A. Architectural choices and experimental setup

For the Sokoban domain, we use 10×10 map layout with four boxes and targets. For level generation, we gained access to the level generator described by Weber et al. (2017). We directly provide a symbolic representation of the environment, coded as $10 \times 10 \times 4$ feature map (with one feature map per type of object: wall, agent, box, target), see Fig. 8 for a visual representation.

A.1. Dataset

Vanilla MCTS with a deep value network (see Alg. 1) and long search times (approx. 2000 simulations per step) was employed to generate good quality trajectories, as an oracle surrogate. Specifically, we use a pre-trained value network for leaf evaluation, depth-wise transposition tables to deal with symmetries, and we reuse the relevant search subtree after each real step. Other solving methods could have been used since this is only to generate labels for supervised learning. The training dataset consists of 250000 trajectories of distinct levels; approximately 92% of those levels are solved by the agent; solved levels take on average 60 steps, while unsolved levels are interrupted after 100 steps. We also create a testing set with 2500 trajectories.

A.2. Network architecture details

Our embedding network ϵ is a convolution network with 3 residual blocks. Each residual block is composed of two 64-channel convolution layers with 3x3 kernels applied with stride 1. The residual blocks are preceded by a convolution layer with the same properties, and followed by a convolution layer of 1x1 kernel to 32 channels. A linear layer maps the final convolutional activations into a 1D vector of size 128.

The readout network is a simple MLP with a single hidden layer of size 128. Non-linearities between all layers are ReLus. The policy prior network has a similar architecture to the embedding network, but with 2 residual blocks and 32-channel convolutions.

A.3. Implementation and Training

We implemented MCTSnet in Tensorflow, making extensive use of control flow operations to generate the necessary search logic. Tree node and simulation statistics are stored in arrays of variables and tensors that are accessed with sparse operators. The variables help determine the computational graph for a given execution (by storing the relation between nodes and temporary variables such as rewards). A single forward execution of the network generates the full search which includes multiple simulations. For a fixed expanded tree, gradients can flow through all the node statistics to learn the backup and embedding networks.

We train MCTSnet in an asynchronous distributed fashion. We use a batch of size 1, 32 workers and SGD for optimization with a learning rate of 5e-4.

B. Additional figures

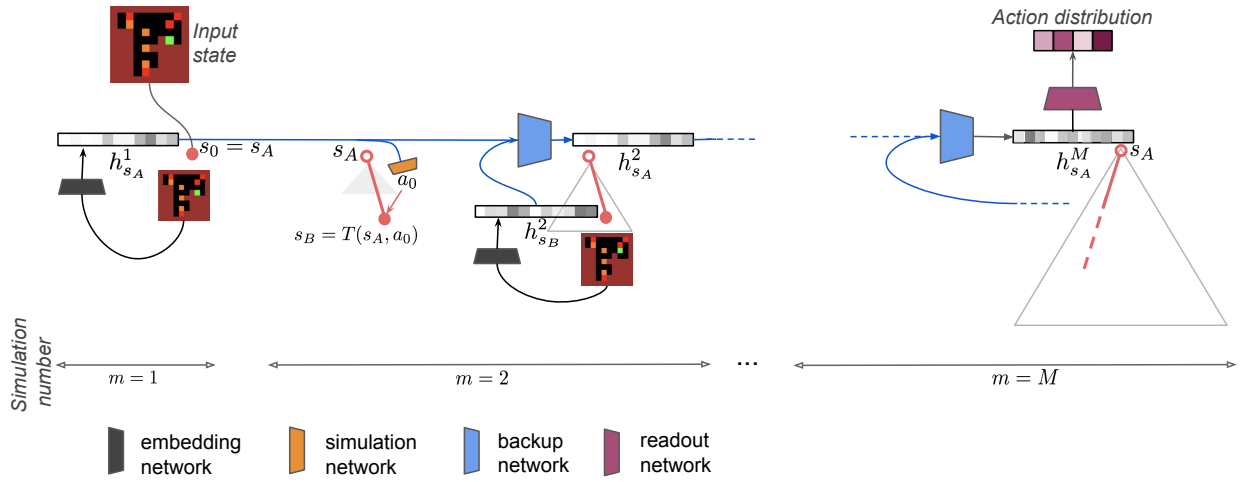

Figure 6. Diagram illustrating a MCTSnet search (i.e., a single forward pass of the network). The first two simulations are detailed, as well as an extract of the last simulation where the readout network is employed to output the action distribution from the root memory statistic. A detail of a longer simulation is given in Figure 7.

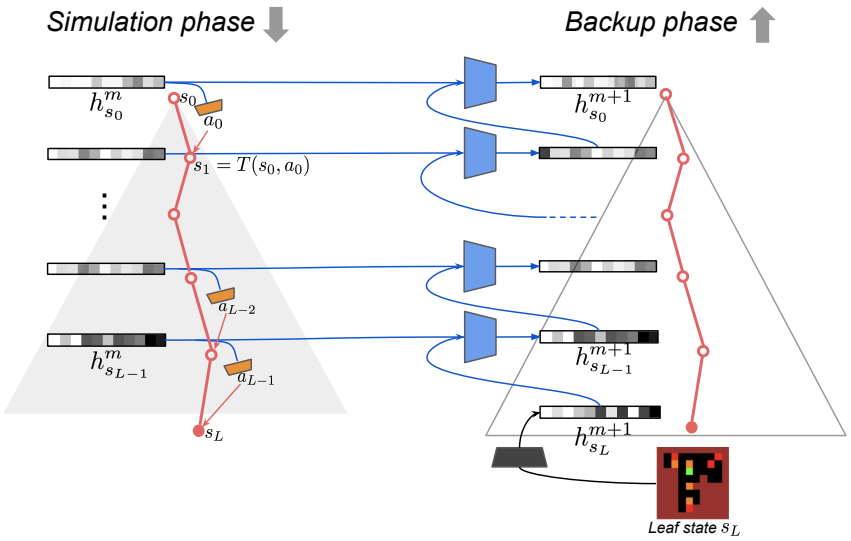

Figure 7. This diagram represents simulation $m + 1$ in MCTSnet (applied to Sokoban). The leftmost part represents the simulation phase down the tree until a leaf node s_L, using the current state of the tree memory at the end of simulation m. The right part of the diagram illustrates the embedding and backup phase, where the memory vectors h on the traversed tree path get updated in a bottom-up fashion. Memory vectors for nodes not visited on this tree path stay constant.

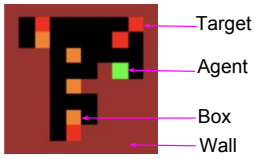

Figure 8. The different elements composing a Sokoban frame. This is represented as four planes of 10x10 to the agent.