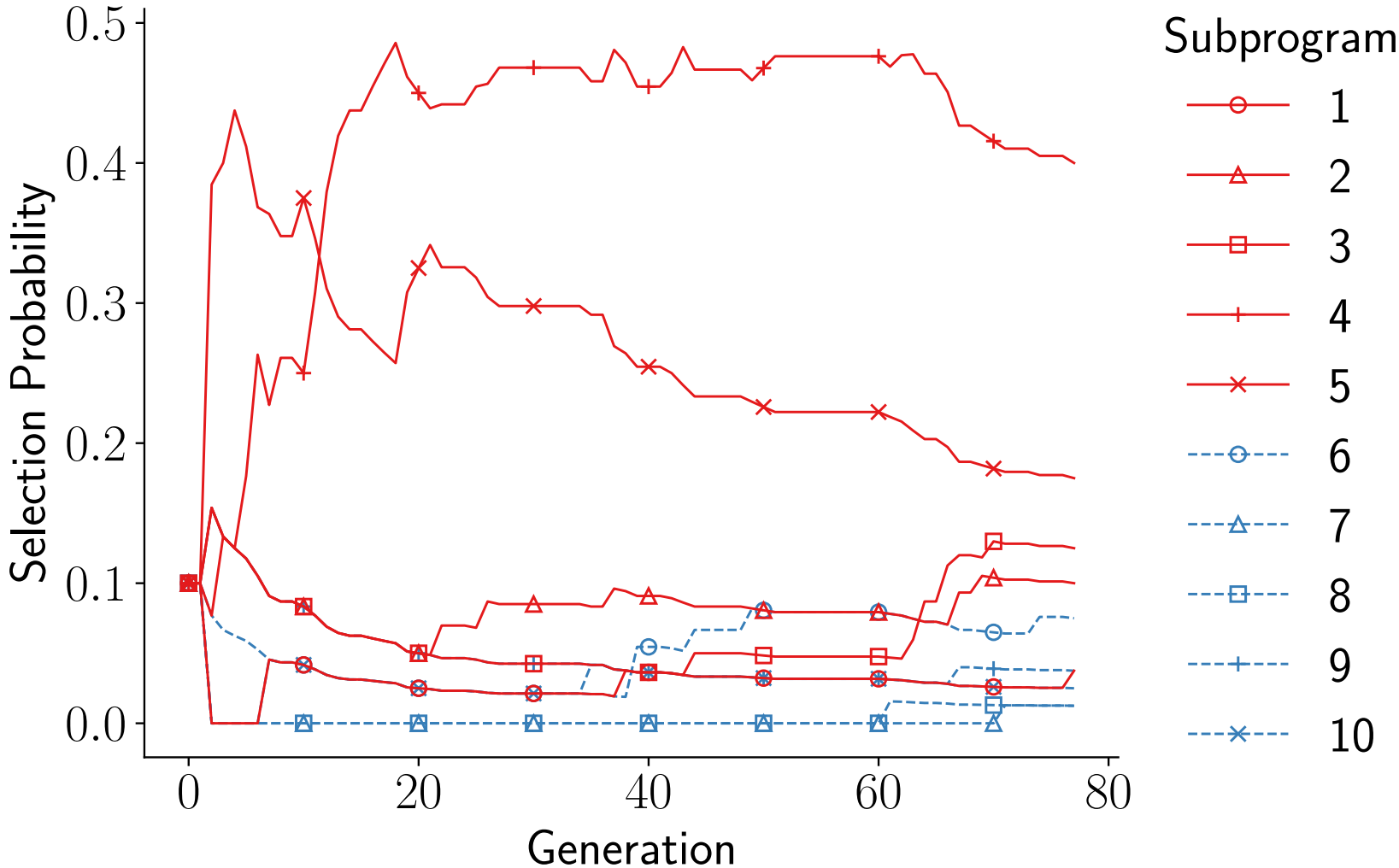
- I/O examples: {[in=500, out="small"], [in=3000, out="large"], ...}
- Correct program: `input_0 2000 int_gte exec_if ("large" print_str) (input_0 1000 int_lt exec_if ("small" print_str))`

Table: Subprogram archive (1–5 from solution; 6–10 randomly generated)

No.	Subprogram (helpful)	No.	Subprogram (unhelpful)
1	`exec_if`	6	`print_int`
2	`input_0 1000 int_lt`	7	`bool_is_empty exec_while str_remove_all_str`
3	`input_0 2000 int_gte`	8	`str_remove_first_str print_bool str_replace_first_str`
4	`"small" print_str`	9	`exec_shove exec_eq`
5	`"large" print_str`	10	`str_but_last bool_stack_depth`

Demonstration of Subprogram Selection Strategy

```
input_0 "large" input_0 1000 int_lt 1000 input_0 2000 int_gte int_lte
exec_when (exec_do_while () "small") exec_if (print_str) ()
```



Pilot Experiments on Subprogram Selection Strategy

Verify the effectiveness of self-adaptive strategy

Select four simple problems from a benchmark[14]

- Median (MD), Compare String Lengths (CSL), Small or Large (SL), and Count Odds (CO)

Create two composite problems

- C_1: composite of SL and CSL
- C_2: composite of MD and CO

Subprograms by hand from simple problems (five for each)

- For every composite problem, some subprograms in the archive are helpful while the rest are unhelpful

Pilot Experiments on Subprogram Selection Strategy

C_1: given a string n, print "small" if $len(n) < 1000$ and "large" if $len(n) \geq 2000$

- Correct program: `in_0 str_len 2000 int_gte exec_if ("large" print_str) (in_0 str_len 1000 int_lt exec_if ("small" print_str))`
- Subprograms in the archive
  - SL: `(exec_if)`, `(in_0 2000 int_gte)`, `(in_0 1000 int_lt)`, `("small" print_str)`, `("large" print_str)`
  - CSL: `(in_0 str_len)`, `(in_1 str_len)`, `(in_2 str_len)`, `(int_lt)`, `(bool_and)`
  - CO: `(in_0 vec_iter)`, `(2 int_mod)`, `(exe_if)`, `(1 int_eq)`, `(int_inc)`
  - MD: `(int_min int_max)`, `(int_max int_min)`, `(in_0 in_1)`, `(in_2)`, `(print_int)`

Pilot Experiments on Subprogram Selection Strategy

Methods to compare

- Adaptive selection of subprograms
- Random selection of subprograms
- Original PushGP[13] without using subprograms

Parameters

- population size = 200, max generation = 300
- 50% time to do RM, 50% time to do UMAD[13]
- $\varepsilon = 0.5$

Table: Number of Successful Runs in 21 Repetitions (_: p-value<0.05)

Method	C_1	C_2
Adaptive Selection	**8**	**8**
Random Selection	0	1
Original PushGP	0	1

Experiments on Sequential Problem-solving

Test problems

- MD, CSL, and SL from the benchmark[14]
- Every two simple problems, we make a composite problem
  - Median String Lengths (MSL), Small or Large Median (SLM), Small or Large String (SLS)

Methods to compare

- Proposed method: PushGP+ Even Partitioning + Adaptive selection + Replacement Mutation (PushGP+EP+ARM)
- Original PushGP[13]

Parameters

- population size = 1000, max generation = 300
- 10% time to do RM, 90% time to do UMAD[13]
- $\varepsilon = 0.5$

Experiments on Sequential Problem-solving

Table: Simple → Complex: Number of Successful Runs in 25 Repetitions

Method	MD	CSL	SL	MSL	SLM	SLS
PushGP+EP+ARM	**19**	6	4	**9**	**1**	**5**
PushGP	18	**8**	**7**	4	0	1

Table: Complex → Simple: Number of Successful Runs in 25 Repetitions

Method	SLS	SLM	MSL	SL	CSL	MD
PushGP+EP+ARM	**2**	**3**	**6**	**12**	**16**	17
PushGP	1	0	4	7	8	**18**

Summary of Case Study II

GP that improves itself when solving a sequence of many distinct tasks

1. Applied the idea of adaptive transfer of genetic knowledge
2. Self-adaptive strategy to select helpful subprograms
   - More successful runs than random selection strategy
3. Implemented Knowledge-Driven Program Synthesis system with PushGP+EP+ARM
   - Better than PushGP if there are helpful subprograms in the archive
   - Worse than PushGP if there is no helpful subprograms

Conclusions

Can EA learn and improve itself when solving a sequence of many distinct tasks like a human?

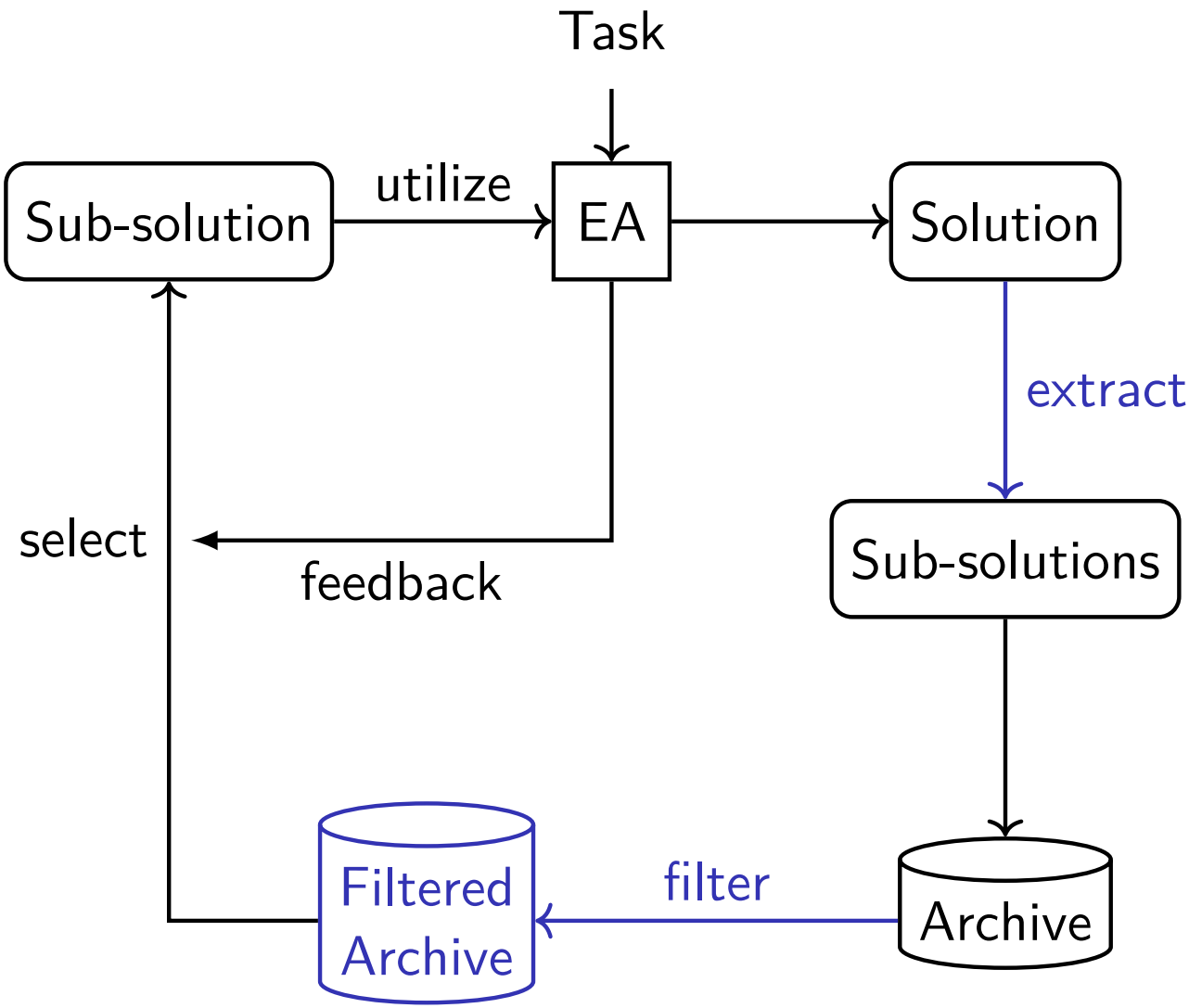
Literature on multiple similar tasks

- Genetic knowledge from past tasks

Main difficulty of many distinct tasks: unhelpful genetic knowledge in the archive

By online search based on self-adaptive strategy, we can automatically discover helpful genetic knowledge and improve the performance of the EA

Future Work: Adaptive Genetic Knowledge Transfer



List of Publications

Journals

1. Yifan He, Claus Aranha, Antony Hallam, and Romain Chassagne. Optimization of subsurface models with multiple criteria using lexicase selection. *Operations Research Perspectives*, 9:100237, 2022.
2. Antony Hallam, Romain Chassagne, Claus Aranha, and Yifan He. Comparison of map metrics as fitness input for assisted seismic history matching. *Journal of Geophysics and Engineering*, 19(3):457-474, 2022.
3. Yifan He and Claus Aranha. Solving portfolio optimization problems using moea/d and lévy flight. *Advances in Data Science and Adaptive Analysis*, 12(03n04):2050005, 2020.

Peer-Reviewed Conferences

4. Yifan He, Claus Aranha, and Tetsuya Sakurai. Knowledge-driven program synthesis via adaptive replacement mutation and auto-constructed subprogram archives. In *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 14-21. IEEE, 2022.
5. Yifan He, Claus Aranha, and Tetsuya Sakurai. Incorporating sub-programs as knowledge in program synthesis by pushgp and adaptive replacement mutation. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 554-557, 2022.
6. Yifan He, Claus Aranha, and Tetsuya Sakurai. Parameter evolution self-adaptive strategy and its application for cuckoo search. In *International Conference on Bioinspired Methods and Their Applications*, pages 56-68. Springer, 2020.

Bibliography I

[1] Genghui Li, Qingfu Zhang, and Weifeng Gao.
Multipopulation evolution framework for multifactorial optimization.
In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 215–216, 2018.

[2] Muhammad Iqbal, Bing Xue, and Mengjie Zhang.
Reusing extracted knowledge in genetic programming to solve complex texture image classification problems.
In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 117–129. Springer, 2016.

[3] Elham Yasari, Mahmoud Reza Pishvaie, Farhad Khorasheh, Karim Salahshoor, and Riyaz Kharrat.
Application of multi-criterion robust optimization in water-flooding of oil reservoir.
Journal of Petroleum Science and Engineering, 109:1–11, 2013.

[4] Junko Hutahaean, Vasily Demyanov, and Mike Christie.
Many-objective optimization algorithm applied to history matching.
In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8. IEEE, 2016.

[5] Thomas Helmuth, Lee Spector, and James Matheson.
Solving uncompromising problems with lexicase selection.
IEEE Transactions on Evolutionary Computation, 19(5):630–643, 2014.

[6] Tony Hallam, Colin MacBeth, Romain Chassagne, and Hamed Amini.
4d seismic study of the volve field-an open subsurface-dataset.
First Break, 38(2):59–70, 2020.

[7] Jingqiao Zhang and Arthur C Sanderson.
Jade: adaptive differential evolution with optional external archive.
IEEE Transactions on evolutionary computation, 13(5):945–958, 2009.

[8] John R Koza.
Genetic programming as a means for programming computers by natural selection.
Statistics and computing, 4(2):87–112, 1994.

Bibliography II

[9] Maarten Keijzer, Conor Ryan, Gearoid Murphy, and Mike Cattolico.
Undirected training of run transferable libraries.
In *European Conference on Genetic Programming*, pages 361–370. Springer, 2005.

[10] Muhammad Iqbal, Bing Xue, Harith Al-Sahaf, and Mengjie Zhang.
Cross-domain reuse of extracted knowledge in genetic programming for image classification.
IEEE Transactions on Evolutionary Computation, 21(4):569–587, 2017.

[11] Damien O'Neill, Harith Al-Sahaf, Bing Xue, and Mengjie Zhang.
Common subtrees in related problems: A novel transfer learning approach for genetic programming.
In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1287–1294. IEEE, 2017.

[12] Jordan Wick, Erik Hemberg, and Una-May OReilly.
Getting a head start on program synthesis with genetic programming.
In *European Conference on Genetic Programming (Part of EvoStar)*, pages 263–279. Springer, 2021.

[13] Thomas Helmuth, Nicholas Freitag McPhee, and Lee Spector.
Program synthesis using uniform mutation by addition and deletion.
In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1127–1134, 2018.

[14] Thomas Helmuth and Lee Spector.
General program synthesis benchmark suite.
In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, pages 1039–1046, 2015.

Details of Simple Problems

Median (MD): given three integers, print their median

Compare String Lengths (CSL): given three strings n_1, n_2, and n_3, return true if $len(n_1) < len(n_2) < len(n_3)$, and false otherwise

Small or Large (SL): given an integer n, print "small" if $n < 1000$ and "large" if $n \geq 2000$ (and nothing if $1000 \leq n < 2000$)

Count Odds (CO): given a vector of integers, return the number of integers that are odd, without use of a specific even or odd instruction (but allowing instructions such as mod and quotient)

Details of Composite Problems in Pilot Experiments

C_1: given a string n, print "small" if $len(n) < 1000$ and "large" if $len(n) \geq 2000$ (and nothing if $1000 \leq len(n) < 2000$)

C_2: given a vector of three integers, return true if the median is even and false if the median is odd, without use of a specific even or odd instruction (but allowing instructions such as mod and quotient)

Details of Composite Problems in Experiments

Median String Length (MSL): given 3 strings, print the median of their lengths

Small or Large Median (SLM): given 4 integers a, b, c, d, print "small" if $median(a, b, c) < d$ and "large" if $median(a, b, c) > d$ (and nothing if $median(a, b, c) = d$)

Small or Large String (SLS): given a string n, print "small" if $len(n) < 100$ and "large" if $len(n) \geq 200$ (and nothing if $100 \leq len(n) < 200$)