Solving set multicover problem with reinforcement learning

Mattia Bertè
Department of Computer Science
University of Bologna
mattia.berte@studio.unibo.it

Omar G. Younis
Department of Computer Science
University of Bologna
omargallal.younis@studio.unibo.it

Abstract

This project contains an analysis of the soft actor-critic algorithm applied in the field of decision-focused learning for solving the set multicover problem. We experimented a variation using prioritized experience replay for improving convergence. The algorithm and the environments were mainly derived from the garage library and can be found on GitHub [1].

1 Decision Focused Learning

Decision-making algorithms usually require predictive models and a combinatorial optimization phase, however, these two tasks are often tackled independently following the so-called "Predict and Optimize" paradigm which consists in training a machine learning model to maximize the prediction accuracy, and then using the prediction as input for the optimization phase. However, the loss function used to train the model may easily be misaligned with the end goal, which is to make the best decisions possible.
Decision-focused learning framework (1) integrates prediction and optimization into a single end-to-end system whose predictive model is trained in such a way its predictions are optimal for the decision algorithm.
The general optimization problem can be expressed as:

$$\underset{\omega}{\operatorname{argmin}} \left\{ \sum_{i=1}^{m} c(z^*(y_i), \hat{y}_i) \mid y = f(\hat{x}, \omega) \right\} \tag{1}$$

where:

- $\hat{x}$ and $\hat{y}$ are input-output pairs drawn from some unknown distribution;
- y the predictions of the model;
- ω the parameters of a machine learning model;
- z the actions selected by the decision algorithm;
- c the cost function.

Even if not explicitly presented, z^* is the solution to an optimization problem (the decision problem), which requires solving an $\operatorname{argmin}/\operatorname{argmax}$ operator which is non-differentiable. To overcome this problem, they proposed to work on a relaxation of the problem. Following this idea is possible to tackle this problem using standard reinforcement learning techniques in which the reward is the negative of the $argmin$ argument in eq. (1).

[1] `https://github.com/TiaBerte/rl-for-dfl`

2 Set Multicover Problem

We define here the set multicover problem.

Definition 2.1 *Let $(X, \mathcal{S})$ be a set system, where X is a finite ground set, $\mathcal{S}$ is a collection of subsets of X, and each element $x \in X$ has a non-negative demand $d(x)$. A set multicover problem requires picking the smallest cardinality sub-collection $\mathcal{S}'$ of $\mathcal{S}$ such that each point is covered by at least $d(x)$ sets from $\mathcal{S}$.*

Some examples of set multicover problems are pharmacies locations, where you want to place pharmacies to cover all the demands of different locations, and project time scheduling where you want to allocate the available work time of employers to different project that requires certain amount of time to complete.

3 Soft Actor Critic

Soft actor-critic (2) is an off-policy actor-critic algorithm based on the maximum entropy framework; the term "soft" is derived from soft Q-learning (3). In this framework, the optimal policy is the one that optimizes at the same time both the expected cumulative reward value and the entropy of the policy. A parameter α is introduced to regulate the importance of the entropy with respect to the reward.

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \sum_{t=0}^{T} \mathbb{E}_{(\mathrm{s}_t,\mathrm{a}_t)\sim\rho_\pi}[\gamma^t(r(\mathrm{s}_t,\mathrm{a}_t) + \alpha\mathcal{H}(\pi(\cdot|\mathrm{s}_t))] \tag{2}$$

Off-policy means that the systems uses data generated from a different policy that the one we are trying to optimize. Indeed, SAC exploits experience which is stored in a replay buffer and from which batches are randomly sampled for improving the policy. After doing a policy update, the data is kept in the replay buffer leading to a mismatch between the policy that generated the data and the current policy.
The system is composed of two soft Q-functions, two target soft Q-functions, and an action network. The action network is defined by a Gaussian whose mean and variance are computed by a neural network with parameter ϕ. The soft Q-functions are defined by two neural networks with parameters θ_1 and θ_2, while the target networks are parametrized with an exponential moving average of the parameters of the soft Q-functions (eq. 3).

$$\bar{\theta_i} \leftarrow (1-\tau)\bar{\theta_i} + \tau\theta_i \tag{3}$$

The presence of two soft Q-function helps to mitigate the effect of positive bias in the policy improvement step and moreover, it helps to speed up the training. However, only the minimum between the two soft Q-value is used for computing the gradients. The soft Q-function networks are trained to minimize eq. 4.

$$J_Q(\theta) = \mathbb{E}_{(\mathrm{s}_t,\mathrm{a}_t)\sim\mathcal{D}}\left[\frac{1}{2}\big(Q_\theta(\mathrm{s}_t,\mathrm{a}_t) - \big(r(\mathrm{s}_t,\mathrm{a}_t) + \gamma\,\mathbb{E}_{\mathrm{a}_{t+1}\sim p}[V_{\bar{\theta}}(\mathrm{s}_{t+1})]\big)\big)^2\right] \tag{4}$$

In the first version of this algorithm was present a value-function approximator network which was then abandoned, since the value function is parametrized through soft Q-function parameters using eq. 5

$$V(s_t) = \mathbb{E}_{\mathrm{a}_t\sim\pi}\left[Q(\mathrm{s}_t,\mathrm{a}_t) - \alpha\, log\, \pi(\mathrm{a}_t|\mathrm{s}_t)\right] \tag{5}$$

Policy network is trained minimizing eq. 6

$$J_\pi(\phi) = \mathbb{E}_{\mathrm{s}_t\sim\mathcal{D}}\left[\mathbb{E}_{\mathrm{a}_t\sim\pi_\phi}[\alpha\, log\, (\pi\phi(\mathrm{a}_t|\mathrm{s}_t)) - Q_\theta(\mathrm{s}_t,\mathrm{a}_t)]\right] \tag{6}$$

For helping the optimization phase, a reparametrization trick (4) is applied.
Choosing the optimal temperature α parameter is non-trivial and it needs to be tuned for each task. Reward and entropy can vary a lot for each task and so also the temperature parameter that establishes the relative importance of the two. Moreover, entropy and rewards vary during training while the

policy improves. In a recent work (5), researchers proposed an automated process for tuning this parameter formulating a maximum entropy reinforcement learning objective, where the entropy is treated as a constraint.

$$\max_{\pi 0:T} \mathbb{E}_{\rho_\pi} \left[\sum_{t=0}^{T} \gamma^t r(\mathrm{s}_t, \mathrm{a}_t) \right] \tag{7}$$

$$s.t. \, \mathbb{E}_{(\mathrm{s}_t, \mathrm{a}_t) \sim \rho_\pi} \left[-log(\pi_t(\mathrm{a}_t|\mathrm{s}_t)) \right] \geq \mathcal{H}, \; \forall t \tag{8}$$

where $\mathcal{H}$ is the desired minimum expected entropy.
Exploiting the duality theorem, it's possible to formulate eq. 9 for obtaining the optimal α.

$$\alpha_t^* = \underset{\alpha_t}{argmin} \, \mathbb{E}_{\mathrm{a}_t \sim \pi_t^*} \left[-\alpha_t log \pi_t^*(\mathrm{a}_t|\mathrm{s}_t; \alpha_t) - \alpha_t \bar{\mathcal{H}} \right] \tag{9}$$

where $\bar{\mathcal{H}}$ is the target entropy.

4 Prioritized Experience Replay

Schaul et al. (6) proposed a new method for sampling the experience from the buffer. They suggested sampling with a higher probability the transitions from which is possible to learn more. A possible approximator of this value is the temporal difference error which in some reinforcement learning algorithms is already computed for updating the network parameters. However this greedy technique presents some issues; for example, it requires to sequentially pass all the replay buffer. To avoid expensive sweeps over the entire replay memory, TD errors are only updated for the transitions that are replayed. One consequence is that transitions that have a low TD error on the first visit may not be replayed for a long time (which means effectively never with a sliding window replay memory). Further, it is sensitive to noise spikes (e.g. when rewards are stochastic), which can be exacerbated by bootstrapping. To overcome this problem, they proposed sampling proportionally to the priority:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_i^\alpha} \tag{10}$$

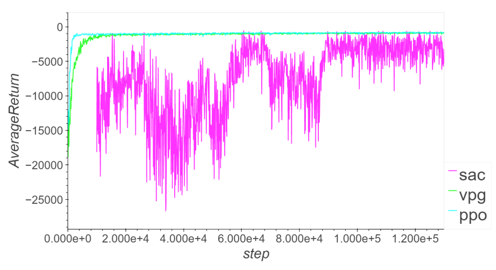
where $p_i = |\delta| + \epsilon$ with δ being TD error and ϵ a small constant to avoid never revisiting states whose error was zero. The exponent α determines how much prioritization is used, with $\alpha = 0$ corresponding to the uniform case.
The prioritized experience introduces bias, indeed prioritizing the transition with an higher error is equivalent to sampling form a distribution different from the one described by the samples present in the buffer and it can lead to a solution different from the desired one. It can be corrected using importance-sampling weights (eq. 11). These weights are then used for rescaling the gradient during the training. For stability reasons, the weights are normalized with respect to the maximum weight. The value β is increased till it reaches 1 toward the end.

$$w_i = \left(\frac{1}{N} \cdot \frac{1}{P_i} \right)^\beta \tag{11}$$

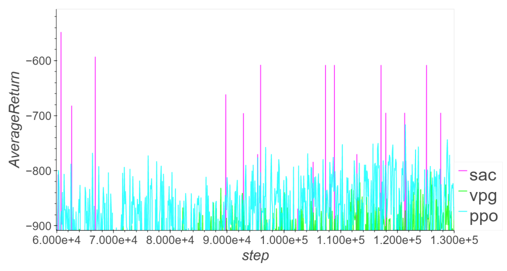
5 Experiments

We started our analysis with a quick comparison between SAC and two on-policy algorithms, PPO (7) and VPG. This round of experiments was conducted using the same backbone for all the algorithms, a 2 fully-connected layers network with 256 neurons for each layer.

From this first trial (Figure 1) we noticed that on-policy algorithms converge faster but they are not capable of achieving the same reward value, so we decided to focus our attention on SAC.
In our successive experiments, we tried to improve the reward and quicken the convergence.
First of all we launched a grid search for identifying the best combination of hidden dimension and batch size, the batch size was chosen in the range $[128, 256, 512, 1024]$, while the hidden dimension was in the range $[128, 256, 512]$.
From Figure 2 we can notice that increasing the number of neurons for the layer helps the convergence, instead there isn't a clear relationship between convergence and batch size since in some cases smaller



(a) Whole plot.



(b) Close-up of Figure 1a.

Figure 1: Reward comparison of VPG, PPO, and SAC algorithms

batches converge faster. However, the best combination is the one with 512 as hidden dimension and 1024 as batch size. Figure 3 is more difficult to read since the convergence at evaluation time is less stable than the one at training time.

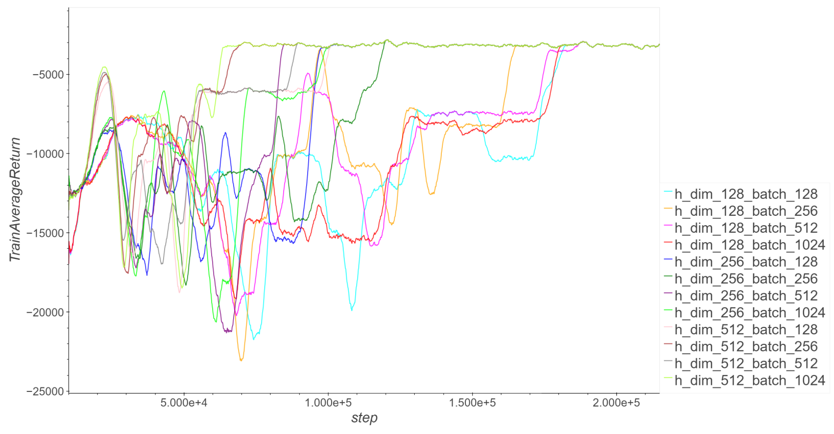


Figure 2: Training reward of SAC during the grid search.

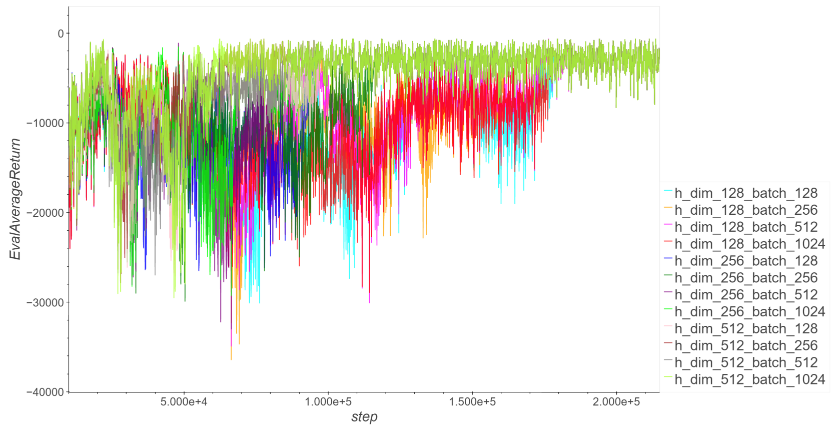


Figure 3: Evaluation reward of SAC during the grid search.

Once the range of possibilities was reduced, we experimented using the prioritized experience replay. We proposed two little variations to the standard technique, instead of using a liner annealing for the β value, we used an exponential one, and instead of assigning the maximum priority to the new samples, we noticed that assigning as priority the mean value of the priorities of the last sampled batch was more robust and improved the performances.

After having fixed the batch size, the hidden dimension, and the type of buffer we launched a

random search to find the best learning rate for both policy network and critic networks and the hyper-parameters related to the prioritized experience replay.
The best configuration we found is presented in table 1. This optimized version not only improved the convergence speed but also the reward value (Figure 4, Figure 5).
In Figure 5 we notice that the SAC curve start after 10000 steps, this is due to the fact that the replay buffer requires a certain number of pre-collected samples before starting the training. We tried to reduce this number but decreasing it showed a drop in performances.
Even if on-policy algorithm seems to converge first, they improve little by little during the whole training so they achieve their best evaluation reward after SAC whose evaluation reward is less stable but whose best results are faster (Table 2).

Networks Hyper-parameters	**Value**
Hidden dimension	512
Batch size	1024
Policy lr	$3 * 10{-}3$
Critic lr	$3 * 10{-}2$

Buffer Hyper-parameters	**Value**
α	0.6
Starting β	0.4
Annealing rate	$3 * 10{-}3$

Table 1: Table with the hyper-parameters of the best configuration.

Algorithm	**Best Reward**	**Steps**
SAC	<u>-509.5</u>	<u>25400</u>
VPG	-885.5	78500
PPO	-767.7	66300

Table 2: Table with the best results for each algorithm.

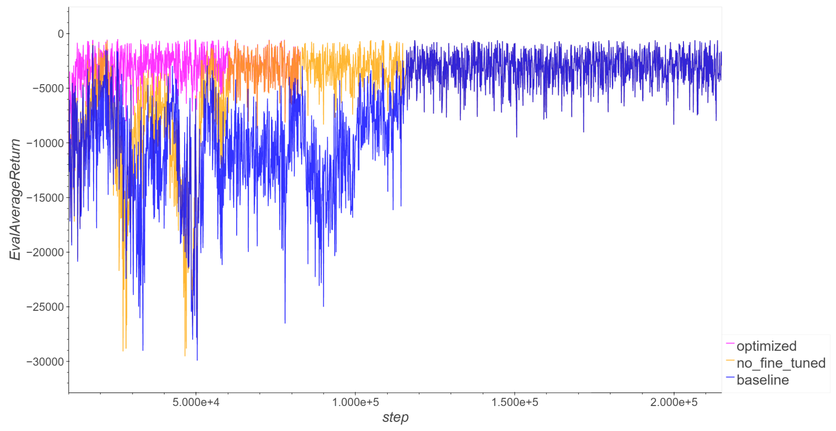


Figure 4: Comparison between best configuration, the baseline and the non-finetuned version.

6 Conclusion

From our project, we can conclude that the decision focused learning framework it's a promising direction for solving decision making problem.
Regarding the analyzed algorithms, it seems that on-policy algorithms converge faster and it's particularly important in this context since faster convergence means being able to provide faster solutions to the decision problem. On the other hand SAC delivers better performances reducing the cost of the decision problem.
After the introduction on the prioritized experience replay technique, the convergence of SAC speeds up, proving the technique to be effective also in this context. Moreover we have to take in account that the first 10000 steps, required for filling the buffer, are just explorative steps and they are quicker

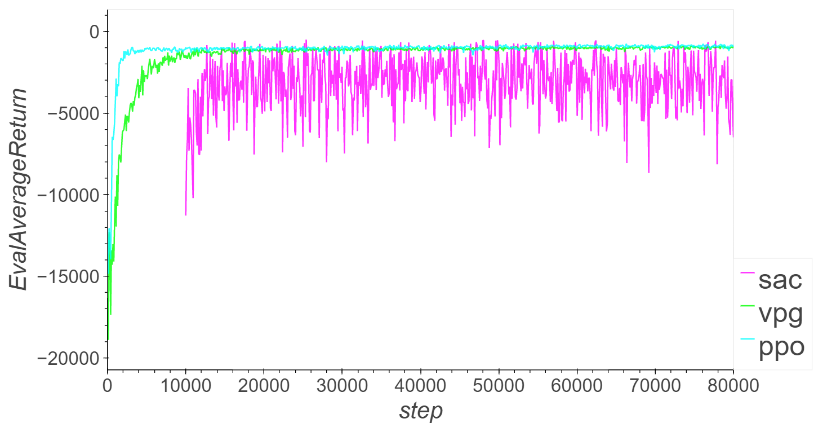


Figure 5: Comparison between VPG, PPO and SAC algorithms.

to execute than the latter ones which also include the policy update phase.
In the end, we can affirm to be satisfied by our project since we achieved better reward, while the increased amount of steps required for the solution seems to be acceptable with respect to the starting phase.
A possible future improvements for this project would be the implementation of Prioritized Sequence Experience Replay (8).

References

[1] B. Wilder, B. N. Dilkina, and M. Tambe, "Melding the data-decisions pipeline: Decision-focused learning for combinatorial optimization," in *AAAI Conference on Artificial Intelligence*, 2018.

[2] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.," in *ICML* (J. G. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 1856–1865, PMLR, 2018.

[3] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine, "Reinforcement learning with deep energy-based policies," in *International Conference on Machine Learning*, 2017.

[4] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[5] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," *ArXiv*, vol. abs/1812.05905, 2018.

[6] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015. cite arxiv:1511.05952Comment: Published at ICLR 2016.

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms.," *CoRR*, vol. abs/1707.06347, 2017.

[8] M. Brittain, J. Bertram, X. Yang, and P. Wei, "Prioritized sequence experience replay," *ArXiv*, vol. abs/1905.12726, 2019.