Deep BOO: Automating Beam Orientation Optimization in Intensity Modulated Radiation Therapy. Part II: Initial Results

Olalekan Ogunmolu*

Department of Radiation Oncology, UT Southwestern Medical Center,
2280 Inwood Road, Dallas, Texas , USA
olalekan.ogunmolu@utsouthwestern.edu

Abstract. This paper is the addendum to the two-part paper on automating the beam orientation optimization in intensity modulated radiotherapy treatment planning systems [1]. The first part presented our methods and formulation using neuro-dynamic programming and monte carlo tree search. In this follow-up paper, we present numerical experiments on the result of training the neural network dynamical system and policy for anonymized prostate cases using select coplanar beams. The results demonstrate the feasibility of our proposal and provides a time-efficient alternative to manually tuning robot gantry angles by dosimetrists in beam orientation optimization scenarios in clinics.

1 Introduction

Beam Orientation Optimization (BOO) is an arduous process in intensity modulated radiotherapy treatment of cancers that affect the human body. Computationally, it involves painstakingly optimizing and tuning a set of weights and parameters in a multi-objective function subject to nonnegeativity constraints of bixel values in each beamlet that is being aimed at a tumor in a target volume. Depending on the geometrical features of the patient's body, and the shape and size of the tumor, the BOO process aims at controlling the intensity of bixels in a radiation profile so as to minimize radiation to critical structures while simultaneously maximizing radiation to malignant cancer tumors.

Searching for the right beam angle combination from which to deliver the most fitting radiation profile for a patient is a large combinatorial search process that is at best solved by an approximation-based algorithm. This is our motivation for treating the beam orientation process as a learning process. As proposed in the Part I of this paper, we proposed using a deep learning approach in an approximate dynamic programming framework in order to find solutions to the BOO problem that generalize across different patients in treatment planning setups. The cornerstone of our design philosophy is to train a neural network multistage decision policy which is capable of predicting good beam orientation candidates on new patient cases that it is tested on. This would save treatment time, improve the IMRT treatment planning process considerably, and improve the patient's clinical experience.

The ensuing results in this paper are initial feasibility results of our approach. The rest of this paper is organized as follows: in section 2, we discuss the end-to-end optimization pipeline for our learing system. We then proceed to summarize our results and findings in 3. We conclude the findings of the paper in section 4.

2 Methods

We cast the decision-making process for what beam angle combination within a gantry's beam phase space could be the "best" for a particular patient as an optimal control problem. Our reasoning is that if one can devise a system model, and pose an appropriate cost function to be optimized, one can expect to recover an extrinsic reward signal; we would expect this signal to adequately capture "the goodness" of a beam candidate set. Formalizing the best beam angle set to be chosen as an inference problem allows us to bring to bear an approximate inference method which provide a model with the capability to reason about partial observability in uncertain high-dimensional state environments.

2.1 The case for monte carlo simulations

Consider the b^d possible move sequences for the state, s_t of a gantry-patient setup where b is the number of defined beam angle moves per time step, and d is the total number of discretized gantry beam angles. If $b = 180$ and $d = 5$, for example, we have 180^5 possible search directions. This renders exhaustive search infeasible. Since the beam space has a high number of branching factors, we leverage on Monte Carlo tree search simulations, encouraged by their recent success in large games e.g. [2–4]. We combine traditional min-max evaluation a tree's leaves with Monte Carlo evaluations by iteratively running random simulations from a tree's root node (containing the beam angles that we start out with) through its sub-nodes; we randomly choose actions until we arrive at a terminal state. A back-up operator progressively averages the outcome of many random simulations to min-max as the amount of simulation expands. For a state s_t other than the root node, s_0, a **tree expansion policy** decides whether to recursively **expand** the current state by executing actions $]\{a_0, a_1, \ldots, a_t\}$ at each time step, t, or if it should roll out the current game to completion by finding the best child of the root node (see Algorithm I of [1]).

During each expansion process (see Algorithm I in [1]), a child node is only added to the tree if it is not already part of the tree; such a child node's reward is computed by finding the optimal fluence objective as defined in [1, §2.4]. Essentially, games of self-play are simulated from the root node, s_0 of the tree, per episode of the Markov Decision Process, and the Monte Carlo value, $Q(s, a)$, of the tree ($i.e.$ the mean outcome of every simulation through state s in which action a was selected) is computed as

$$Q(s, a) = \frac{1}{\mathbb{N}(s, a)} \sum_{i=1}^{n(s)} \mathbb{I}_i(s, a)\zeta_i \tag{1}$$

where $\mathbb{N}(s, a) = \sum_{i=1}^{n(s)} \mathbb{I}_i(s, a)$ is the total number of simulations in which action a was selected in state s; $n(s)$ is the total number of times a game is played through state s, and

ζ_i is the outcome of the ith simulation played out from s. Specifically,

$$\mathbb{I}_i(s,a) = \begin{cases} 1, & \text{if } a \text{ was selected on the } i\text{'th policy rollout} \\ 0, & \text{otherwise.} \end{cases}$$

During simulations, each state and action in the search tree are updated as

$$n(s_t) \leftarrow n(s_t) + 1; \tag{2}$$

$$N(s_t, a_t) \leftarrow N(s_t, a_t) + 1$$

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + r(s_t, a_t) \tag{3}$$

where $r(s_t, a_t)$ is the reward given to the agent after action a is executed while transitioning from state s_t to s_{t+1}. Implementing the MCTS this may makes beam angle transitions evolve in a highly selective, best-first behavior – expanding promising areas of the search space in a deeper way, as long as there is infinite memory and computation.

2.2 Approximate Neuro-Dynamic Programming

The deep neural network, $\boldsymbol{f_\psi}$, (see [1, Fig. 2]) predicts a probability distribution over the m beam angle configurations (we set $m = 5$ in our implementation), $p_a = p(s, a)$ and a *value*, $v_\psi(s)$, which is an estimate of the current beamlets being the optimal beamlet combination in the current tree. Initially, these values are not reliable, but as we play games between the current neural network and any of its previous iterations, we would expect the move predictions from $\boldsymbol{f_\psi}$ to be more accurate. So that the network doesn't memorize each patient's geometry and overfit, we randomly load every other patient data within our dataset and optimize the network end-to-end with it. In our formulation, each player does not make its decision independently but acts in a predetermined order, transmitting its action to an opponent player after it has finished playing. This assures that each player plays its security strategy as an "optimal response" to the action choice of the previous player. To guarantee that the value of player p_1 is similar to that of player, p_2 we calculate the cross-entropy loss, $\boldsymbol{\pi}^T \boldsymbol{p}(s)$ between the final mixed strategies for the two players, based on the tree search policy. This aids in maximizing the similarity of the values predicted by both players. Note that both players' tree search strategy run asynchronously during each episodic setting MDP.

More generally, we apply the **REINFORCE** algorithm of [5] to optimize the weights of either player:

$$\Delta\psi_{ij} = \alpha_{ij} \frac{\partial \log p_\psi(a_t|\boldsymbol{\pi}^\psi(s_t))}{\partial \psi_{ij}} r_t, \tag{4}$$

where ψ are the parameters of the neural network, α is the learning rate, r_t is the reward obtained from the best node using our lookout sparse search simulation, and a_t is the action stored in node, $\boldsymbol{x}$, which was generated by a policy feedback, $\boldsymbol{\pi}$, from state s_t. The gradients of (4) were optimized with Adam (with a 0.01 learning rate) on the accumulated rewards (of the best node returned by MCTS) after finishing each episode of self-play. In general, performing rollouts with the Bellman equation is susceptible

to simulation noise [6]. We thus improve the reliability of the probability estimates by selecting actions that are biased on past neural network experience (weights), $\boldsymbol{f}_{\boldsymbol{\psi}}(\cdot|\psi_{i-1})$ using a variant of the monte carlo tree search algorithm [7, 8, pp. 9]. This helps in selecting moves with a higher intermediate reward.

Our reinforcement learning optimization pipeline can be seen as a form of **policy iteration** algorithm: we generate a policy, $p(s, a)$, in a **policy evaluation** phase; we then improve the policy with MCTS in a **policy improvement** stage. In the policy evaluation phase, we evaluate a suitably chosen cost function, $\bar{Q}$ using neural network $\boldsymbol{f}_{\boldsymbol{\psi}}$, and in the policy improvement step, we improve the new policy, $\boldsymbol{\pi}$. The policy improvement step is tantamount to improving $\boldsymbol{\pi}$ by finding a policy $\bar{\boldsymbol{\pi}}$ that satisfies $T_{\boldsymbol{\pi}}Q_{\boldsymbol{\pi}} = TQ_{\boldsymbol{\pi}}$, where T is a sup-norm contraction mapping, so the iteration converges to an optimal cost-policy pair for any initial conditions [6].

2.3 End-to-End Optimization

The following describes our end-to-end optimization pipeline and ADP approach. We randomize the data that were erstwhile collected on real anonymized prostate CT masks; we use 40 of the 77 Prostate cases as training set, while we use the remainder as test set. Before training, we first preprocess the data as described in [1, §2.2] so that we end up with square shaped slices for each patient's 3D CT.

Each episode of the training scheme consists of randomly sampling from the training dataset, a patient per episode, and concatenating the patient's geometry with the set of five previous beam blocks as outlined in Fig. 2 of [1]. In a first-in, first-out policy, the beam blocks are gradually replaced with the ones that the expansion process of the MCTS algorithm choose. The probability head of the tower residual block produces a probability distribution over all possible discretized beam angles in our defined state space $\mathcal{S}$. We turn this probability prior into a mxing strategy by adding a normalized zero-mean, random walk Gaussian noise with variance 2.0. When we sample from the probability distribution in the **EXPAND** function of Algorithm I (see [1]) , we amend the angle with the move that is recommended and modify the current beam block accordingly. Each node of the tree is formed by using the prior probability computed by the deep neural network as well as the state of the patient (including the current and previous five beam block configuration). When a node is expanded and we transition to other nodes, a full fluence map optimization (FMO) is run in the patient's mask in consideration at that time step alongside the beam angles that the search produces. We take the optimal convex objective of the FMO cost and we use it in scoring the "goodness" of candidate beam blocks that we transition to during the tree traversal.

To aid faster learning, we optimize the deep neural network and search the tree in separate concurrent threads. We write the weights of the network to a shared memory map and make updates to the weights of the network via gradient descent asynchronously while the tree search is runs in parallel. Once we reach a terminal node, new search probabilities are computed, exponentiated by the inverse of the temperature parameter (set to 0.98 in our implementation) as outlined in [1, §2.5]. At inference (test time), we use the last episode of the network in generating search strategies through the beam angle space. The results are presented in the next session.

The value head of the network is optimized to minimize the squared average between the winner of games of self play, ζ and the value predicted by the network, $v_{\psi}^{p}(s)$. Essentially, the mean-square error of the weights are optimized to minimize the partial derivatives of the predicted value with respect to the network weights, ψ as follows:

$$\Delta\psi_{\boldsymbol{v}} = \frac{\partial \boldsymbol{v}_{\psi}}{\partial \psi}(\boldsymbol{\zeta} - \boldsymbol{v}_{\psi}(s)).$$

At the end of each episodic simulation, we compare the value predicted by either player , average their mixing strategies and update the gradients of the loss with respect to the values based on the above equation. Note that for each episodic run, we start from the same random choice of beam angles in order to minimize beam angles collision early on during the search process.
Altogether, we minimize the combined loss,

$$l = (\boldsymbol{\zeta} - \boldsymbol{v})^2 - \boldsymbol{\pi}^T \mathbf{log}(\boldsymbol{p}) + \lambda \|\boldsymbol{\psi}\|_2^2, \tag{5}$$

where λ (set to 1.5) controls regularization of the weights of the network to avoid overfitting. We take the induced tensor norm of the network weights (given its robustness to the assymetrical nature of the weights of each module in the network), and we weight the cross entropy term by 0.9, and the mean square error (mse) loss by 0.01 to keep the overall loss in the direction of persistent reduction in training error. The cross-entropy and mse loss were weighted based on empirical evaluations of the loss surface. To efficiently combine MCTS with deep neural networks, we implement an asynchronous multi-threaded search that executes the fluence map optimization during each expansion phase of the MCTS on CPUs, and optimizes the policy and value networks in parallel on GPUs. We used four concurrent search threads, 8 CPUs, and 4 GPUs for the final version of this algorithm. Similarly, we continually update the opponent, adjusting its weights from previous episodes of the training to allow adequate distribution on the space of beam angles.

Similarly, we train the probability distribution over current beams by maximizing the similarity between the computed search probabilities $\boldsymbol{\pi}$ and the predicted distribution p (by the search process) with the cross-entropy loss:

$$\Delta\psi_p = \frac{\log \partial \boldsymbol{p}_{\psi}(a|s)}{\partial \psi}(\boldsymbol{\pi}^T \mathbf{p})).$$

3 Results and Discussion

In this section, we present the results of our proposal in [1] on 3D prostate cases. The results highlighted are for the central slices in each patient. A snapshot of the training loss surface is given in Fig. 1, demonstrating the steady decline in the cummulative objective function loss for both the value head, prior, and weights of the network (see eq. 5). We provide a visualization of the results in terms of the dose wash plots and dose-cvolume histograms in Tables 1 and 2.

The first results (top half of each of the figures in Tables 1 and 2), are highlights on the dose wash plots and dose-volume histograms for the prostate cases we train the policy

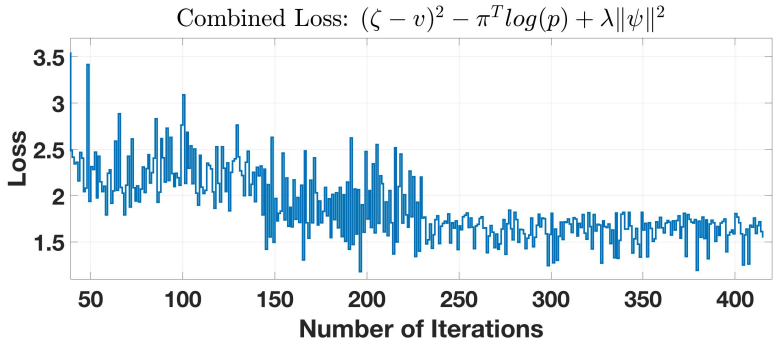


Fig. 1: Combined Loss snapshot for the value head, policy head, and network weights.

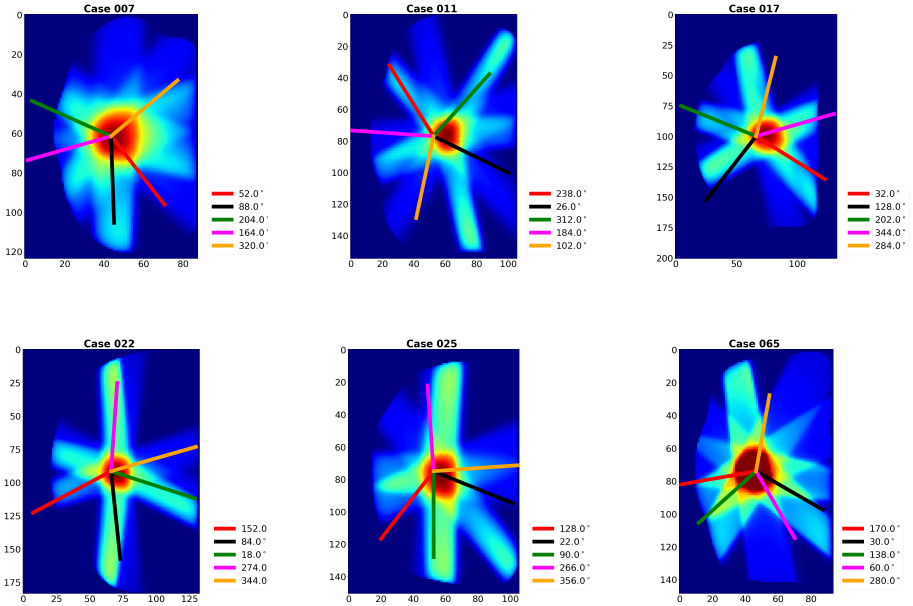
on. The bottom-half of the two tables illustrate the results on test data after training. Note that for all of these cases, the tumor is at the middle of the patient's target volume and this is reflected in the dose washes. We use the fractional DVH curves to evaluate the quality of the plan found by our network oracle. The DVH gives a summary of the distribution of simulated doses within a volume of interest in a 2D format – indicating what volume of a structure in consideration received a particular amount of dose.

From Table 1, we see that the algorithm does select beam angles that are fairly equidistant from one another, thanks to the pairwise distance operator that constrained the selectivity of beamlets in during the search strategy; it generally yields dose wash plots that with good dosimetric concentration on the tumor and sharp gradients at the transition between tumors and OARs; it also largely avoids strong dose to OARs. This is desirable in clinical settings. The dose volume histogram plots in Table 2 shows the quantitative dosimetric efficacy of the resulting beams selected by the deep neural network in terms of the volume of dose delivered to tumor volume and the sparing provided to organs-at-risk volumes. We do acknowledge that there edge cases where a strong dose is given to certain OARs before a sharp drop-off in the dose to the particular organs (e.g. the hot dose to the tumor in case 065). We conjecture that this is due to the network settling on beam blocks with two or more angles that are lfairly close together – hence resulting in this hot dose. For cases such as this, some manual weighting of the relevant terms in the FMO cost could be introduced after running the network to in order to yield a satisfactory wah plot. While the DVH curves do show a consistent amount of dose to each of the tumor in all of the cases for both training and testing data, we notice for example for cases 7, 65 and 62, the high dose to the femoral heads, femoral heads and rectum, and femoral heads and rectum. Examining carefully the wash plots for these cases, we notice fairly close angles in some of the beamlets that were chosen by the deep policy. It becomes clear that mitigating beamlets that are close in angle to one another is an important challenge in taking this proposal from a simulation setting to the real-world. We are currently investigating this and we will send out our findings as soon as we find a solution.

We conjecture that this could that using the optimal FMO cost as a criteria in judging the goodness of beam angle candidates is not adequate enough as a criterion in the search for best beam angles. In a future work, these limitations will be addressed.

Table 1: **Dose wash plots for select patients during training/testing of the self-play network**

Training Regime



Testing Regime

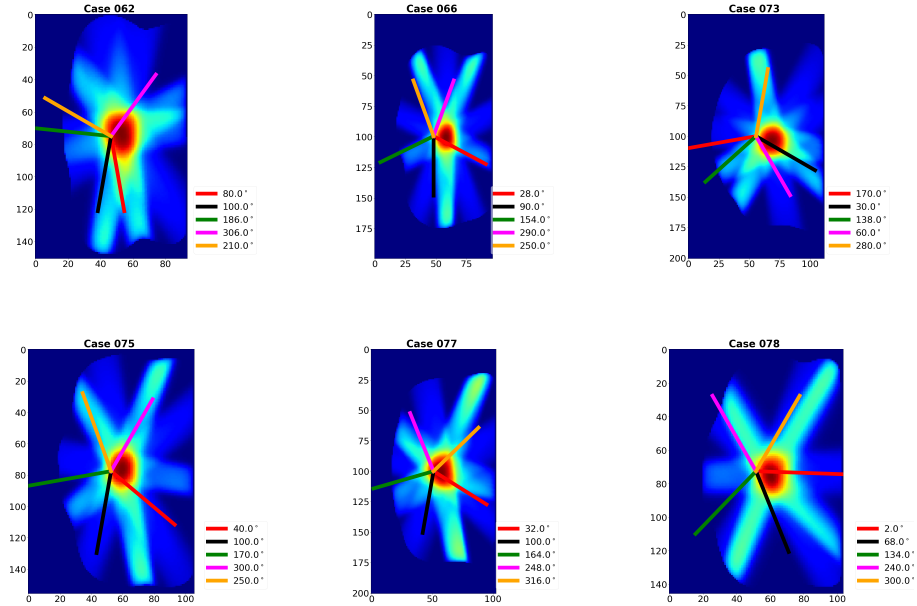
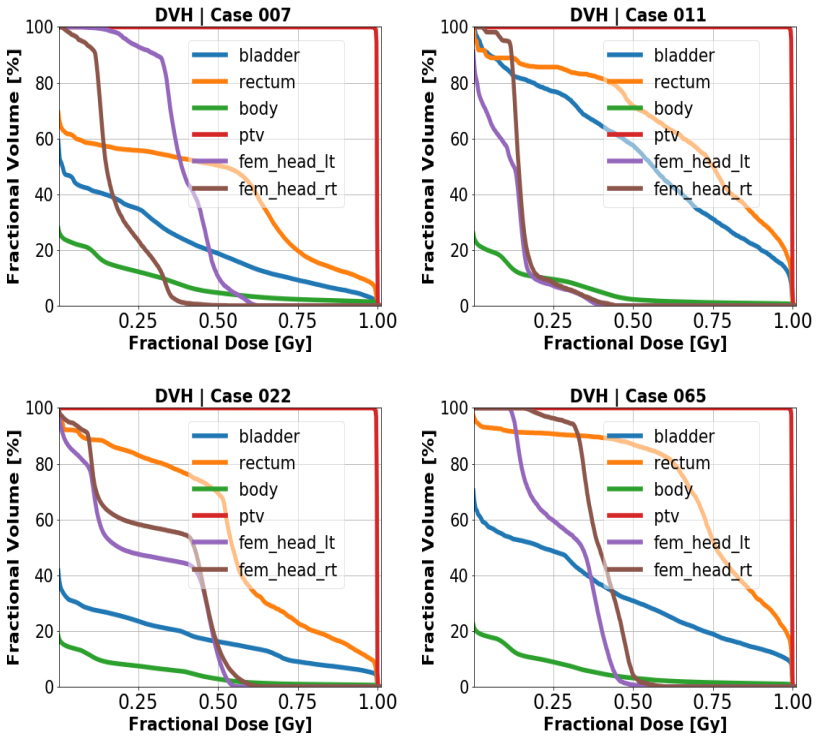
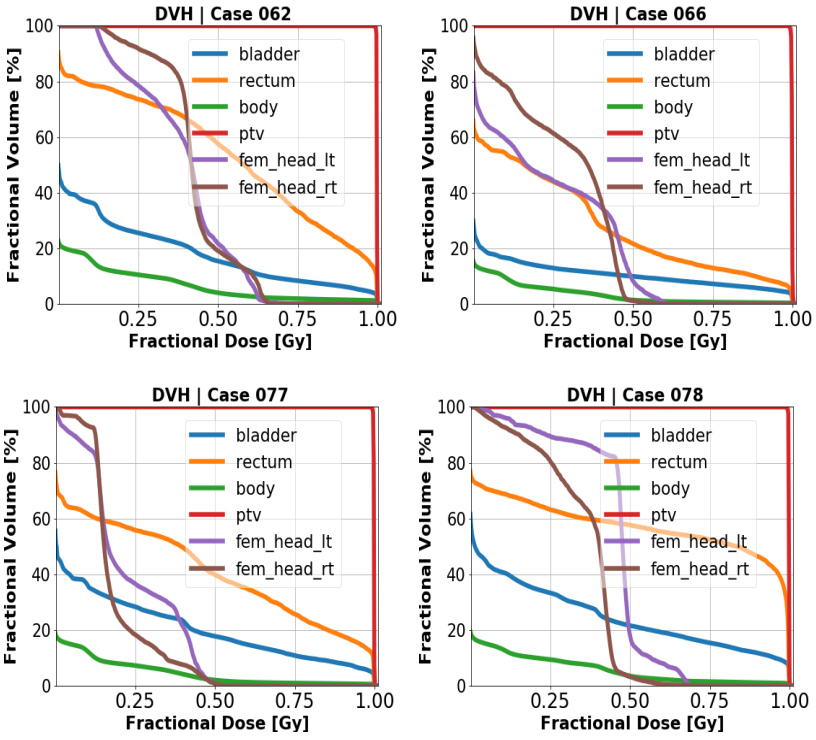


Table 2: **Dose-Volume Histogram For training/testing data**

Training Regime



Testing Regime



However, the advantage of this neural network policy is that finding the beam angles for a simulated TPS is orders of magnitude faster than the current way these angles are found in the clinic. At test time, we pick a saved checkpoint from a previous iteration of training the neural network, we use it to search with the robot's beam-eye-view. The search process tpically takes between 2-3 minutes before we settle on a good candidate beam angle set. This is a lot of time saved compared to manually tuning beam angles by dosimetrists or radiation therapists in clinics – significantly saving time in treatment time incurred in radiation oncology treatment planning systems.

4 Conclusion

Beam orientation optimization is a key component of conformal IMRT radiotherapy TPS. It has a nonconvex solution surface given the way the dose coefficients can significantly alter based on the gantry angle from which beams are aimed to a target volume. As such, in modern clinics, this problem is typically solved with many hours of planning, hand-tuning and refinement – usually by experienced dosimetrists and radiation therapists.

Given the long time traditional optimization approaches take in solving this problem, we have proposed a hybrid approach, leveraging on monte carlo simulation of games in high dimensional state-spaces, neuro-dynamic programming, convex optimization of fluence profiles of a target radiation, and game theory to arrive at good candidate beamlets. Our monte carlo tree search formulation is the first, to the best of our knowledge, that transforms the BOO problem into a monte carlo tree search strategy; and provides a pseudocode that shows how we implement the tree search to navigate the complex state space of respective beamlets. In this paper, we present the results of our feasibility study on prostate cancer cases mostly with a single PTV. We have elucidated on the end-to-end optimization pipeline and provide our initial results based on a set of numerical simulations for select anonymized patients data in our clinic. In the future, we will investigate the lack of spread in chosen beam angles and devise a constraint that ensures that chosen beam angles produce a fluence that is clinically feasible all the time.

References

1. Ogunmolu, O.: Deep BOO: Automating beam orientation optimization in intensity modulated radiation therapy. part i: Methodologies. 13th Workshop on the Algorithmic Foundations of Robotics, Merida, Mexico. (2018) 1, 2, 3, 4, 5
2. Chung, M., Buro, M., Schaeffer, J.: Monte carlo planning in rts games. In: CIG. Citeseer (2005) 2
3. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of Go with deep neural networks and tree search. nature 529, no. 7587: 484-489. (2016) 2
4. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al.: Mastering the Game Of Go Without Human Knowledge. Nature **550**(7676), 354 (2017) 2
5. Williams, R.J.: Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. Machine Learning **8**, 229–256 (1992) 3
6. Bertsekas, D.P.: Approximate Policy Iteration : A Survey and Some New Methods. (2013) 4
7. Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A Survey of Monte Carlo Tree Search Methods. IEEE Transactions on Computational Intelligence and AI in games **4**(1) (2012) 4
8. Gelly, S., Silver, D.: Monte-Carlo tree search and rapid action value estimation in computer Go. Artificial Intelligence **175**, 1856–1875 (2011) 4
9. Okuta, R., Unno, Y., Nishino, D., Hido, S., Loomis, C.: CuPy: A NumPy-Compatible Library for NVIDIA GPU Calculations. In: Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS) (2017)