

Peer Review

Review of: "Solving Sokoban using Hierarchical Reinforcement Learning with Landmarks"

Evgenii Arsentev¹

1. Independent researcher

1. What the work claims

The paper presents HalfWeg, a hierarchical reinforcement learning scheme in which a planning problem is a tuple (u, v, b) : a start state, a target state, and a flag that asks the agent either to get as close as possible to v ($b = 0$) or as far as possible from it ($b = 1$). Two small ResNet models do all the work. The actions model MA emits d primitive actions; the state model MS, conditioned on recursion depth r , proposes a landmark w that splits the problem from u to v into two halves. A level- i policy PL_i calls MS once, solves the first half with PL_{i-1} , runs the emulator to see where it really landed, then solves the second half from there (§3.3). With $d = 4$ (Supplementary, "Models details") and $R = 5$, the top policy PL_5 produces plans of up to $2^5 \times 4 = 128$ actions. Training is a self-improvement loop (§4): self-play exploration fills a replay buffer, planning instances are sampled from it, every policy proposes a plan via two-leg search with $N_{DSS} = 100$ random intermediate goals (exhaustive search over A^d sequences at level 0), and only the best plan across the hierarchy becomes the supervised target for MA (cross-entropy) and MS (mean squared error).

The empirical claims are three. On Boxoban, PL_5 solves 12.9% of test puzzles with one call and one target (Table 1, row PL_5 , Targets 1, Searches $\rightarrow$), 46% with 36 targets, and 90.2% with 36 targets and 1000 searches (Table 1, last row). An ablation on generated 6x6 Sokoban levels with 3 boxes (§5.2, Figure 3) reports that larger models train better, that $R = 5$ beats $R = 4$ by a modest margin, and that refinement rows help. Finally, the Boxoban-trained PL_5 transfers to generated 10x10 levels with 1 to 9 boxes, solving between 73.3% and 93.8% (Table 2). The headline message is that a six-level goal decomposition can be learned from scratch with under 700K parameters in total.

2. Whether the conclusion holds on the data

I like this paper. The idea is simple to state, it is implemented with honest, small models, and the numbers in Table 1 tell a coherent story: every extra level raises both the solve rate and the plan length. PL_2 to PL_5 at one target go $0.4 \rightarrow 3.7 \rightarrow 8.3 \rightarrow 12.9\%$, with average lengths 14.5, 21.4, 37.0, and 65.6 (each level adds 1.5 to 1.8 times the length; the solve-rate gains shrink from about ninefold at PL_3 to about 1.6-fold at PL_5). I checked these lengths against the caps implied by §3.3 (16, 32, 64, and 128 actions for $d = 4$) and every value sits below its cap, as it should. The parameter counts are also internally consistent: $383,124 + 306,372 = 689,496$ (Supplementary, Boxoban main model), which is exactly the total in the Figure 2 caption, and the 6x6 "Baseline" of §5.2 (700,000 and 680,000) matches the large 6x6 model rounded (696,143 and 679,876). Figure 5, where, as the Supplementary describes it, the two lowest levels plan to move one box at a time, is useful qualitative evidence that the landmarks mean something.

Where I think the claims run ahead of the evidence:

The state-of-the-art comparison is asserted, not shown. The introduction says the agents achieve "performance comparable to state-of-the-art methods". I could not find a single number from another method anywhere in the paper. Related work cites the Boxoban model-free planning work of Guez et al. and the kSubS subgoal search of Czechowski et al., both of which report Sokoban/Boxoban results, yet neither appears in Table 1. Without a side-by-side row, a reader cannot tell whether 90.2% after 36,000 policy calls is strong or weak. Compute matters here too. Policy calls are not a common currency across methods, so I would also report wall-clock time or network evaluations per puzzle.

The search budget behind the best number deserves a plainer description. The 90.2% row uses $36 \text{ targets} \times 1000 \text{ searches} = 36,000 \text{ } PL_5 \text{ calls}$ (the text says this, and the product checks out). Each PL_5 call itself, by the recursion of §3.3, makes $2^5 - 1 = 31 \text{ MS calls}$ and $2^5 = 32 \text{ MA calls}$ (this is my count from the definition, not a number in the paper), plus emulator rollouts. So the best result costs roughly $36,000 \times 63 \approx 2.3 \text{ million network evaluations per puzzle}$. That is fine for a search method. It should be said because the abstract frames the result as "long action sequences from a single high-level call," and the single-call number is 12.9%.

The size of the test set is not stated. The introduction says the work uses the "unfiltered level set" and reports evaluations on the unseen validation and test subsets, and the Table 1 legend says "Boxoban test instances," so I read Table 1 as the unfiltered test subset. Figure 2, however, refers to "validation test instances," and I could not find how many levels Table 1 is computed on. With no N , no confidence

interval, and a single training run, I cannot say whether, for instance, 79.4% vs. 90.2% is separated by more than noise, although with a test set of a thousand levels, it likely would be.

"Targets = 36" needs a definition. Table 1's legend says higher values mean the player is placed "in all possible empty positions." The number of empty cells differs between Boxoban levels, so a fixed 36 is either a cap, a sample, or an average. Which one changes how the 46% row should be read.

Generalization (Table 2) is encouraging, but the solution lengths raise a question. HalfWeg plans are much longer than the generator's: 115.0 against 12.3 for one box (Table 2, row 1), a ratio of 9.3, and 99.98 against 21.9 for four boxes (row 4), a ratio of 4.6. In every row, the HalfWeg length is between 89.4 and 115.0, which is close to the 128-step ceiling of PL₅. I would like to know whether these plans contain long detours or idle moves after the goal is reached; a stop action exists (Supplementary), but it is not clear whether it is used in practice. The non-monotone solve rate is also worth a sentence: one box gives 93.8%, three boxes give 73.3%, nine boxes 79.4%. A one-box 10x10 puzzle failing 6.2% of the time is surprising, and the number of generated levels per row is not given.

The ablations are qualitative. §5.2 says $R = 5$ performs better than $R = 4$ "but not drastically so" and that refinement rows improve training, but the only evidence is Figure 3, which, as rendered on Qeios, I cannot read numerically. Final values with spread across seeds would make these conclusions checkable.

"Without any domain knowledge" is slightly too strong. The method uses the emulator during both training and inference (§3.3, step 3; §4.2), the goal state for evaluation is built by placing boxes on known goal cells (§3.1), and the state encoding uses four hand-chosen channels (player, wall, box, goal). None of this is a flaw. It does place HalfWeg alongside simulator-based planners, which is the right comparison class, and the abstract would be more accurate saying "no demonstrations or hand-designed subgoals."

3. Reproducibility

The Supplementary gives a good start: the input tensor shape (B, C, H, W) with $C = 4$, $d = 4$, the 5-way action space including a stop token, the architecture family (3x3 convolution, ResNet blocks, linear or convolutional head), and exact parameter counts for four model sizes. That is more than many RL papers give.

Several things needed to rerun the work are missing. §5.1 promises that "the detailed architecture and training procedure are presented in the supplementary materials," but the rendered Supplementary contains the architecture only. I could not find the optimizer, learning rate, batch size, number of training

iterations, replay buffer size, number of self-play episodes per iteration, or the length of the random walks in §4.1. No compute is reported (hardware, GPU hours), even though §5.1 cites "computational constraints" as the reason for not scaling the Boxoban model. No random seeds or number of runs. No code repository or trained checkpoints. The custom level generator used in §5.2 and §5.3 is not described, so Table 2 cannot be regenerated.

Two pieces of the method itself are underspecified. First, the distance metric. §3.1 defines the dense reward as "the distance between the target state v and the actual state v' ," and §4.2 ranks plans by it, but I could not find what this distance is (Hamming over cells, per-channel, box positions only, something else). This is the core training signal. Second, MS is trained with mean squared error and outputs a real-valued (B, C, H, W) tensor, yet the lower level must receive a state. How the continuous output is turned into a valid Sokoban state, if at all, is not stated. The target for MS at depth r in §4.3 is the state after $2^{r-1}d$ actions, the midpoint of a level- r plan, which is sensible; a sentence saying so in words would help readers.

A smaller point: at inference with Searches = 100 or 1000, §4.2 says the random intermediate goals come from experience replay. It is not said which buffer is used at test time on unseen levels.

4. *What specifically to fix*

1. **Introduction, last paragraph, and Table 1.** Either add rows for published Boxoban/Sokoban baselines on the same split, with their compute budgets, or remove "comparable to state-of-the-art methods."
2. **§5.1 and the Table 1 caption.** State explicitly that Table 1 uses the unfiltered test subset, say which subset Figure 2 uses, and give the number of test levels. A Wilson interval per row would take one line of code.
3. Define "Targets = 36" in the Table 1 legend: cap, sample, or exact count of empty cells.
4. **§3.1 and §4.2:** give the distance function as an equation. It is the reward, so it belongs in the main text rather than the supplement.
5. In §3.2 or the Supplementary, I would add how the MSE-trained MS output becomes a discrete state before it is passed to PL_{i-1} (argmax per cell, thresholding, or used as a soft target).
6. **Supplementary, "Models details".** Add the training procedure the text promises: optimizer, learning rate, batch size, iterations, buffer size, self-play settings, hardware, and total training time.

7. Report, in the Table 1 discussion, the cost per puzzle of the 90.2% row in network evaluations or seconds, and soften the abstract's single-call framing to match the 12.9% single-call result.
8. **Table 2.** The first column is headed "Boxes %" but holds box counts, so rename it. Add the number of levels per row, and comment on why HalfWeg plans (89.4 to 115.0 steps) are 4 to 9 times longer than the generator's; a post-hoc shortcut-removal pass would also be an easy improvement to report.
9. **§5.2 and Figure 3.** Give final solve rates for each ablation arm in a small table, ideally over three seeds, so that "not drastically so" becomes a number.
10. Abstract: replace "without any domain knowledge" with a statement of what is used (simulator access, goal-cell positions, the 4-channel encoding) and what is not (demonstrations, hand-made subgoals).
11. Release code, the level generator, and the Boxoban checkpoint, and link them in the paper.
12. §4.2, last paragraph: explain briefly why the final policy PL_R is excluded from the collective search.
As written, I had to read it twice.

5. Recommendation

Minor revisions. The method is original and clearly built, the internal numbers I could check are consistent, and the landmarks in Figure 5 are described as interpretable at the two lowest levels. What is missing is mostly reporting: an external baseline, the test-set size, the distance function, and the training details. None of it requires changing the method, and with those additions, the paper would be a solid, citable contribution on deep learned hierarchies for combinatorial planning.

Declarations

Potential competing interests: No potential competing interests to declare.