Fast and Precise: Adjusting Planning Horizon with Adaptive Subgoal Search

NEURIPS 2022 DEEP RL WORKSHOP, MICHAŁ ZAWALSKI*, MICHAŁ TYROLSKI*, KONRAD CZECHOWSKI*, DAMIAN STACHURA, PIOTR PIĘKOS, TOMASZ ODRZYGÓŹDŹ, YUHUAI WU, ŁUKASZ KUCIŃSKI, PIOTR MIŁOŚ

PRESENTED BY MICHAŁ TYROLSKI

Let say that we hale „hard” environment

Given a problem P, we would like to solve it „efficiently”

We need a search algorithm

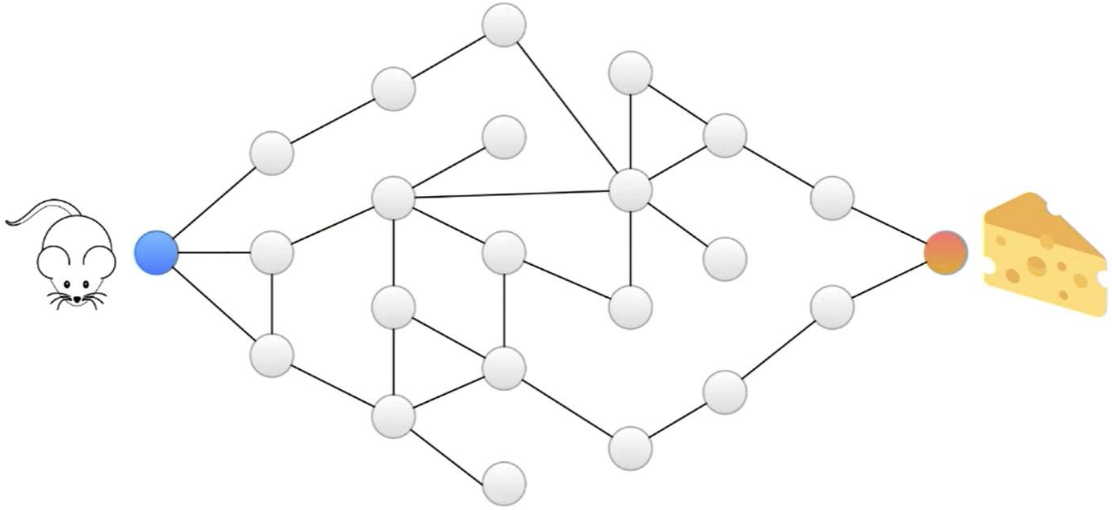
It should be efficient in terms of computional budget

Motivation

Motivation

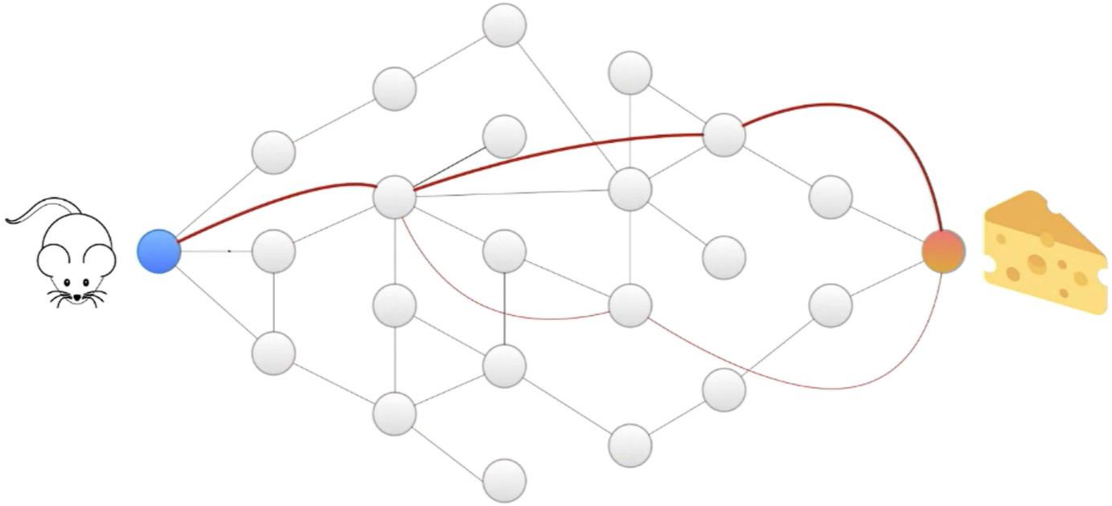
- Let say that we hale „hard” environment
- Given a problem P, we would like to solve it „efficiently”
- We need a search algorithm
- It should be **efficient** in terms of **computional budget**

Subgoal Search



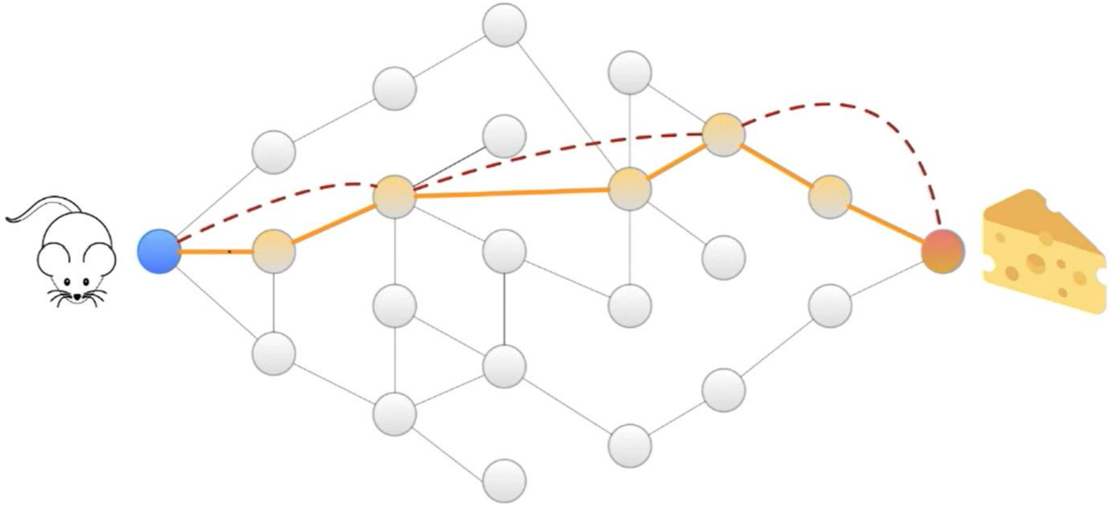
Czechowski, K., Odrzygóźdź, T., Zbysiński, M., Zawalski, M., Olejnik, K., Wu, Y., Kuciński, Ł. and Miłoś, P., 2021. Subgoal search for complex reasoning tasks. *Advances in Neural Information Processing Systems*, *34*, pp.624-638.

Subgoal Search



Czechowski, K., Odrzygóźdź, T., Zbysiński, M., Zawalski, M., Olejnik, K., Wu, Y., Kuciński, Ł. and Miłoś, P., 2021. Subgoal search for complex reasoning tasks. *Advances in Neural Information Processing Systems*, *34*, pp.624-638.

Subgoal Search



Czechowski, K., Odrzygóźdź, T., Zbysiński, M., Zawalski, M., Olejnik, K., Wu, Y., Kuciński, Ł. and Miłoś, P., 2021. Subgoal search for complex reasoning tasks. *Advances in Neural Information Processing Systems*, *34*, pp.624-638.

Definitions

Cost is defined as a number of nodes which at least one were inferenced by a neural network.

Computional budget is defined as a maximum allowed cost. If search doesn’t finish after it with found solution, we say that problem is not solved.

Efficient approach menas that our search is able to achieve „good” solve rate given small computional budget

This solution is great but...

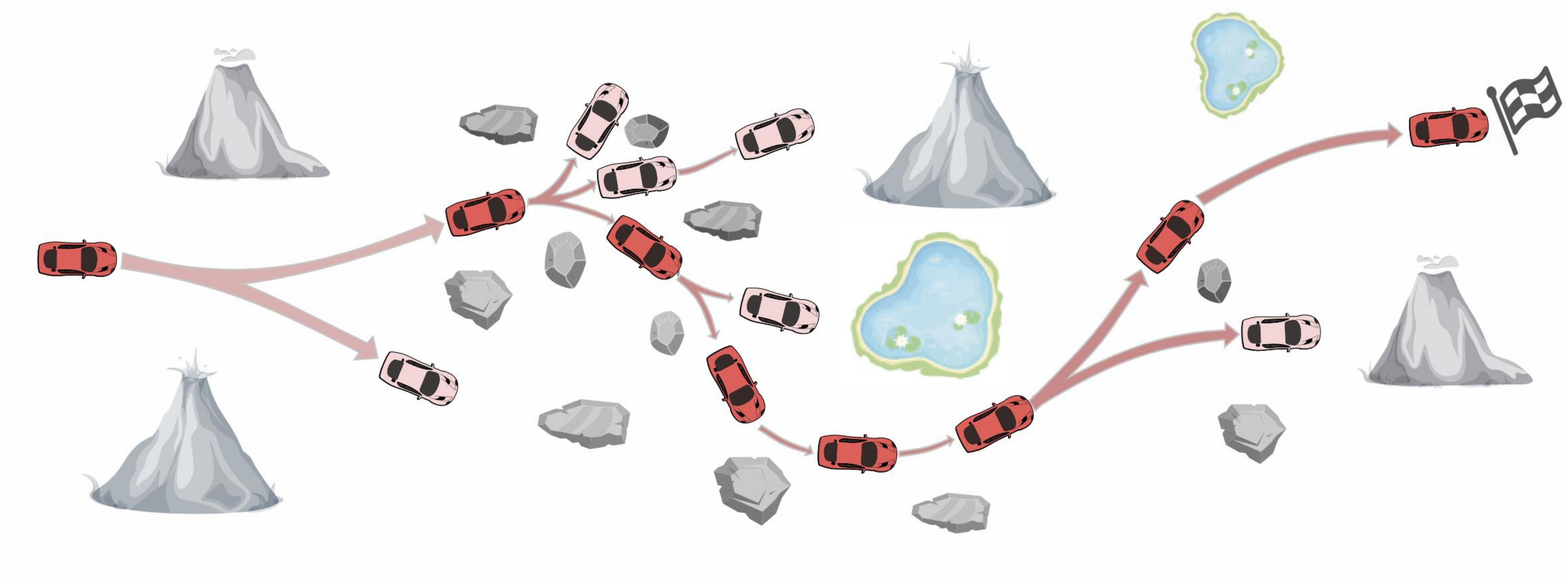
K (distance between subgoals) is fixed

What about involving into action different distances?	What about testing many subgoals with different distance?	What are the approaches for adaptivity?

Filling gaps between subgoals with policy is very **costly**, especially for bigger K

Do we hale always to fill those gaps?

Adaptivity: Motivational example



Adaptive approaches

MixSubS: for which node in search graph, we generate on subgoal of each possible distance.

In each iteration, MixSubS chooses a state with the highest value estimation V (s) to process

Iterative Mixing is similar to MixSubS and allows for advanced schedules of generators to be used. In the consecutive iterations, the i-th generator is used to expand l_i nodes before switching to the next generator. This allows us to flexibly prioritize the better generators, but at the cost of tuning additional hyperparameters l1, . . . , ln. For these reasons, it is not practical, but useful as a reference point.

Strongest-first uses one generator at a time with the longest distance not previously used in state s. In each iteration, Strongest-first chooses a state with the highest value estimation V (s) to process.

Longest-first prioritizes long subgoals over the whole search procedure. Only if the queue does not contain any nodes with the higher k, it uses subgoals of lower distances. The nodes are processed in the order of their value estimation V (s).

Filling gaps

Filling gaps between subgoals is costly.

The longest distance to subgoal, the quality of Subgoal Generator gets much worse

Weak Subgoal Generator is able to generate many invalid subgoals

Running policy for filling those gaps has no sense – we would like to avoid it

Verifier

- Verifier is a neural network
- Verifier:: State x State → [0, 1]
- Verifier(start_state, proposed_subgoal_state) predicts if policy is able to reach subgoal (proposed by subgoal generator) strating from start state and leading to subgoal state
- It saves a lot of computations, especially on small budgets

Algorithm 2 Conditional low-level policy

Requires:	C_2	steps limit
	π	conditional low-level policy network
	M	model of the environment

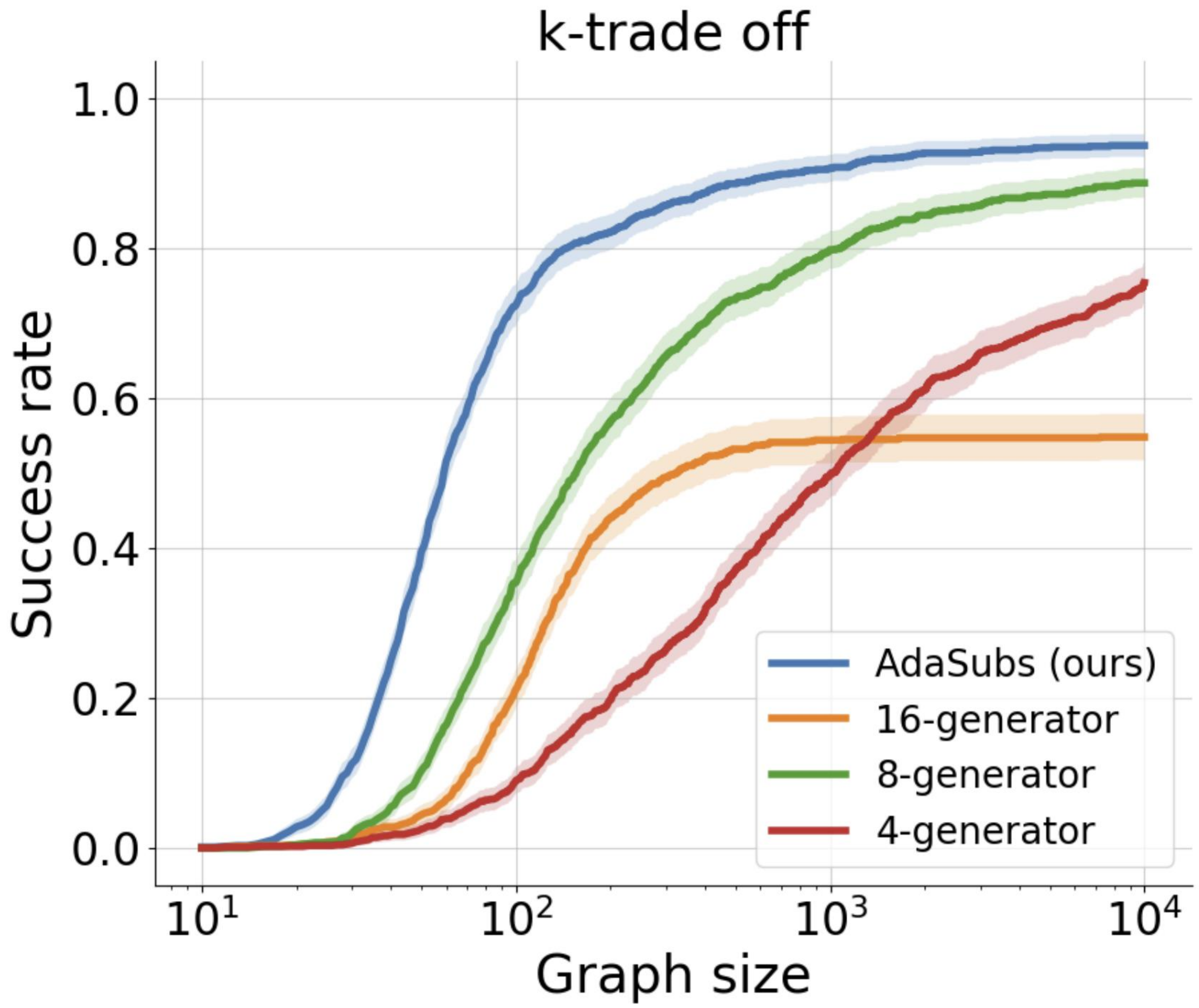
function GET_PATH($\mathtt{s}_0$, $\mathtt{subgoal}$)
  $\mathtt{step} \leftarrow 0$
  $\mathtt{s} \leftarrow \mathtt{s}_0$
  $\mathtt{action\_path} \leftarrow []$
  while $\mathtt{step} < C_2$ **do**
    $\mathtt{action} \leftarrow \pi.\text{PREDICT}(\mathtt{s}, \mathtt{subgoal})$
    $\mathtt{action\_path}.\text{APPEND}(\mathtt{action})$
    $\mathtt{s} \leftarrow M.\text{NEXT\_STATE}(\mathtt{s}, \mathtt{action})$
    if $\mathtt{s} = \mathtt{subgoal}$ **then**
      return $\mathtt{action\_path}$
    $\mathtt{step} \leftarrow \mathtt{step} + 1$
  return $[]$

Algorithm 3 Verification algorithm

Requires:	v	verifier network
	$\mathtt{t_{hi}}$	upper threshold
	$\mathtt{t_{lo}}$	lower threshold

function IS_VALID($\mathtt{s}, \mathtt{s}'$)
  if $v(\mathtt{s}, \mathtt{s}') > \mathtt{t_{hi}}$ **then return** True
  else if $v(\mathtt{s}, \mathtt{s}') < \mathtt{t_{lo}}$ **then return** False
  return $\text{GET\_PATH}(\mathtt{s}, \mathtt{s}') \neq []$

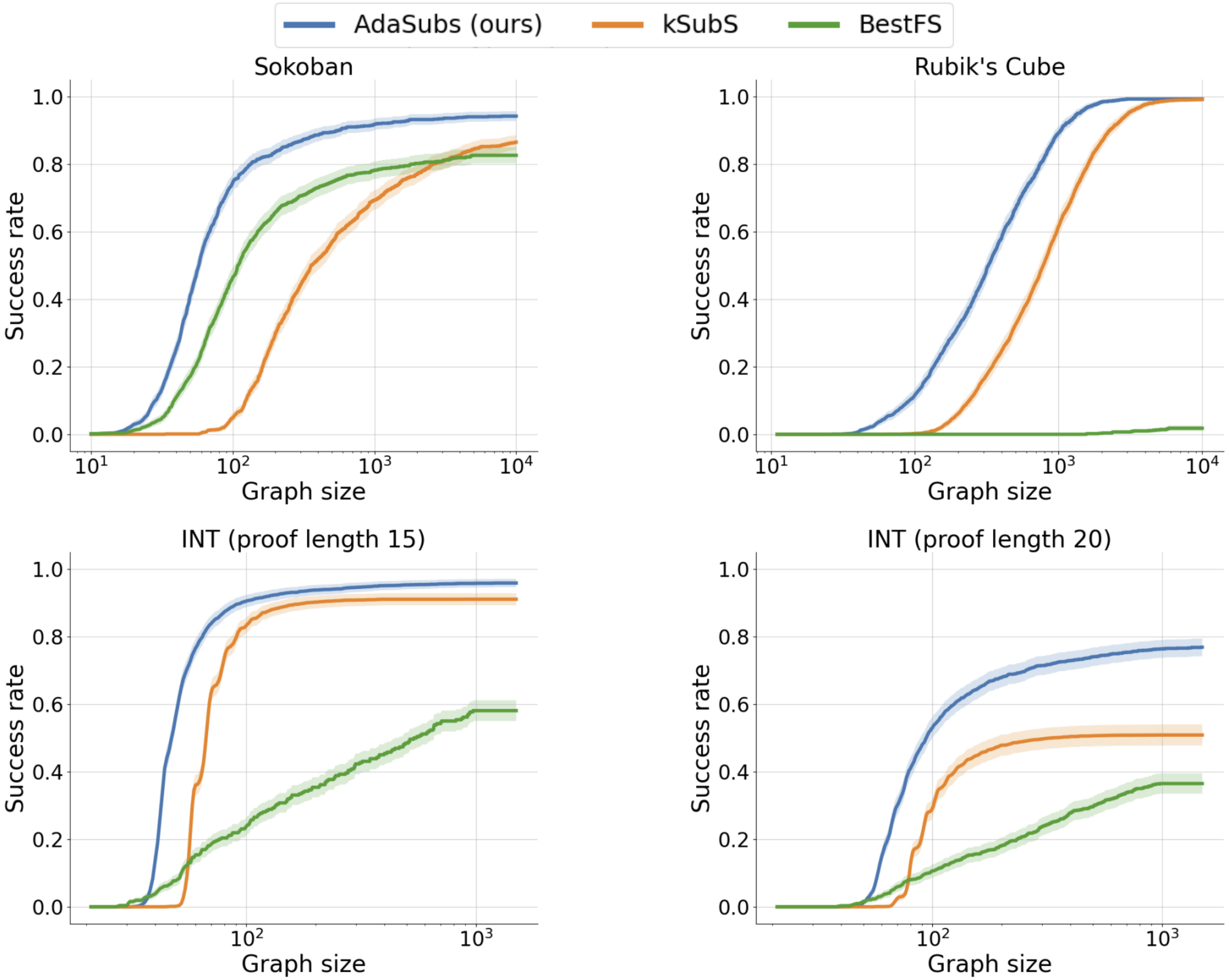
Results



Comparison of success rates for different subgoal generators for Sokoban. AdaSubS-k describes using a single generator with distance k.

Comparison

INT					
		Small budget (50 nodes)		Large budget (1000 nodes)	
		with verifier	without	with verifier	without
BestFS		-	1.7%	-	36.7%
kSubS	$k = 4$	2.2%	0.1%	82.4%	83.0%
	$k = 3$	4.0%	0.2%	89.6%	90.7%
	$k = 2$	2.1%	0.5%	89.8%	91.7%
	$k = 1$	0.0%	0.0%	34.7%	46.0%
MixSubS	$k = [4, 3, 2]$	0.0%	0.0%	94.6%	95.0%
	$k = [3, 2, 1]$	0.0%	0.0%	92.2%	92.9%
	$k = [3, 2]$	17.0%	14.8%	92.2%	93.5%
Iterative mixing	iterations $= [1, 1, 1]$	32.0%	30.1%	87.0%	88.6%
	iterations $= [10, 1, 1]$	43.0%	44.8%	95.1%	96.0%
	iterations $= [4, 2, 1]$	54.0%	52.1%	93.6%	95.5%
Strongest-first		39.5%	40.8%	88.5%	89.8%
Longest-first		59.0%	51.5%	95.7%	95.5%


AdaSubs (ours)
kSubS
BestFS
Sokoban
Rubik's Cube
INT (proof length 15)
INT (proof length 20)
Success rate
Graph size
0.0
0.2
0.4
0.6
0.8
1.0
10^1
10^2
10^3
10^4

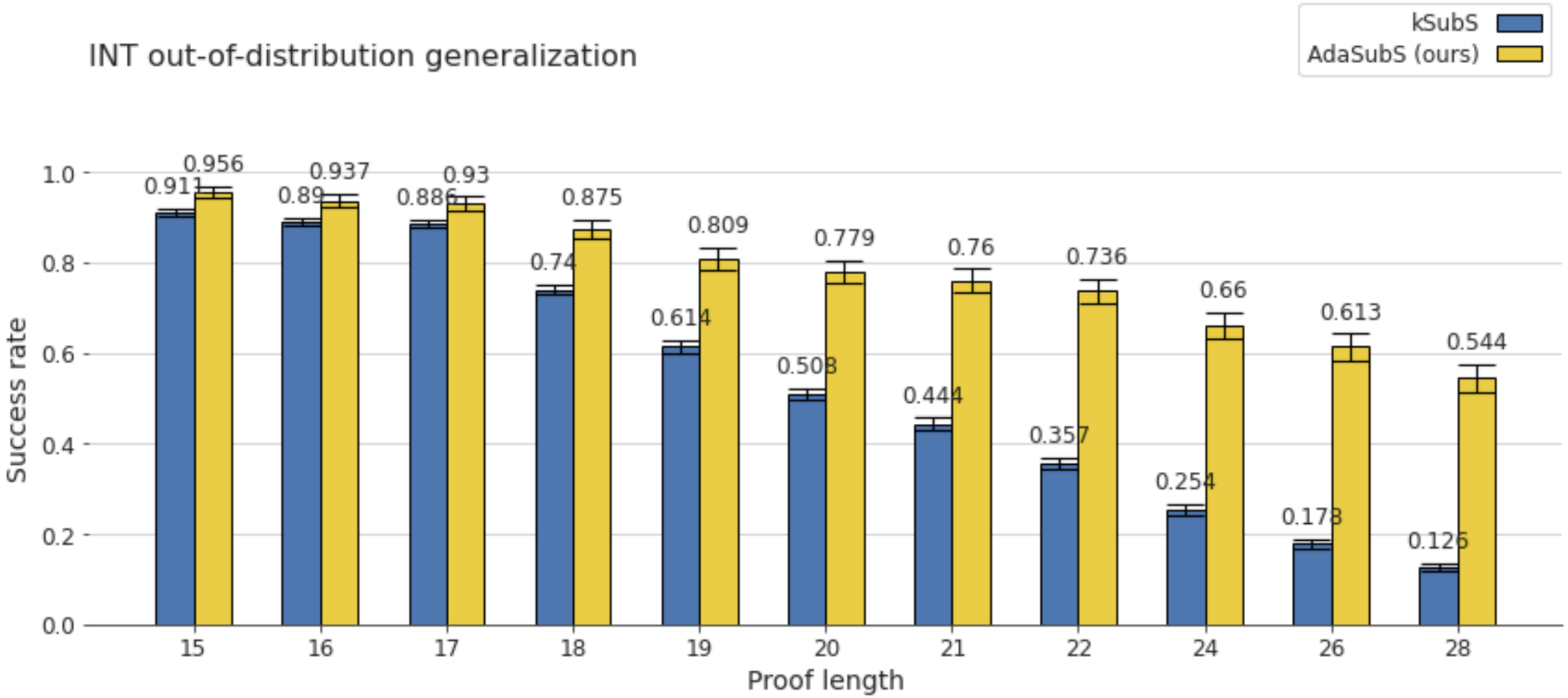


Figure 3: Out-of-distribution performance of AdaSubS and kSubS for long proofs in INT with budget of 5000 nodes. Both methods were trained on proofs of length 15. Error bars correspond to 95% confidence intervals.

Thank you!

michal.tyrolski at gmail.com