EcoSearch: A Constant-Delay Best-First Search Algorithm for Program Synthesis

Paper ID #10442

Théo Matricon
Nathanaël Fijalkow
Guillaume Lagarde

AAAI'25 - 28/02/2025

Program Synthesis?

An old dream: Church's Problem (1957)

Program Synthesis?

An old dream: Church's Problem (1957)

Logical formulas

Specification = **ϕ** a logical formula
A program **P** such that for all x, ϕ(x, P(x)) = True

Natural language

« A program that removes odd elements and sort the rest »

A set of I/O examples

```
[1, 5, 4, 2]  ⟶  [2, 4]
[6, 3, 0, 8]  ⟶  [0, 6, 8]
```

Program Synthesis?

An old dream: Church's Problem (1957)

Logical formulas

Specification = **ϕ** a logical formula
A program **P** such that for all x, ϕ(x, P(x)) = True

Natural language

« A program that removes odd elements and sort the rest »

A set of I/O examples

```
[1, 5, 4, 2]  ⟶  [2, 4]
[6, 3, 0, 8]  ⟶  [0, 6, 8]
```

SORT; FILTER(EVEN)

DeepCoder

Microsoft (Balog et al., 2017) — it manipulates list of integers

Program 4:

```
x ← [int]
y ← [int]
c ← SORT x
d ← SORT y
e ← REVERSE d
f ← ZIPWITH (*) d e
g ← SUM f
```

Input-output example:

Input:

```
[7 3 8 2 5],
[2 8 9 1 3]
```

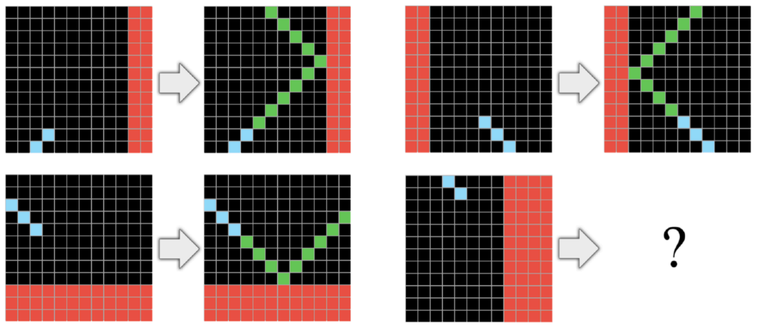
Output:

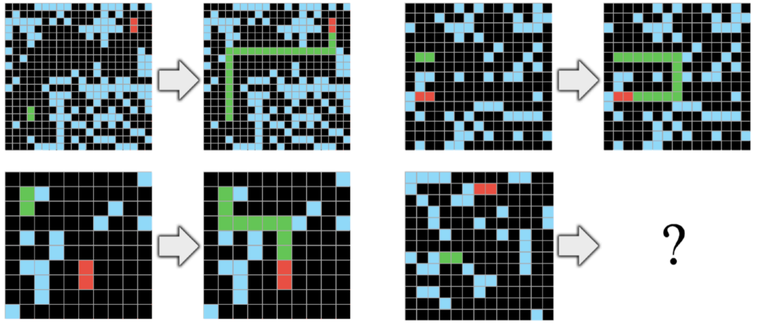
```
79
```

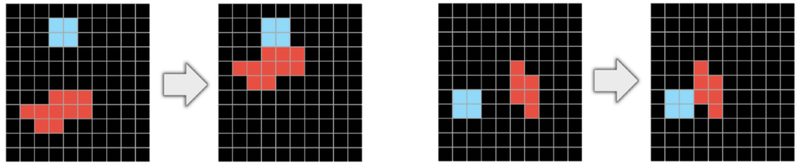
Description:
Xavier and Yasmine are laying sticks to form non-overlapping rectangles on the ground. They both have fixed sets of pairs of sticks of certain lengths (represented as arrays `x` and `y` of numbers). Xavier only lays sticks parallel to the x axis, and Yasmine lays sticks only parallel to y axis. Compute the area their rectangles will cover at least.

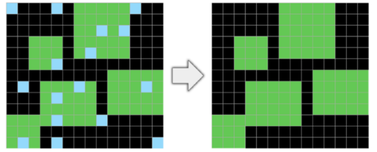
ARC Dataset

« The Abstraction and Reasoning Corpus », in « On the measure of intelligence » François Chollet, 2019







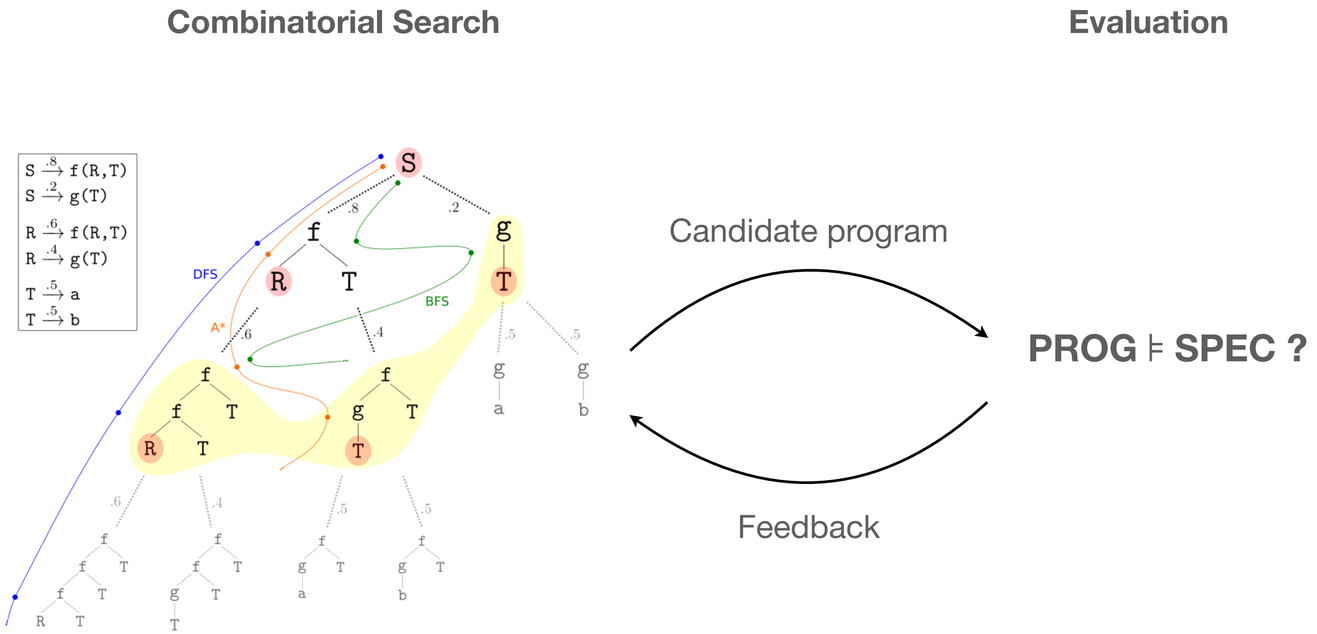


Cost-Guided Program Synthesis

Combination of formal methods and machine learning

reliable

efficient



Cost-Guided Program Synthesis

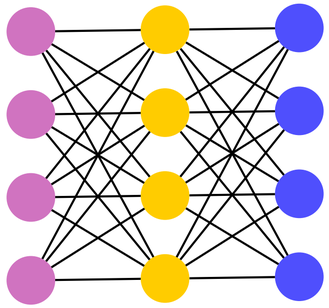
Combination of formal methods and machine learning

reliable

efficient

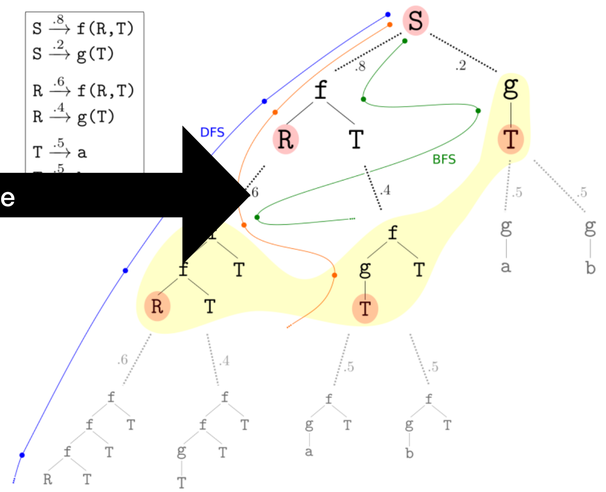
Heuristic cost function

w : **Program** $\rightarrow \mathbb{R}_{>0}$



Influence

Combinatorial Search



Evaluation

Candidate program

PROG ⊨ SPEC ?

Feedback

First practical instance: DeepCoder, 2017

DSL = Context-free Grammar (CFG)

CFG

$$
\begin{aligned}
&r_1 : \text{str} \rightarrow \text{"Hello"} \\
&r_2 : \text{str} \rightarrow \text{"World"} \\
&r_3 : \text{str} \rightarrow \text{cast}(\text{int}) \\
&r_4 : \text{str} \rightarrow \text{concat}(\text{str}, \text{str}) \\
&r_5 : \text{int} \rightarrow \text{var} \\
&r_6 : \text{int} \rightarrow 1 \\
&r_7 : \text{int} \rightarrow \text{add}(\text{int}, \text{int})
\end{aligned}
$$

DSL = Context-free Grammar (CFG)

CFG

$$
\begin{aligned}
r_1 &: \text{str} \rightarrow \text{”Hello”} \\
r_2 &: \text{str} \rightarrow \text{”World”} \\
r_3 &: \text{str} \rightarrow \text{cast}(\text{int}) \\
r_4 &: \text{str} \rightarrow \text{concat}(\text{str}, \text{str}) \\
r_5 &: \text{int} \rightarrow \text{var} \\
r_6 &: \text{int} \rightarrow 1 \\
r_7 &: \text{int} \rightarrow \text{add}(\text{int}, \text{int})
\end{aligned}
$$

concat(“Hello", cast(add(var,1)))

From CFG to Weighted CFG

CFG

$r_1 : \text{str} \rightarrow \text{"Hello"}$
$r_2 : \text{str} \rightarrow \text{"World"}$
$r_3 : \text{str} \rightarrow \text{cast}(\text{int})$
$r_4 : \text{str} \rightarrow \text{concat}(\text{str}, \text{str})$
$r_5 : \text{int} \rightarrow \text{var}$
$r_6 : \text{int} \rightarrow 1$
$r_7 : \text{int} \rightarrow \text{add}(\text{int}, \text{int})$

concat(“Hello", cast(add(var,1)))

WCFG

$r_1 : \text{str} \rightarrow \text{"Hello"}$	$\text{cost} : 1.1$
$r_2 : \text{str} \rightarrow \text{"World"}$	$\text{cost} : 2.0$
$r_3 : \text{str} \rightarrow \text{cast}(\text{int})$	$\text{cost} : 4.4$
$r_4 : \text{str} \rightarrow \text{concat}(\text{str}, \text{str})$	$\text{cost} : 5.3$
$r_5 : \text{int} \rightarrow \text{var}$	$\text{cost} : 1.8$
$r_6 : \text{int} \rightarrow 1$	$\text{cost} : 3.3$
$r_7 : \text{int} \rightarrow \text{add}(\text{int}, \text{int})$	$\text{cost} : 5.3$

From CFG to Weighted CFG

CFG

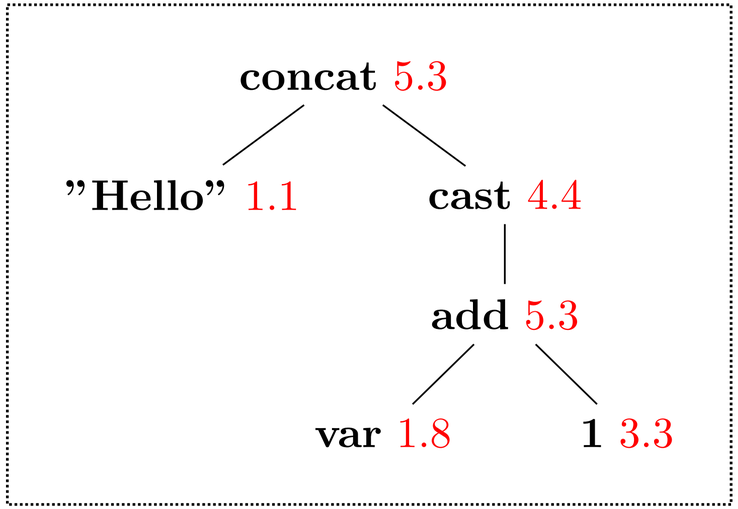
$r_1 : \text{str} \to \text{"Hello"}$
$r_2 : \text{str} \to \text{"World"}$
$r_3 : \text{str} \to \text{cast}(\text{int})$
$r_4 : \text{str} \to \text{concat}(\text{str}, \text{str})$
$r_5 : \text{int} \to \text{var}$
$r_6 : \text{int} \to 1$
$r_7 : \text{int} \to \text{add}(\text{int}, \text{int})$

WCFG

$r_1 : \text{str} \to \text{"Hello"}$	$\text{cost} : 1.1$
$r_2 : \text{str} \to \text{"World"}$	$\text{cost} : 2.0$
$r_3 : \text{str} \to \text{cast}(\text{int})$	$\text{cost} : 4.4$
$r_4 : \text{str} \to \text{concat}(\text{str}, \text{str})$	$\text{cost} : 5.3$
$r_5 : \text{int} \to \text{var}$	$\text{cost} : 1.8$
$r_6 : \text{int} \to 1$	$\text{cost} : 3.3$
$r_7 : \text{int} \to \text{add}(\text{int}, \text{int})$	$\text{cost} : 5.3$

A WCFG *induces* a cost function w over **trees = programs**

concat("Hello", cast(add(var,1)))



Cost = $5.3 + 1.1 + 4.4 + 5.3 + 1.8 + 3.3 = 21.2$

How to use the heuristic for the search?

Best-first search algorithms

Natural strategy.

Given a *heuristic cost function* w : Program $\to \mathbb{R}_{>0}$
Explore the program space in the exact order of non-increasing weights.

Some previous work

- 2017. *A**, Alur et al.
- 2018. *Euphony*, Lee et al.
- 2021. *Dreamcoder*, Ellis et al.
- 2022. *TF-Coder*, Shi et al.
- 2022. *Heap Search*, Fijalkow et al.
- 2023. *Bee Search*, Ameen and Lelis.

Best-first search algorithms

Natural strategy.

Given a *heuristic cost function* w : Program $\to \mathbb{R}_{>0}$

Explore the program space in the exact order of non-increasing weights.

Some previous work

- 2017. *A**, Alur et al.
- 2018. *Euphony*, Lee et al.
- 2021. *Dreamcoder*, Ellis et al.
- 2022. *TF-Coder*, Shi et al.
- 2022. *Heap Search*, Fijalkow et al.
- 2023. *Bee Search*, Ameen and Lelis.

SOTA

- Bottom-up enumeration
- Delay $O(\log n)$
  - i.e., i-th program in time $O(\log i)$

Best-first search algorithms

Natural strategy.

Given a *heuristic cost function* w : Program $\to \mathbb{R}_{>0}$

Explore the program space in the exact order of non-increasing weights.

Some previous work

- 2017. *A**, Alur et al.
- 2018. *Euphony*, Lee et al.
- 2021. *Dreamcoder*, Ellis et al.
- 2022. *TF-Coder*, Shi et al.
- 2022. *Heap Search*, Fijalkow et al.
- 2023. *Bee Search*, Ameen and Lelis.

SOTA

- Bottom-up en
- Delay $O(\log n)$
  - i.e., i-th program in time $O(\log i)$

Is $O(\log n)$ optimal?
Can we achieve O(1)?

Our result — a positive answer

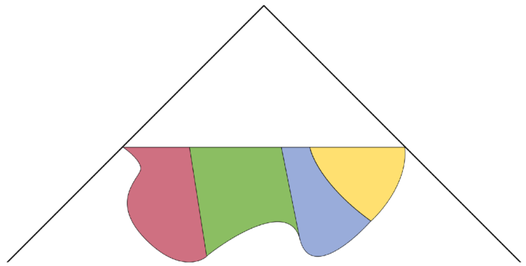
We take the best of both *Heap Search* and *Bee Search*

Eco Search.

- A new best-first bottom-up search algorithm
- Theoretical guarantee → Constant delay
- Performs well on experiments

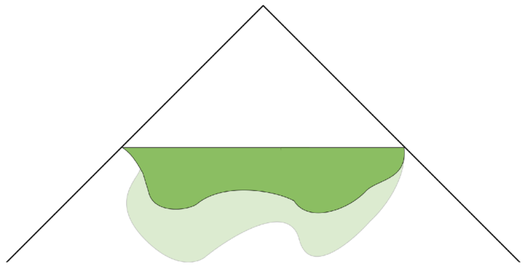
A few insights from the heart of the machine

Heap Search



Bottom-up
Delay $O(\log n)$

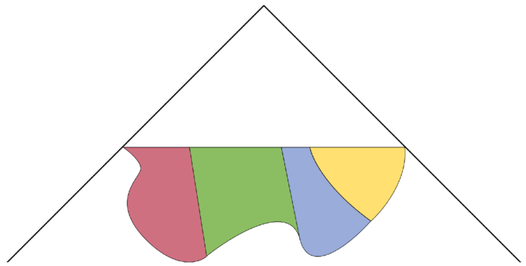
Bee Search



Introduce cost tuple representation
Better frontier expansion

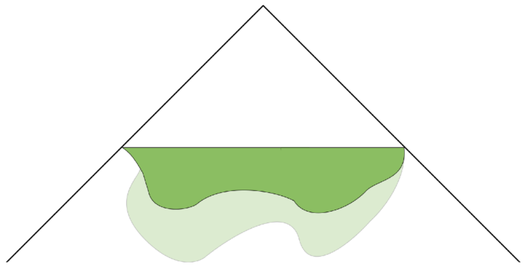
A few insights from the heart of the machine

Heap Search



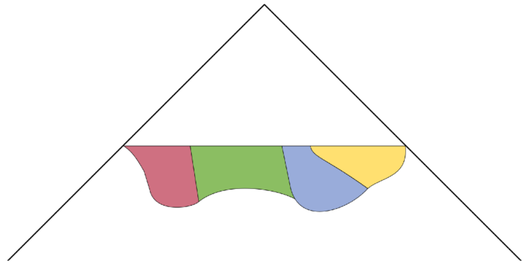
Bottom-up
Delay $O(\log n)$

Bee Search



Introduce cost tuple representation
Better frontier expansion

Eco Search w/o bucketing



Delay $O(\log n)$
Frugal frontier expansion

A few insights from the heart of the machine

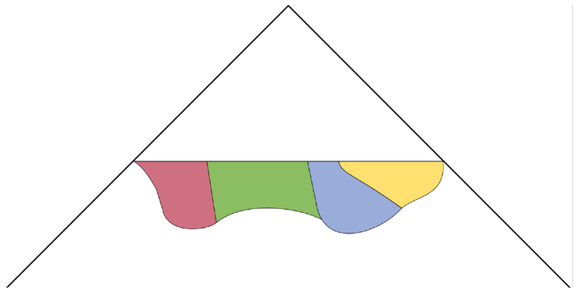
Theorem

There is a constant $M \geq 0$ such that,

for any program p and its successors p'

we have $cost(p') - cost(p) \leq M$

Eco Search w/o bucketing



Delay $O(\log n)$
Frugal frontier expansion

A few insights from the heart of the machine

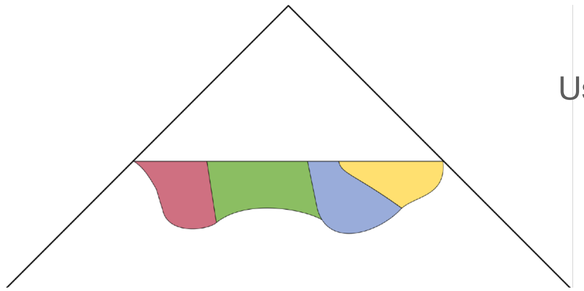
Theorem

There is a constant $M \geq 0$ such that,

for any program p and its successors p'

we have $cost(p') - cost(p) \leq M$

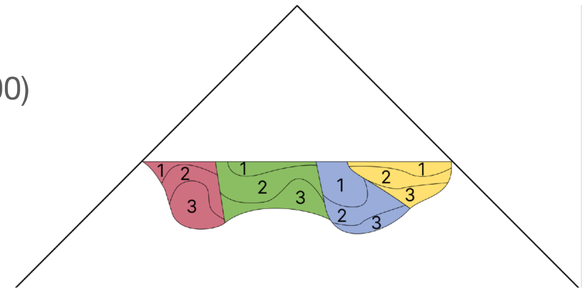
Eco Search w/o bucketing



Delay $O(\log n)$
Frugal frontier expansion

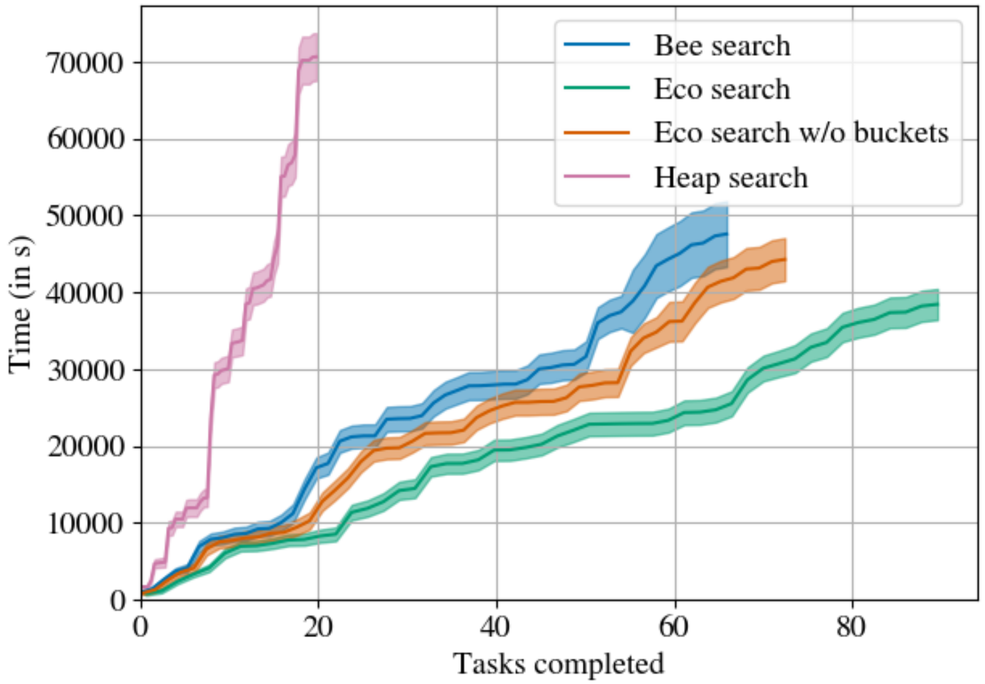
Using bucket queues (Thorup 2000)

Eco Search with bucketing

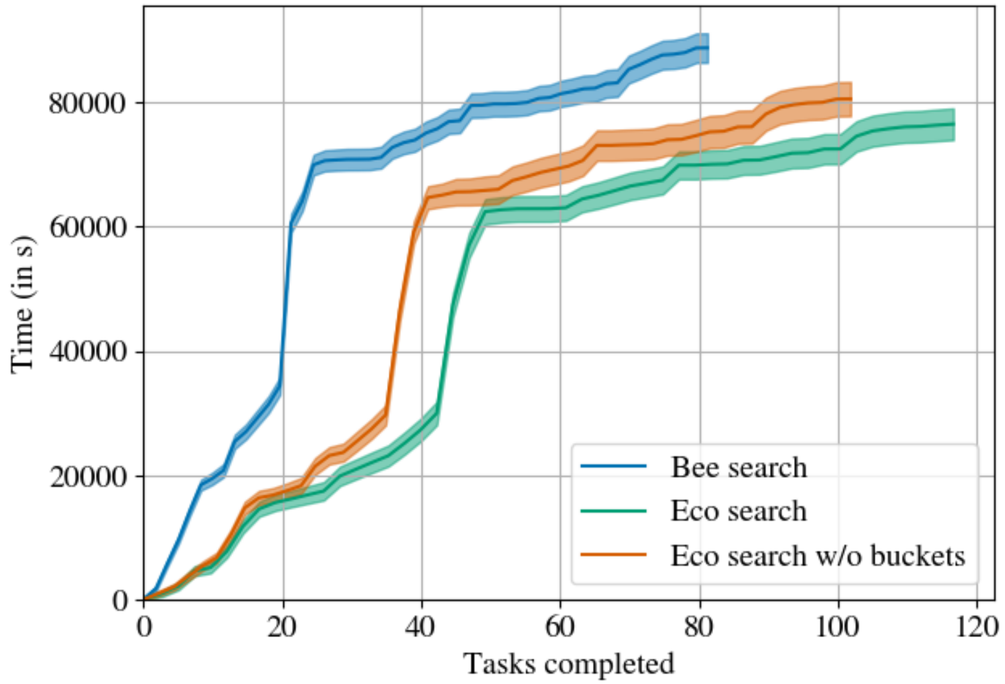


Delay $O(1)$
Integer costs

Experimental results



On the **FlashFill dataset**
- String manipulation
- 200 tasks from SyGuS
- Timeout of 300s



On the **DeepCoder dataset** (Balog et al.)
- Integer list manipulation
- 200 tasks
- Timeout of 300s

Thanks!

- New best-first bottom-up search algorithm
  - use heuristic cost function to guide the search
  - with constant delay
- Check out DeepSynth
  - https://github.com/SynthesisLab/DeepSynth2/tree/eco_search_aaai
- GPU version of the algorithm?
- How to reduce the memory needed? Are there any trade-offs?