A Further Analysis

In this section we broaden understanding of the DirPG update by developing an alternate interpretation of DirPG as the gradient of some other function, which we discovered by reverse-engineering the update. This provides insight into the precise effect of ϵ, provides an interpretation of DirPG as having a built-in control variate, and allows relating the algorithm to other areas of reinforcement learning.

A.1 Reverse Engineering an Objective Function

The final objective that we arrived at via reverse engineering is

$$l(\theta, \epsilon) = \mathbb{E}_{\boldsymbol{S}\sim P}\left[\frac{1}{\epsilon}\log\left(\mathbb{E}_{\boldsymbol{a}\sim\Pi_\theta(\cdot|\boldsymbol{S})}\left[\exp(\epsilon R(\boldsymbol{a},\boldsymbol{S}))\right]\right)\right]. \tag{16}$$

Here we show that differentiating it indeed leads to the DirPG update. To derive the DirPG update, first divide by 1:

$$l(\theta, \epsilon) = \frac{1}{\epsilon}\,\mathbb{E}_{\boldsymbol{S}\sim P}\left[\log\frac{\sum_{\boldsymbol{a}}\exp\left\{\log\Pi_\theta\left(\boldsymbol{a}\mid\boldsymbol{S}\right)+\epsilon R(\boldsymbol{a},\boldsymbol{S})\right\}}{\sum_{\boldsymbol{a}}\exp\left\{\log\Pi_\theta\left(\boldsymbol{a}\mid\boldsymbol{S}\right)\right\}}\right], \tag{17}$$

and then differentiate to get

$$\nabla_\theta l(\theta,\epsilon) = \frac{1}{\epsilon}\mathbb{E}_{\boldsymbol{S}\sim P}\left[\mathbb{E}_{\boldsymbol{a}\sim P_R(\cdot|S)}\left[\nabla_\theta\log\Pi_\theta\left(\boldsymbol{a}\mid\boldsymbol{S}\right)\right]\right] - \frac{1}{\epsilon}\mathbb{E}_{\boldsymbol{S}\sim P}\mathbb{E}_{\boldsymbol{a}\sim\Pi_\theta(\cdot|\boldsymbol{S})}\left[\nabla_\theta\log\Pi_\theta\left(\boldsymbol{a}\mid\boldsymbol{S}\right)\right]. \tag{18}$$

where $P_R(\boldsymbol{a}\mid\boldsymbol{S}) \propto \Pi_\theta\left(\boldsymbol{a}\mid\boldsymbol{S}\right)\exp(\epsilon R(\boldsymbol{a},\boldsymbol{S}))$. Now we can reparameterize the expectations in (18) using Gumbel-max and express the samples in terms of (13):

$$= \frac{1}{\epsilon}\,\mathbb{E}_{\boldsymbol{S}\sim P}\left[\mathbb{E}_{\Gamma}\left[\nabla_\theta\log\Pi_\theta\left(\boldsymbol{a}^*(\epsilon)\mid\boldsymbol{S}\right)\right]\right] - \frac{1}{\epsilon}\,\mathbb{E}_{\boldsymbol{S}\sim P}\left[\mathbb{E}_{\Gamma}\left[\nabla_\theta\log\Pi_\theta\left(\boldsymbol{a}^*(0)\mid\boldsymbol{S}\right)\right]\right]. \tag{19}$$

Having expressed both expectations in terms of Gumbel noise Γ with the same distribution, we can use common random numbers to recover the direct policy gradient:

$$= \frac{1}{\epsilon}\mathbb{E}_{\boldsymbol{S}\sim P,\Gamma}\left[\nabla_\theta\log\Pi_\theta\left(\boldsymbol{a}^*(\epsilon)\mid\boldsymbol{S}\right) - \nabla_\theta\log\Pi_\theta\left(\boldsymbol{a}^*(0)\mid\boldsymbol{S}\right)\right]. \tag{20}$$

The final result is the DirPG gradient, and note that there are no approximate equalities here: (16) is in some sense the underlying objective that DirPG optimizes when ϵ is treated as a hyperparameter.

A.2 Control Variate Interpretation.

The $\mathbb{E}_{\boldsymbol{a}\sim\Pi_\theta(\cdot|\boldsymbol{S})}\left[\nabla_\theta\log\Pi_\theta\left(\boldsymbol{a}\mid\boldsymbol{S}\right)\right]$ term of (18) is the expected value of a score function and thus is identically equal to $\mathbf{0}$. There would be no benefit of including the term in (19). The benefit of including it only becomes apparent in (20), where we can interpret it as a control variate. The optimization problems that define $\boldsymbol{a}_{dir}$ and $\boldsymbol{a}_{opt}$ differ only in value of ϵ, so for small ϵ we expect the solutions to have similar features and correlated score functions. When this is the case, control variates reduce the variance of the overall gradient estimate. To our knowledge, direct optimization has not previously be understood in these terms.

Further experiment on effect of control variate on variance. We measured the variance of DirPG updates with and without the control variate term of Eq. 19 during training on MiniGrid. See Fig. 4. The control variate reduces variance, particularly later in training, when $\boldsymbol{a}_{opt}$ is better correlated with the reward function.

A.3 Risk Sensitivity

A.3.1 Relation to Risk-Sensitive Control

The objective (16) is closely related to a classical objective in risk-sensitive control [33, 14, 9], $\log\mathbb{E}\left[\exp(\epsilon R(\boldsymbol{a},\boldsymbol{S}))\right]/\epsilon$. For $\epsilon > 0$, optimal policies under the classical objective prefer high risk

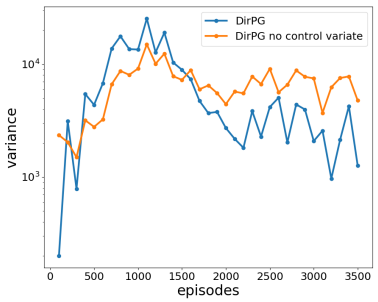

Figure 4: Total empirical variance of the DirPG update as a function of the number of training episodes on MiniGrid.

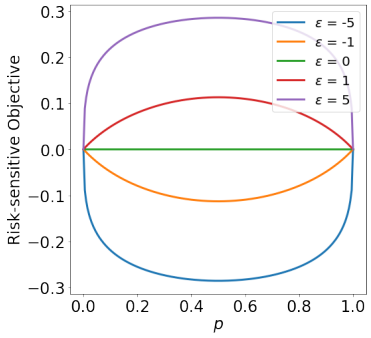

Figure 5: Quadrature evaluation of (16) for the Gaussian choice problem for varying ϵ.

strategies as long as high rewards have some positive probability. For $\epsilon < 0$, optimal policies prefer low risk strategies that avoid placing probability on low rewards. (16) has an important difference. Following [9, 22], we take a Taylor expansion of $\exp(t)$ and $\log(1+t)$ at $t = 0$ to get

$$l(\theta, \epsilon) = \mathbb{E}_{\boldsymbol{S}\sim P, \boldsymbol{a}\sim\Pi_\theta(\cdot|\boldsymbol{S})}[R(\boldsymbol{a}, \boldsymbol{S})] + \frac{\epsilon}{2}\,\mathbb{E}_{\boldsymbol{S}\sim P}[\mathrm{var}_{\boldsymbol{a}\sim\Pi_\theta(\cdot|\boldsymbol{S})}(R(\boldsymbol{a}, \boldsymbol{S}))] + \mathcal{O}(\epsilon^2), \tag{21}$$

where we use the notation $\mathrm{var}_{\boldsymbol{a}\sim\Pi_\theta(\cdot|\boldsymbol{S})}(R(\boldsymbol{a}, \boldsymbol{S}))$ to mean the conditional variance of $R(\boldsymbol{a}, \boldsymbol{S})$ given $\boldsymbol{S}$. Note that expected conditional variance is not equal to the joint variance, which makes this objective different from the typical risk-sensitive analysis. If the second term were simply the variance under the joint, then the agent is sensitive to variance in return regardless of whether it was due to stochasticity in the environment or in the policy. In (21), we see that the agent only seeks out or suppresses "controllable risk," which is variance in return created due to stochasticity in its policy.

Further experiment on "controllable risk." To further illustrate this, we used numerical integration to compute (16) for a simplified "Gaussian choice" setting where an agent chooses to take a reward sampled from $\mathcal{N}(0, 1)$ with probability p and 0 reward with probability $1 - p$. Fig. 5 shows that the risk-seeking objective favors "controllable risk" created due to stochasticity in the agent's policy but not variance created due to stochasticity in the environment.

B Approximate Optimization of $\boldsymbol{a}_{dir}$

Proof of Correctness of Gumbel-approx-max in Deterministic Multi-armed Bandits Suppose we have N arms, each with a fixed but unknown reward $R(i)$ and that arms are ordered according to their reward so $R(i) > R(j)$ iff $i > j$, and $\epsilon > 0$. Let the following:

- $\pi_\theta(i) \propto \exp\theta_i$ be the probability of arm i under a softmax policy parameterized by θ,
- $G_\theta(i) \sim \mathrm{Gumbel}(\theta_i)$

- $D_\theta(i, \epsilon) = G_\theta(i) + \epsilon R(i)$ be the direct objective
- $i_{opt} = \text{argmax}_i \, D_\theta(i, 0)$
- $i_{dir} = \text{argmax}_i \, D_\theta(i, \epsilon)$

Finally, let i_{approx} be the value of i_{direct} arising from running Algorithm 1 using $G_\theta(i)$ as priority. That is, we iterate over i in descending order of $G_\theta(i)$ until we find an i such that $D_\theta(i, \epsilon) > D_\theta(i_{opt}, \epsilon)$ or we have enumerated all i, in which case we set $i_{approx} = i_{opt}$.

We prove that learning using i_{approx} in place of i_{dir} still leads to learning the optimal policy.

Lemma 1. $i_{direct} \geq i_{approx} \geq i_{opt}$.

Proof. To prove $i_{approx} \geq i_{opt}$, observe that by definition we have $D_\theta(i_{approx}, \epsilon) \geq D_\theta(i_{opt}, \epsilon)$ and $G_\theta(i_{opt}) \geq G_\theta(i_{approx})$ This implies

$$G_\theta(i_{approx}) + \epsilon R(i_{approx}) \geq G_\theta(i_{opt}) + \epsilon R(i_{opt}) \tag{22}$$

$$\epsilon R(i_{approx}) - \epsilon R(i_{opt}) \geq G_\theta(i_{opt}) - G_\theta(i_{approx}) \geq 0. \tag{23}$$

Thus $R(i_{approx}) \geq R(i_{opt})$ and $i_{approx} \geq i_{opt}$.

To prove $i_{dir} \geq i_{approx}$ observe that we must have $G_\theta(i_{approx}) \geq G_\theta(i_{dir})$, because otherwise we would have encountered i_{dir} before i_{approx} when iterating i's, and because $D_\theta(i_{dir}, \epsilon) \geq D_\theta(i_{approx}, \epsilon)$ by definition, we would have chosen i_{dir} as i_{approx} when we encountered it.

So we have $G_\theta(i_{approx}) - G_\theta(i_{dir}) \geq 0$, which implies

$$G_\theta(i_{dir}) + \epsilon R(i_{dir}) \geq G_\theta(i_{approx}) + \epsilon R(i_{approx}) \tag{24}$$

$$\epsilon R(i_{dir}) - \epsilon R(i_{approx}) \geq G_\theta(i_{approx}) - G_\theta(i_{dir}) \geq 0 \tag{25}$$

$$\tag{26}$$

Thus $R(i_{dir}) \geq R(i_{approx})$ and $i_{dir} \geq i_{approx}$. □

Lemma 2. *We're at a stationary point iff $i_{direct} = i_{opt}$ (or $i_{approx} = i_{opt}$) almost surely.*

Proof. In one direct, if $i_{direct} = i_{opt}$ almost surely, then DirPG updates on 0 almost surely. In the other direction, suppose for the sake of contradiction that there is some realization of G_θ where i_{direct} is not equal to i_{opt}. By Lemma 1, $i_{direct} > i_{opt}$. Then the gradient vector will have a positive entry for $\theta_{i_{direct}}$ and a negative entry for $\theta_{i_{opt}}$. In order to be at a stationary point, other realizations of G_θ need to cancel these contributions. Because of Lemma 1, however, it is only possible to simultaneously decrement the gradient vector at i and increment it at j if $j > i$. The only way to decrement the previously incremented entry for i_{direct} would be to increment an even larger entry, and the only way to increment the previously decremented entry for i_{opt} would be to decrement an even smaller entry. Thus, there is no way to cancel gradients if any entry is nonzero, and thus the only way to get a zero gradient is if $i_{direct} = i_{opt}$ for all realizations of G_θ. In Lemma 1 we have $i_{direct} \geq i_{approx} \geq i_{opt}$, so the same argument holds for i_{approx}. □

Proposition 1. *The stationary points assuming exact optimization of i_{direct} are the same as the stationary points assuming approximate optimization to get i_{approx}.*

Proof. By Lemma 2, all stationary points assuming exact optimization have $i_{direct} = i_{opt}$ for all realizations of G_θ. By Lemma 1, in each of these realizations we have $i_{direct} \geq i_{approx} \geq i_{opt}$. Thus, for all realizations we have $i_{approx} = i_{opt}$ and thus we are at a stationary point assuming approximate search. In the other direction, Lemma 2 implies that all stationary points assuming approximate optimization have $i_{approx} = i_{opt}$ almost surely. The only way for this to happen is that in trying to find i_{approx} we exhaustively iterated over all arms and found no improvement. Thus, i_{direct} could not have been an improvement and $i_{direct} = i_{opt}$ almost surely. □

C Further Details on $A^\star$ sampling trajectories

Here we provide a more detailed version of Sec. 5, which allows us to more precisely state the limitations of the original $A^\star$ sampling algorithm for RL, and how our algorithm fixes the problems.

Gumbel Processes. To evaluate $D_\theta(\boldsymbol{a}, \epsilon)$, which defines $\boldsymbol{a}_{opt}$ and $\boldsymbol{a}_{dir}$, we need to sample a $G_\theta(\boldsymbol{a})$ value for each complete trajectory encountered during the search. It is not possible to generate $G_\theta(\boldsymbol{a})$ for each $\boldsymbol{a}$ before starting the search, because there may be exponentially (or even infinitely) many possible trajectories. Another option would be to expand the search tree independently of G_θ values and then sample $G_\theta(\boldsymbol{a})$ via (9) for each singleton region encountered during the search. This would produce G_θ values with the right distribution, but it is also a non-starter because we are precisely interested in biasing the search towards trajectories with large G_θ values.

The solution to this problem comes from Maddison et al. Instead of only assigning G_θ values to trajectories, we also assign them to regions. To assign random variables to overlapping regions in a consistent way, Maddison et al. introduce the *Gumbel Process*. A Gumbel process is defined in terms of a sample space Ω and measure μ. In our case, $\Omega = \mathcal{A}^T$ is the set of all length T trajectories and μ assigns probabilities to any subset $\mathcal{R} \subseteq \mathcal{A}^T$ as $\mu(\mathcal{R} \mid \boldsymbol{S}) = \sum_{\boldsymbol{a} \in \mathcal{R}} \Pi_\theta(\boldsymbol{a} \mid \boldsymbol{S})$. A Gumbel Process is then defined as the set $\{G(\mathcal{R}) \mid \mathcal{R} \subseteq \Omega\}$ where the following properties hold:

1. $G(\mathcal{R}) \sim \text{Gumbel}(\log \mu(\mathcal{R}))$,
2. $\mathcal{R}_1 \cap \mathcal{R}_2 = \emptyset \implies G(\mathcal{R}_1) \perp G(\mathcal{R}_2)$,
3. $G(\mathcal{R}_1 \cup \mathcal{R}_2) = \max(G(\mathcal{R}_1), G(\mathcal{R}_2))$.

That is, (1) the G values are marginally distributed as Gumbels with location given by the log measure of the region, (2) the random variables for disjoint regions are independent, and (3) the random variable in the union of two regions is equal to the max of the random variables in the two regions. A fourth property is implied by the first three, which we state in our context:

4. $X(\mathcal{R}) = \text{argmax}_{\boldsymbol{a} \in \mathcal{R}} G(\boldsymbol{a}) \sim 1\{\boldsymbol{a} \in \mathcal{R}\} \Pi_\theta(\boldsymbol{a} \mid \boldsymbol{S})$.

That is, the argmax trajectory $X(\mathcal{R})$ in a region is distributed according to $\Pi_\theta(\cdot \mid \boldsymbol{S})$ that is masked out to only give support to $\mathcal{R}$. Finally, an important property that comes from Gumbel distributions is that $G(\mathcal{R})$ and $X(\mathcal{R})$ are independent random variables [24]. This means that we are free to interleave the sampling of X and G as we please, and it will be leveraged in the algorithms in the following sections.

Top-Down Sampling. Conceptually, if we had sampled $G_\theta(\boldsymbol{a})$ for all $\boldsymbol{a}$, then the rest of the Gumbel process would be determined by $G_\theta(\mathcal{R}) = \max_{\boldsymbol{a} \in \mathcal{R}} G_\theta(\boldsymbol{a})$. However, Maddison et al. show that assuming μ is computable for all regions, a Gumbel Process can be constructed lazily in a "top-down" fashion, first sampling $G(\Omega)$, and then recursively subdividing regions $\mathcal{R}_0$ and sampling G's for the child regions conditional upon the value of $G(\mathcal{R}_0)$. Specifically, they divide $\mathcal{R}_0$ into three disjoint regions: $\mathcal{R}_1, \mathcal{R}_2$, and $\{X(\mathcal{R}_0)\}$ such that $\mathcal{R}_0 = \mathcal{R}_1 \cup \mathcal{R}_2 \cup \{X(\mathcal{R}_0)\}$. They show that for $i \in \{1, 2\}$ the conditional distribution of $G(\mathcal{R}_i)$ given previous splits in the tree is $\text{TruncGumbel}(\log \mu(\mathcal{R}_i), G(\mathcal{R}_0))$ and $G(\{X(\mathcal{R}_0)\}) = G(\mathcal{R}_0)$.

Under our choice of regions, $\mu(\mathcal{R} \mid \boldsymbol{S}) = \sum_{\boldsymbol{a} \in \mathcal{R}} \Pi_\theta(\boldsymbol{a} \mid \boldsymbol{S})$ can indeed be computed efficiently as

$$\Pi_\theta(\mathcal{R}(\tilde{\boldsymbol{a}}, \mathcal{B}; \boldsymbol{S}) \mid \boldsymbol{S}) = \left(\prod_{t'=0}^{t-1} \pi_\theta\left(a_{t'} \mid s_{(a_0, \ldots, a_{t'-1})}\right) \right) \sum_{a \in \mathcal{B}} \pi_\theta(a \mid s_{\tilde{\boldsymbol{a}}}). \tag{27}$$

$\mathcal{B}$ is the set of actions that can be taken after the prefix $\tilde{\boldsymbol{a}}$.

If all prefixes eventually terminate with probability 1, then it is possible to apply one step of Top-Down Sampling to sample trajectories. To split a region $\mathcal{R}_0 = \mathcal{R}(\tilde{\boldsymbol{a}}, \mathcal{B})$, we would sample $X(\mathcal{R}_0) \sim 1\{\boldsymbol{a} \in \mathcal{R}_0\} \pi_\theta(\boldsymbol{a} \mid s_{\tilde{\boldsymbol{a}}})$. This is straightforward because it is essentially conditioning on a prefix in an autoregressive model. Specifically, start with $\tilde{\boldsymbol{a}}$, sample $a_t \sim 1\{a_t \in \mathcal{B}\} \pi_\theta(a_t \mid s_{\tilde{\boldsymbol{a}}})$, and then sample a completion according to

$$\prod_{t'=t+1}^{T} \pi_\theta\left(a_{t'} \mid s_{(a_0, \ldots, a_{t'-1})}\right) \tag{28}$$

However, recursing would be problematic because we do not have a way of splitting $\mathcal{R}_0 \backslash \{X(\mathcal{R}_0)\}$ into two regions that can compactly be represented as a prefix plus legal set of next actions. To

address a similar issue, Kim et al. propose a modified split criteria that divides a region $\mathcal{R}_0$ into two regions. Roughly the idea is to group together $\mathcal{R}_1 \cup \{X(\mathcal{R}_0)\}$ from above into one region, and $\mathcal{R}_2$ as the other region.

Applying the idea to our setting (which is slightly different because we support $|\mathcal{A}| > 2$), to split a region $\mathcal{R}_0 = \mathcal{R}(\tilde{\boldsymbol{a}}, \mathcal{B})$, we assume inductively that we have already sampled $G(\mathcal{R}_0)$ and $X(\mathcal{R}_0)$. Let prefix $\tilde{\boldsymbol{a}}$ have t states and $X(\mathcal{R}_0) = (a_0, \ldots, a_{t-1})$. Note that $X(\mathcal{R}_0) \in \mathcal{R}_0$ by definition, so $\tilde{\boldsymbol{a}}$ is a prefix of $X(\mathcal{R}_0)$ and $a_t \in \mathcal{B}$. We can then define $\mathcal{R}_1 = \mathcal{R}(\tilde{\boldsymbol{a}} \oplus a_t, \mathcal{A})$ and $\mathcal{R}_2 = \mathcal{R}(\tilde{\boldsymbol{a}}, \mathcal{B} \backslash \{a_t\})$. We then need G and X for the new regions. First, $X(\mathcal{R}_0) \in \mathcal{R}_1$, so it must be the case that it continues to be the argmax when considering a smaller region. Thus $\mathcal{R}_1$ "inherits" the parent's max and argmax: $G(\mathcal{R}_1) = G(\mathcal{R}_0)$ and $X(\mathcal{R}_1) = X(\mathcal{R}_0)$. Creating a child region that does not contain the parent argmax follows the same logic as in standard Top-Down sampling: $G(\mathcal{R}_2) \sim \text{TruncGumbel}(\log \mu(\mathcal{R}_2), G(\mathcal{R}_0))$, and we can sample $X(\mathcal{R}_2) \sim 1\{\boldsymbol{a} \in \mathcal{R}_2\}\pi_\theta(\boldsymbol{a} \mid s_{\tilde{\boldsymbol{a}}})$ as described in the previous subsection.

Top-Down Sampling Trajectories. Adapting the search space structure from Kim et al. makes it practical to implement Top-Down sampling for trajectories. However, the algorithm is wasteful in its interactions with the environment, particularly if trajectories can be long, because $X(\mathcal{R})$ is instantiated fully for each region that is put on the queue. This would also prevent applying the algorithm at all if trajectories are of infinite length. We develop a further modification that addresses these issues.

Our idea is to use a similar search space as Kim et al. but to lazily sample $X(\mathcal{R})$. The key observation is that the full value of $X(\mathcal{R})$ is never used when splitting regions. Paired with the fact that maxes and argmaxes are independent, this means that we are free to only maintain prefixes of $X(\mathcal{R})$ and sample extensions when they are needed. Using the same notation as above, we just need samples of the next action a_t to define the split. In fact, we can do away with explicitly maintaining X's in the algorithm altogether. They can be recovered when we encounter a singleton region as the only trajectory in the region. The resulting algorithm is our Modified Top-Down algorithm and appears in Algorithm 2.

D Additional Experimental Details

D.1 Combinatorial Bandits

DirPG interacts with an environment to construct spanning trees as a sequence of binary decisions about whether to include each edge. The environment provides a set of legal actions at each step. If adding an edge would create a cycle, the only legal action is to not add the edge. If there are k steps left and only $n - k - 1$ edges so far, the only legal action is to add the edge. If there is only one legal action, we take it with probability 1. While this reduces the chance of the agent generating an invalid tree, it is possible to generate an invalid spanning tree, in which case we continue searching over trajectories in descending order of $G_\theta(\boldsymbol{a})$ until finding a valid tree. The first valid tree found is returned as the agent's predicted tree. The baseline methods always generate valid spanning trees. Thus, this ensures that the algorithms are not being evaluated in terms of how quickly they learn to generate valid spanning trees. They are all evaluated in terms of how quickly they learn to generate spanning trees with high reward.

As baselines, we use a privileged "semi-bandit" version of UCB that observes per-edge rewards and a version that assumes the per-tree rewards are attributed evenly to the edges, i.e., $r_e = \frac{r_T}{n-1}$. Both baselines choose a tree at time t by computing a maximum spanning tree given upper confidence bound edge costs $u_e = \hat{\mu}_e + \frac{1.5 \log t}{c_e}$ where $\hat{\mu}_e$ is the average per-edge reward for edge e and c_e is the number of times edge e has been chosen.

D.2 DeepSea

The policy model is a linear layer which gets as input one-hot vector of size 5x5 and outputs log probability for each action [FC(number of states, number of actions)]. We used Adam optimizer with a learning rate of 0.001

D.3 Minigrid

The observations are provided as a tensor of shape 7x7x3. Each of the 7×7 tiles is encoded using 3 integer values: one describing the type of object contained in the cell, one describing its color, and a flag indicating whether doors are open or closed. In addition, the agent's orientation is also provided as one-hot vector of size 4.

The policy model consists of 3 convolutional layers and one linear layer on top of them. $Conv1(3, 32) \rightarrow ReLU \rightarrow Conv2(32, 48) \rightarrow ReLU \rightarrow Conv3(48, 64)$. The linear layer gets as input a concatenation of orientation vector and the output of the convolutional layers, namely $FC(64 + 4, 7)$. The output of the linear layer is the log-probabilities of possible action. We used Adam optimizer with a learning rate of 0.001. We used the same architecture for our algorithm and the baselines.

We trained the model for 9M iterations, with a maximum of 3000 iterations per episode. In our algorithm we used the interactions budget for searching for direct candidates. In REINFORCE and cross-entropy method algorithms we used the interactions budget to sample 30 independent trajectories (100 steps trajectories) while we used the simulator to reset the environment. For REINFORCE we averaged the gradients of the 30 trajectories before updating the policy model. For the cross-entropy method we averaged $\nabla_\theta \log \Pi_\theta \left(\boldsymbol{a} \mid \boldsymbol{S} \right)$ over the best 2 out of 30 trajectories. The results shown in Fig. 3 are an average of 5 trials with different random seeds.

We consider two versions of REINFORCE algorithm. The first is the standard trajectory-level $\nabla \mathbb{E}_{\boldsymbol{a},\boldsymbol{s},\boldsymbol{r} \sim p_\theta} \left[\sum_{t=0}^{T-1} r_t \right] = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta \left(a_t \mid s_t \right) \sum_{i=0}^{T-1} r_i$. However, the variance of the trajectory-level is high. The other version is an action-level which consider only the future rewards and serves as a variance reduction technique $\nabla \mathbb{E}_{\boldsymbol{a},\boldsymbol{s},\boldsymbol{r} \sim p_\theta} \left[\sum_{t=0}^{T-1} r_t - b \right] = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta \left(a_t \mid s_t \right) \sum_{i=t}^{T-1} r_i - b$ where the baseline b is the average of the rewards over all time steps.