Game Solving with Online Fine-Tuning

Ti-Rong Wu,[1]* Hung Guei,[1]* Ting Han Wei,[2] Chung-Chin Shih,[1,3] Jui-Te Chin,[3] I-Chen Wu[3,4]

[1]Institute of Information Science, Academia Sinica, Taiwan
[2]Department of Computing Science, University of Alberta, Canada
[3]Department of Computer Science, National Yang Ming Chiao Tung University, Taiwan
[4]Research Center for Information Technology Innovation, Academia Sinica, Taiwan

`tirongwu@iis.sinica.edu.tw, hguei@iis.sinica.edu.tw, tinghan@ualberta.ca`
`rockmanray.cs02@nycu.edu.tw, pikachin.cs10@nycu.edu.tw, icwu@cs.nctu.edu.tw`

A Implementation details

A.1 PCN training

We basically follow the same PCN training method by Wu et al. [1] but replace the AlphaZero algorithm with the Gumbel AlphaZero algorithm [2], where the simulation count is set to 32[2] in self-play and starts by sampling 16 actions. The architecture of the PCN contains three residual blocks with 256 hidden channels. A total of 400,000 self-play games are generated for the whole training. During optimization, the learning rate is fixed at 0.02, and the batch size is set to 1,024. The PCN is optimized for 500 steps for every 2,000 self-play games. The pre-trained PCN requires around 13 hours to train on a machine with four 1080Ti GPUs, i.e. 52 1080Ti GPU-hours. For the online trainer, we use the same hyperparameters as the pre-trained PCN but only use one GPU.

A.2 7x7 Killall-Go solver

Our solver is built upon the state-of-the-art (SOTA) 7x7 Killall-Go solver [3] except for the following three changes. First, our solver uses PCN as heuristics while the SOTA solver trains a network with *Faster to Life* (FTL) techniques. Both networks aim to provide a faster move for solving, but FTL requires additional (*komi*[3]) settings in solving, so PCN is much easier to use in our solver. Second, we implement the transposition table based on Shih et al. [4]. This greatly reduces the solving time. Finally, we implement a solution for resolving *Graph-History-Interaction* (GHI, i.e. cycles in Go) [5] problems to ensure the correctness of reusing solutions in the transposition table, based on Kishimoto and Müller [6, 7]'s GHI solution.

A.3 Worker design

The worker is itself a Killall-Go solver. It is GPU bound, i.e. it relies on GPUs more than CPUs since the PCN (a neural network) requires intensive GPU computation. Thus, to fully utilize GPU resources, we implement batch GPU inferencing to accelerate PCN evaluations for workers. In practice, we collect 48 workers together in one process with multiple threads. The process runs MCTS selection for each worker independently. Namely, a total of 48 leaf nodes are generated and evaluated by PCN with one GPU at once. The 48 leaf nodes are collected as a batch for batch GPU

---

[*] These authors contributed equally.

[2] The original PCN training used 400 simulation counts in the self-play, requiring much more computing resources than using Gumbel algorithm.

[3] Since Black plays the first stone in the game of Go, White usually earns some extra points called komi for balance.

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

inferencing, with a batch size of 48. This method greatly reduces the solving time when more workers are used. The baseline distributed game solver creates eight processes as workers, each with one GPU, for a total of 384 workers (eight processes with 48 workers). The online fine-tuning solver has the same number of workers for fairness, but uses seven GPUs (one GPU is spared for the online trainer); the configuration is six processes with 55 workers and one process with 54 workers.

B Experiment details

B.1 Setup

All experiments are conducted in three machines, each equipped with two Intel Xeon E5-2678 v3 CPUs, 192G RAM, and four GTX 1080Ti GPUs. We list other hyperparameters in Table 1.

For the memory used in solving, the manager requires 20G RAM for expanding every 1M nodes, and every 48 workers together in one process requires 30G RAM at most. Note that workers use the same amount of memory regardless of problem size. They are limited to 100,000 nodes per job; the job result is “unsolved” if a solution is not obtained within that limit.

Specifically, for BASELINE with 384 workers, solving *KA* used 2,103 seconds, required 3G RAM for the manager and 240G RAM for the workers; solving *KB* used 156,583 seconds, required 170G RAM for the manager and 240G RAM for the workers. However, for BASELINE with only 48 workers, solving *KA* used 12,151 seconds but only required 2G RAM for the manager and 30G RAM for the workers. Overall, the settings can be varied depending on available machines.

Table 1: Hyperparameters used in the baseline and online fine-tuning solvers. All variants of online fine-tuning solvers use the same settings.

		BASELINE	ONLINE
Manager	# GPUs	1	1
	v_{thr}	16.5	16.5
	k for top-k selection	4	4
Worker	# GPUs	8	7
	# workers	384	384
	# node limitation per job	100,000	100,000
Trainer	# GPUs	0	1

B.2 Scalability of the distributed game solver

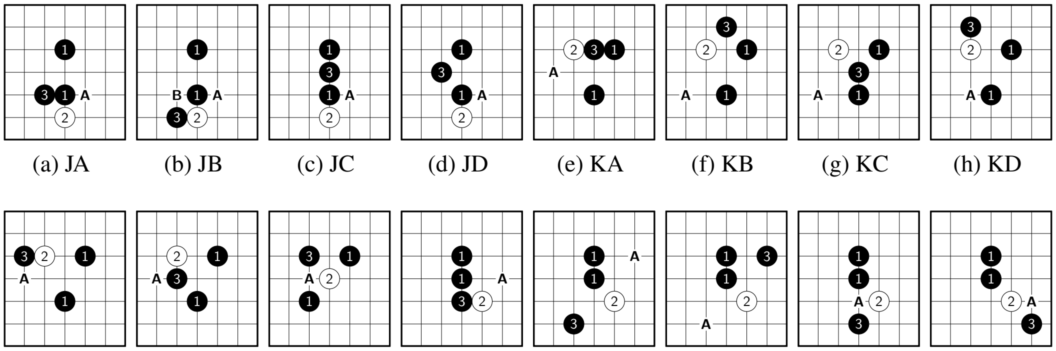
To evaluate the scalability of the distributed game solver, we run BASELINE with different numbers of workers on *KA*. Specifically, the solvers use 384, 192, 96, and 48 workers, using 8, 4, 2, and 1 GPU, respectively. Every 48 workers share one GPU. The results are shown in Table 2. Overall, the speedup is around 1.8 times faster when the number of workers is doubled (up to 384 workers due to our machine limitation).

Table 2: Detailed statistics for solving *KA* by BASELINE with different numbers of workers.

# Workers	# Nodes	Time (s)	Manager # Nodes	# Jobs	Avg. Job Time (s)	Avg. Job # Nodes	# PCN	Solved Jobs (%)	Avg. Worker Loading (%)	Speedup
384	134,881,952	2,103	121,236	21,748	34.48	6,196.46	0	97.87%	94.53%	5.78
192	120,676,465	3,596	99,678	18,598	35.92	6,483.32	0	98.44%	98.57%	3.09
96	112,344,894	6,502	84,752	16,422	37.45	6,835.96	0	98.87%	98.90%	1.71
48	109,362,406	12,151	74,665	15,292	37.78	7,146.73	0	99.05%	98.60%	1.00

B.3 Statistics of solving 7x7 Killall-Go three-move openings

Figure 1 shows the next winning moves (the fourth moves) of 16 three-move openings for both baseline and ONLINE-CP solvers. Generally, both solvers solve the openings at the same next moves, except *JB*. The full solution trees for each opening can be found in this link: `https://rlg.iis.sinica.edu.tw/papers/neurips2023-online-fine-tuning-solver/solution-trees`. We also provide a tool and a README file for explaining the solution tree.


(a) JA (b) JB (c) JC (d) JD (e) KA (f) KB (g) KC (h) KD

(i) KE (j) KF (k) DA (l) SA (m) SB (n) SC (o) SD (p) SE

Figure 1: The solutions of the next winning move for 16 7x7 Killall-Go openings. For each opening, "A" and "B" represents the winning move found by the baseline solver and the online fine-tuning solver respectively. If both solvers solve the opening with the same winning move, only "A" is shown on the board.

It is worth mentioning that *JA* and *JB* are similar to one of the common josekis[4] played in 19x19 Go. The joseki usually occurs when Black makes a *corner enclosure* move, also known as *shimari* in Japanese, like the two stones marked as "1" in *JA* and *JB*. Then, White attempts to invade Black's territories by playing at the stone marked as "2". Judging by the online fine-tuning solver's ability to solve *JA* and *JB*, we foresee a high potential to extend our work to solving other 19x19 Go corner josekis in the future.

In addition, Figure 2 shows the curve for average critical position lengths. These curves are all similar in the sense that it starts with small average lengths, which gradually increases during fine-tuning.

Table 3, Table 4, Table 5 and Table 6 list the experiment results of the baseline and three variants of online fine-tuning solvers respectively, in more detail than those in Table 1 in the main text. These tables include the number of nodes for solving, the solving time in seconds, the number of nodes used in the manager, the number of jobs, the average time for solving each job, the average number of nodes for solving each job, the number of updated PCNs, the success rate of solving jobs, and the average worker load during solving. In general, the solving time is correlated with the number of nodes and the number of jobs. For online fine-tuning, the solving time is also correlated with the number of PCNs as the trainer updates PCNs at a stable speed. Note that the number of PCNs is always 0 for the baseline solver, as they do not update PCNs during solving.

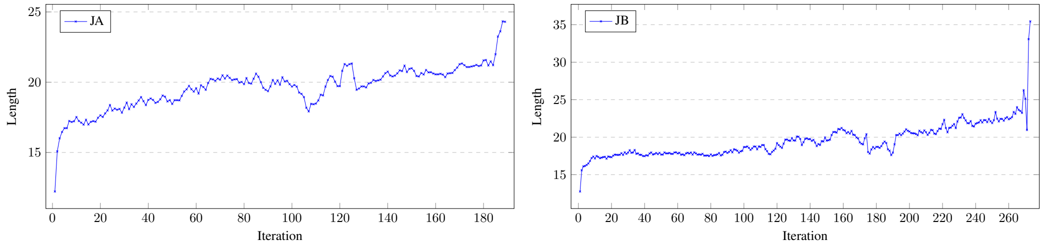
In our experiments, the average success rates of solving jobs are around 97.30%, 98.44%, 99.14% and 99.08% for the baseline and the online fine-tuning solvers, respectively. In addition, for some quickly solved openings, e.g. *KC*, *SA*, and *SB*, the average time for solving each job is far less than other difficult openings. While the workers are able to solve jobs quickly, the managers are relatively unable to create enough jobs for the workers, causing the workers to be relatively idle (lower avg. worker loading). Compared with the baseline solver, online fine-tuning solvers have better success rates of solving as well as lesser nodes for each job. This confirms that online fine-tuning successfully fine-tuned the PCNs for critical positions that the manager is interested in, thereby increasing the job efficiency overall.

B.4 Different PCN thresholds

We examine different v_{thr} from 11.5 to 21.5 on opening *JC*, using the baseline solver. The experiment result is presented in Table 7, where the four columns represent the examined v_{thr}, the total solving time, the average time for workers to solve jobs, and the job success rate. Among these PCN

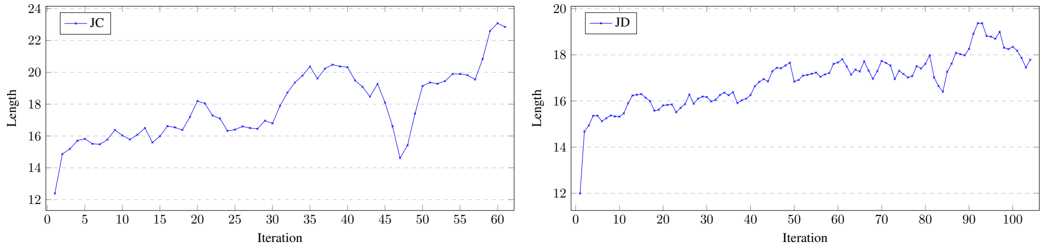
---

[4] A joseki is a move sequence that is widely believed to be balanced play by both players.



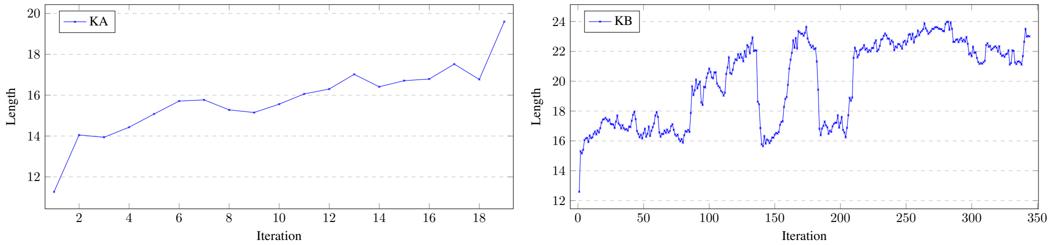
(a) JA

(b) JB



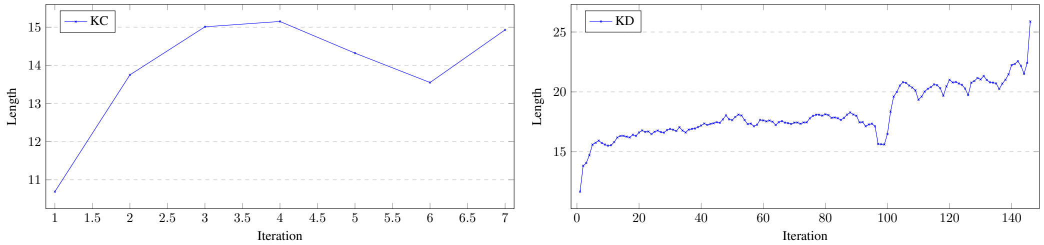
(c) JC

(d) JD



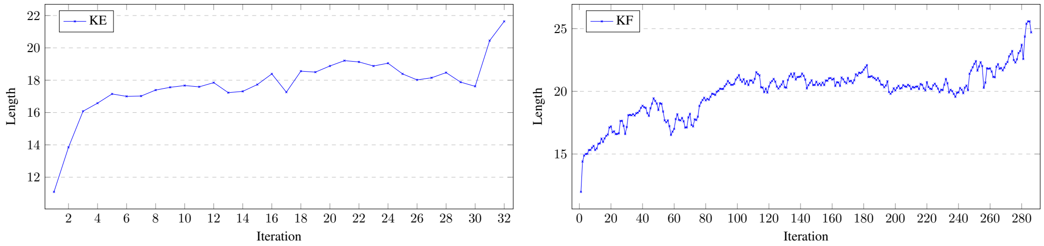
(e) KA

(f) KB



(g) KC

(h) KD

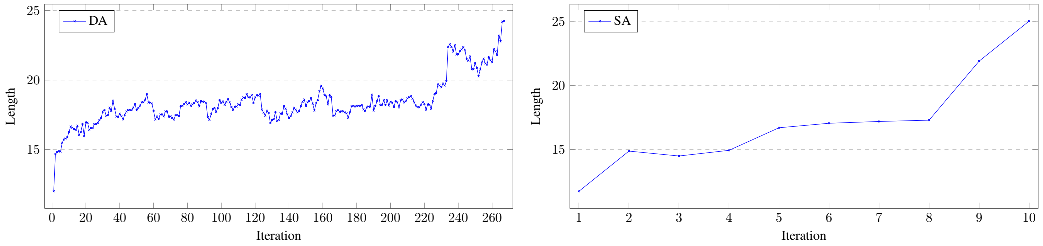


(i) KE

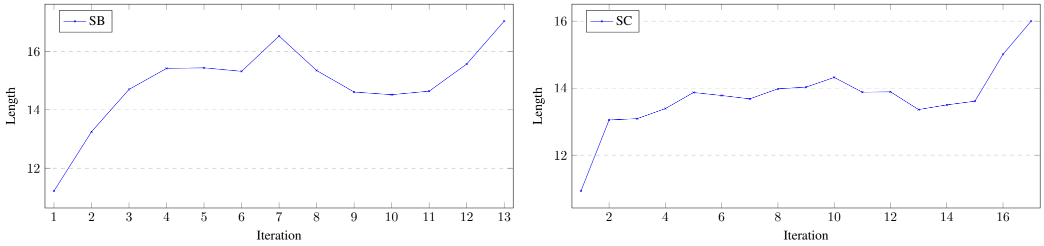
(j) KF

Figure 2: Average length of critical positions for each opening.

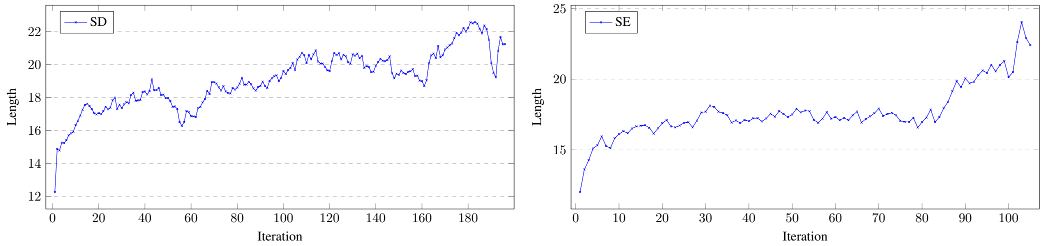
thresholds, we consider $v_{thr} = 16.5$ to be a balanced setting as it performs well in the three metrics. However, the results also show that the performance is not necessarily sensitive to different v_{thr} settings, i.e. the solving time is similar when $v_{thr} \in (15.5, 17.5)$.



(k) DA (l) SA



(m) SB (n) SC



(o) SD (p) SE

Figure 2: Average length of critical positions for each opening.

Table 3: Detailed statistics for the openings solved by BASELINE.

	# Nodes	Time (s)	Manager # Nodes	# Jobs	Avg. Job Time (s)	Avg. Job # Nodes	# PCN	Solved Jobs (%)	Avg. Worker Loading (%)
JA	8,964,444,959	142,115	4,842,554	792,465	68.68	11,305.99	0	96.83%	99.48%
JB	7,137,514,712	155,786	3,689,548	635,263	93.90	11,229.72	0	96.80%	99.48%
JC	721,004,784	12,514	900,221	165,308	23.66	4,356.14	0	98.96%	73.70%
JD	1,271,426,148	30,209	655,078	128,885	89.32	9,859.73	0	97.75%	98.96%
KA	134,881,952	2,103	121,236	21,748	34.48	6,196.46	0	97.87%	94.53%
KB	10,153,035,632	156,583	8,241,207	1,240,258	48.34	8,179.58	0	98.20%	99.48%
KC	38,217,263	747	72,880	15,284	10.33	2,495.71	0	98.39%	62.02%
KD	2,754,213,379	47,494	1,499,735	246,500	73.67	11,167.20	0	97.07%	99.22%
KE	1,197,819,407	18,771	1,024,660	150,490	47.14	7,952.65	0	97.79%	98.44%
KF	9,516,440,320	147,271	5,208,724	789,225	71.50	12,051.36	0	96.88%	99.68%
DA	7,322,743,383	112,874	4,326,195	636,200	67.90	11,503.33	0	95.62%	99.22%
SA	51,272,288	937	79,967	17,772	14.26	2,880.50	0	98.37%	75.78%
SB	215,380,103	3,860	288,751	52,191	22.91	4,121.23	0	97.99%	78.65%
SC	97,559,402	1,557	113,821	22,376	23.31	4,354.92	0	97.94%	90.62%
SD	8,187,017,679	124,644	4,286,025	668,654	71.36	12,237.62	0	95.18%	99.48%
SE	4,297,808,879	64,227	2,234,093	345,124	71.00	12,446.47	0	95.10%	98.86%

As demonstrated in the table, v_{thr} outside of this range deteriorates the solving performance. On the one hand, when v_{thr} is too high, e.g. $v_{thr} = 21.5$, only about 95% of jobs can be solved, implying that about 5% of the jobs are wasted. On the other hand, when v_{thr} is too low, e.g. $v_{thr} = 11.5$, the assigned jobs can be solved quickly with a high success rate. However, this requires the manager to assign more jobs, which increases the overhead of handling job assignments between the manager and

Table 4: Detailed statistics for the openings solved by ONLINE-SP.

	# Nodes	Time (s)	Manager # Nodes	# Jobs	Avg. Job Time (s)	Avg. Job # Nodes	# PCN	Solved Jobs (%)	Avg. Worker Loading (%)
JA	4,054,562,593	69,699	2,491,326	479,455	55.45	8,451.41	359	98.49%	98.95%
JB	3,378,672,517	83,454	1,607,080	327,626	97.49	10,307.68	424	97.27%	99.14%
JC	819,264,890	13,963	704,540	137,752	35.96	5,942.28	57	98.08%	88.34%
JD	846,365,092	19,396	500,259	106,495	69.00	7,942.77	113	98.50%	98.91%
KA	143,814,448	2,621	174,202	34,222	26.17	4,197.31	14	98.55%	90.25%
KB	3,794,290,131	64,493	3,671,889	548,306	43.54	6,913.33	305	98.99%	99.05%
KC	45,217,101	1,156	100,985	21,847	11.12	2,065.09	6	99.47%	54.07%
KD	1,504,977,329	25,715	986,849	202,651	48.37	7,421.58	126	98.86%	98.95%
KE	214,614,577	3,917	246,259	50,422	28.29	4,251.48	21	98.85%	95.48%
KF	6,080,836,868	100,690	5,431,753	855,725	44.93	7,099.72	519	99.19%	99.21%
DA	3,015,438,589	50,046	2,682,998	418,088	45.56	7,206.03	248	98.55%	98.90%
SA	54,574,495	1,471	122,611	25,403	11.78	2,143.52	7	98.47%	56.24%
SB	65,970,358	1,423	124,986	25,917	15.46	2,540.62	7	98.10%	79.84%
SC	213,889,777	3,553	141,447	32,739	39.32	6,528.86	19	98.06%	95.22%
SD	3,821,472,453	63,058	2,224,352	406,191	59.27	9,402.59	329	97.37%	99.01%
SE	2,065,528,927	33,437	1,647,455	282,992	44.79	7,293.07	166	98.21%	98.26%

Table 5: Detailed statistics for the openings solved by ONLINE-CP.

	# Nodes	Time (s)	Manager # Nodes	# Jobs	Avg. Job Time (s)	Avg. Job # Nodes	# PCN	Solved Jobs (%)	Avg. Worker Loading (%)
JA	1,288,601,416	22,384	1,314,785	259,008	32.68	4,970.07	186	99.48%	98.44%
JB	1,576,437,139	31,957	1,643,806	327,442	36.67	4,809.38	272	99.46%	96.66%
JC	316,391,324	6,537	538,369	109,298	17.09	2,889.83	59	99.27%	72.38%
JD	545,655,175	11,083	501,953	109,891	38.12	4,960.85	102	99.32%	98.75%
KA	111,838,889	2,102	159,614	32,153	22.60	3,473.37	18	98.70%	92.69%
KB	2,242,789,149	38,947	3,202,296	575,365	24.45	3,892.46	343	99.65%	93.25%
KC	26,441,989	758	69,052	15,423	9.12	1,709.97	6	99.01%	50.78%
KD	955,257,191	17,434	1,222,772	227,427	26.86	4,194.90	145	99.69%	89.35%
KE	181,418,954	3,336	261,253	49,484	24.25	3,660.93	30	98.86%	94.53%
KF	2,107,185,330	35,418	2,555,399	427,993	31.27	4,917.44	285	99.60%	98.06%
DA	1,761,842,477	30,313	2,556,573	421,268	26.16	4,176.17	266	99.60%	94.03%
SA	41,863,480	992	86,749	19,634	12.60	2,127.77	9	98.57%	70.96%
SB	55,541,455	1,364	125,338	27,612	11.64	2,006.96	12	98.48%	64.32%
SC	98,661,355	1,715	116,980	24,770	24.34	3,978.38	16	98.17%	94.28%
SD	1,395,444,447	23,751	1,575,572	278,198	32.24	5,010.35	195	99.13%	98.33%
SE	757,256,934	12,465	892,615	153,343	30.43	4,932.50	103	99.17%	98.26%

Table 6: Detailed statistics for the openings solved by ONLINE-SP+CP.

	# Nodes	Time (s)	Manager # Nodes	# Jobs	Avg. Job Time (s)	Avg. Job # Nodes	# PCN	Solved Jobs (%)	Avg. Worker Loading (%)
JA	1,425,668,707	24,865	1,370,665	278,183	33.85	5,120.00	225	99.45%	98.18%
JB	1,601,479,130	31,455	1,743,905	351,934	33.16	4,545.55	283	99.50%	95.31%
JC	414,108,746	8,343	693,560	140,608	17.68	2,940.20	69	99.54%	75.26%
JD	502,966,563	10,896	401,057	89,954	45.82	5,586.92	103	99.13%	98.44%
KA	104,905,173	1,931	109,111	23,739	28.97	4,414.51	18	98.21%	95.83%
KB	2,527,488,112	43,200	3,148,579	578,009	27.79	4,367.30	386	99.59%	96.09%
KC	25,508,784	706	58,399	13,172	10.58	1,932.16	6	99.18%	53.65%
KD	920,902,808	16,357	1,092,352	210,345	27.80	4,372.87	148	99.57%	91.15%
KE	168,590,287	3,095	214,173	42,447	26.24	3,966.74	28	98.81%	94.79%
KF	2,027,558,505	35,197	2,203,830	383,317	34.79	5,283.76	305	99.57%	98.33%
DA	1,665,511,033	28,337	2,252,356	377,189	27.77	4,409.62	235	99.53%	95.57%
SA	41,796,555	1,105	95,672	21,325	10.67	1,955.49	10	98.67%	55.73%
SB	109,591,487	2,258	167,238	34,085	19.62	3,210.33	20	98.04%	78.12%
SC	93,535,813	1,655	94,822	21,820	26.28	4,282.36	15	98.12%	93.49%
SD	1,485,439,307	25,531	1,674,274	296,617	32.26	5,002.29	224	99.15%	97.12%
SE	1,200,741,176	20,428	1,289,405	231,498	33.15	5,181.26	182	99.17%	98.01%

the workers, thereby increasing the solving time. Note that the appropriate v_{thr} may vary for different games and for different numbers of available workers. It is possible to adjust v_{thr} dynamically during solving, which is left for future work.

Table 7: The solving time, average job completion time, and success rate of solvable jobs for solving opening *JC* by the baseline solver with different PCN thresholds.

v_{thr}	Time (s)	Avg. Job Time (s)	Solved Jobs (%)
11.5	23,559	2.00	99.92%
12.5	22,870	3.60	99.84%
13.5	18,356	5.75	99.74%
14.5	19,458	12.06	99.55%
15.5	12,519	16.29	99.29%
16.5	12,514	23.66	98.96%
17.5	12,877	33.73	98.34%
18.5	17,536	46.06	97.35%
19.5	22,343	52.04	96.75%
20.5	24,469	58.73	95.99%
21.5	27,810	70.50	94.94%

B.5 Comparison to offline fine-tuning

We now investigate how much benefit we can gain from offline fine-tuning for a specific opening. To do this, we first train θ_0 by generating 400,000 self-play games (around 52 1080Ti GPU-hours) from the empty board. The resulting network is the same as the one referred to as θ_0 in the main text. Next, we fine-tune θ_0 by generating 200,000 additional self-play games (around 26 1080Ti GPU-hours) from the specific opening we are interested in. That is, if we want to solve the opening *JC*, we generate self-play games starting from that opening, and perform updates on θ_0 to obtain what we refer to as $\theta_0^{'}$-*JC*. For this experiment, we used four openings, so the networks $\theta_0^{'}$-*JC*, $\theta_0^{'}$-*KE*, $\theta_0^{'}$-*DA*, and $\theta_0^{'}$-*SE* were produced. Lastly, in the baseline case, we do not update the network with critical positions; the same network is used all throughout the proof search. In ONLINE-CP, critical positions are chosen and the $\theta_0^{'}$ is further fine-tuned using the OFT (resulting in $\theta_1^{'}$, $\theta_2^{'}$, ..., $\theta_t^{'}$, ...).

Table 8: Comparing the impact of a single batch, offline fine-tuning, i.e. pre-training for the specific opening instead of from an empty board.

	w/o offline fine-tuning (θ_0)		w/ offline fine-tuning ($\theta_0^{'}$)	
	BASELINE	ONLINE-CP	BASELINE	ONLINE-CP
JC	12,514	6,537	22,748	10,099
KE	18,771	3,336	2,248	2,417
DA	112,874	30,313	90,298	33,055
SE	64,227	12,465	28,905	42,522

Table 8 shows the times for solving these four openings with and without offline fine-tuning. The left two columns use θ_0 while the right two columns use $\theta_0^{'}$. With offline fine-tuning, the solving times for these openings generally decrease in the baseline solver, since the $\theta_0^{'}$ is specifically fine-tuned for each opening, but exceptions may still occur, as in opening *JC*. However, when using $\theta_0^{'}$, the solving times for ONLINE-CP increase for opening *JC*, *DA*, and *SE*. This may be because $\theta_0^{'}$ only helps learn better heuristics for the opening positions, but does not always guarantee providing accurate heuristics for all varieties of positions during solving. In addition, it is worth noting that although offline fine-tuned $\theta_0^{'}$ accelerates the solving time for the baseline solver, it is impractical since we cannot expect to pre-train $\theta_0^{'}$ for each opening, especially if our eventual goal is to solve complete games from an empty board outright. In contrast, our online fine-tuning solver provides an automatic method that fine-tunes the PCN dynamically without too much extra computation cost.

References

[1] Ti-Rong Wu, Chung-Chin Shih, Ting Han Wei, Meng-Yu Tsai, Wei-Yuan Hsu, and I-Chen Wu. AlphaZero-based Proof Cost Network to Aid Game Solving. In *10th International Conference on Learning Representations, ICLR 2022*, 2022.

[2] Ivo Danihelka, Arthur Guez, Julian Schrittwieser, and David Silver. Policy improvement by planning with Gumbel. In *10th International Conference on Learning Representations, ICLR 2022*, 2022.

[3] Chung-Chin Shih, Ti-Rong Wu, Ting Han Wei, and I-Chen Wu. A Novel Approach to Solving Goal-Achieving Problems for Board Games. In *36th AAAI Conference on Artificial Intelligence, AAAI 2022*, volume 36, pages 10362–10369, 2022.

[4] Chung-Chin Shih, Ting Han Wei, Ti-Rong Wu, and I-Chen Wu. A Local-Pattern Related Look-Up Table. *IEEE Transactions on Games*, 2023.

[5] Andrew J Palay. *Searching with Probabilities*. PhD thesis, Carnegie Mellon University, 1983.

[6] Akihiro Kishimoto and Martin Müller. A General Solution to the Graph History Interaction Problem. In *19th AAAI Conference on Artificial Intelligence, AAAI 2004*, volume 4, pages 644–649, 2004.

[7] Akihiro Kishimoto and Martin Müller. A solution to the GHI problem for depth-first proof-number search. *Information Sciences*, 175(4):296–314, 2005.