Neuro-Dynamic Programming for Fractionated Radiotherapy Planning*

Geng Deng[1] and Michael C. Ferris[2]

[1] Department of Mathematics, University of Wisconsin at Madison, 480 Lincoln Dr., Madison, WI 53706, USA, `geng@cs.wisc.edu`
[2] Computer Sciences Department, University of Wisconsin at Madison, 1210 W. Dayton Street, Madison, WI 53706, USA, `ferris@cs.wisc.edu`

Summary. We investigate an on-line planning strategy for the fractionated radiotherapy planning problem, which incorporates the effects of day-to-day patient motion. On-line planning demonstrates significant improvement over off-line strategies in terms of reducing registration error, but it requires extra work in the replanning procedures, such as in the CT scans and the re-computation of a deliverable dose profile. We formulate the problem in a dynamic programming framework and solve it based on the approximate policy iteration techniques of neuro-dynamic programming. In initial limited testing the solutions we obtain outperform existing solutions and offer an improved dose profile for each fraction of the treatment.

1 Introduction

Every year, nearly 500,000 patients in the US are treated with external beam radiation, the most common form of radiation therapy. Before receiving irradiation, the patient is imaged using computed tomography (CT) or magnetic resonance imaging (MRI). The physician contours the tumor and surrounding critical structures on these images and prescribes a dose of radiation to be delivered to the tumor. *Intensity-Modulated Radiotherapy* (IMRT) is one of the most powerful tools to deliver conformal dose to a tumor target [6, 23, 17]. The treatment process involves optimization over specific parameters, such as angle selection and (pencil) beam weights [8, 9, 16, 18]. The organs near the tumor will inevitably receive radiation as well; the physician places constraints on how much radiation each organ should receive. The dose is then delivered by radiotherapy devices, typically in a fractionated regime consisting of five doses per week for a period of 4–9 weeks [10].

* This material is based on research partially supported by the National Science Foundation Grants DMS-0427689 and IIS-0511905 and the Air Force Office of Scientific Research Grant FA9550-04-1-0192

Generally, the use of fractionation is known to increase the probability of controlling the tumor and to decrease damage to normal tissue surrounding the tumor. However, the motion of the patient or the internal organs between treatment sessions can result in failure to deliver adequate radiation to the tumor [14, 21]. We classify the delivery error in the following types:

1. *Registration Error* (see Fig. 1 (a)). Registration error is due to the incorrect positioning of the patient in day-to-day treatment. This is the *interfraction error* we primarily consider in this paper. Accuracy in patient positioning during treatment set-up is a requirement for precise delivery. Traditional positioning techniques include laser alignment to skin markers. Such methods are highly prone to error and in general show a displacement variation of 4–7mm depending on the site treated. Other advanced devices, such as electronic portal imaging systems, can reduce the registration error by comparing real-time digital images to facilitate a time-efficient patient repositioning [17].
2. *Internal Organ Motion Error*, (Fig. 1 (b)). The error is caused by internal motion of organs and tissues of a human body. For example, intracranial tissue shifts up to 1.5 mm when patients change position from prone to supine. The use of implanted radio-opaque markers allow physicians to verify the displacement of organs.
3. *Tumor Shrinkage Error,* (Fig. 1 (c)). This error is due to tumor area shrinkage as the treatment progresses. The originally prescribed dose delivered to target tissue does not reflect the change in tumor area. For example, the tumor can shrink up to 30% in volume within 3 treatments.
4. *Non-rigid Transformation Error,* (Fig. 1 (d)). This type of intrafraction motion error is internally induced by non-rigid deformation of organs, including for example, lung and cardiac motion in normal breathing conditions.

In our model formulation, we consider only the registration error between fractions and neglect the other three types of error. Internal organ motion error occurs during delivery and is therefore categorized as an *intrafraction error*. Our methods are not real-time solution techniques at this stage and hence are not applicable to this setting. Tumor shrinkage error and non-rigid transformation error mainly occur between treatment sessions and are therefore called interfraction errors. However, the changes in the tumor in these cases are not volume preserving and incorporating such effects remains a topic of future research. The principal computational difficulty arises in that setting from the mapping of voxels between two stages.

Off-line planning is currently widespread. It only involves a single planning step and delivers the same amount of dose at each stage. It was suggested in [5, 15, 19] that an optimal inverse plan should incorporate an estimated probability distribution of the patient motion during the treatment. Such distribution of patient geometry can be estimated [7, 12], for example using

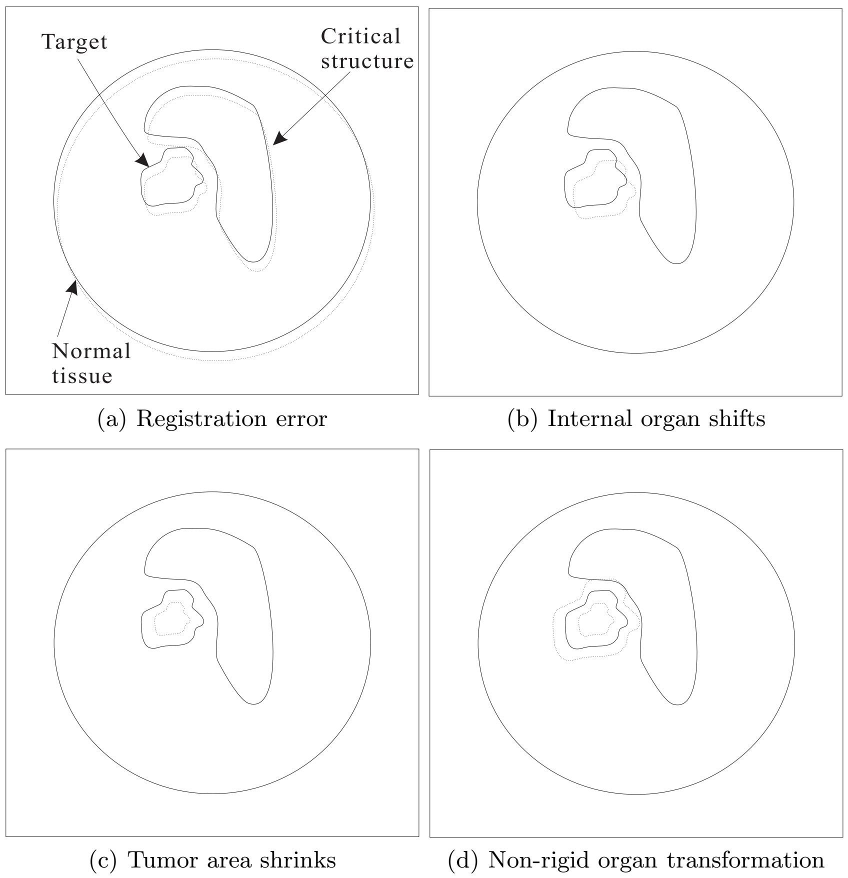


Fig. 1. Four types of delivery error in hypo-fraction treatment

a few pre-scanned images, by techniques such as Bayesian inference [20]. The probability distributions vary among organs and patients.

An alternative delivery scheme is so called on-line planning, which includes multiple planning steps during the treatment. Each planning step uses feedback from images generated during treatment, for example by CT scans. On-line replanning accurately captures the changing requirements for radiation dose at each stage, but it inevitably consumes much more time at every replanning procedure.

This paper aims at formulating a *dynamic programming* (DP) framework that solves the day-to-day on-line planning problem. The optimal policy is selected from several candidate deliverable dose profiles, compensating over time for movement of the patient. The techniques are based on neuro-dynamic programming (NDP) ideas [3]. In the next section, we introduce the model formulation and in Sect. 3, we describe serval types of approximation architecture and the NDP methods we employ. We give computational results on a real patient case in Sect. 4.

2 Model Formulation

To describe the problem more precisely, suppose the treatment lasts N periods (stages), and the state $x_k(i), k = 0, 1, \ldots, N, i \in \mathcal{T}$, contains the actual dose delivered to all voxels after k stages (x_k is obtained through a replanning process). Here $\mathcal{T}$ represents the collection of voxels in the target organ. The state evolves as a discrete-time dynamic system:

$$x_{k+1} = \phi(x_k, u_k, \omega_k),\ k = 0, 1, \ldots, N-1, \tag{1}$$

where u_k is the control (namely dose applied) at the kth stage, and ω_k is a (typically three dimensional) random vector representing the uncertainty of patient positioning. Normally, we assume that ω_k corresponds to a shift transformation to u_k, hence the function ϕ has the explicit form

$$\phi(x_k(i), u_k(i), \omega_k) = x_k(i) + u_k(i + \omega_k),\ \forall i \in \mathcal{T}. \tag{2}$$

Since each treatment is delivered in succession and separately, we also assume the uncertainty vector ω_k are i.i.d. In the context of voxelwise shifts, ω_k is regarded as a discretely distributed random vector. The control u_k is drawn from an applicable control set $U(x_k)$.

Since there is no recourse for dose delivered outside of the target, an instantaneous error (or cost) $g(x_k, x_{k+1}, u_k)$ is incurred when evolving between stage x_k and x_{k+1}. Let the final state x_N represent the total dose delivered on the target during the treatment period. At the end of N stages, a terminal cost $J_N(x_N)$ will be evaluated. Thus, the plan chooses controls $\boldsymbol{u} = \{u_0, u_1, \ldots, u_{N-1}\}$ so as to minimize an expected total cost:

$$\begin{aligned} J_0(x_0) = \min \mathbb{E}&\left[\sum_{k=0}^{N-1} g(x_k, x_{k+1}, u_k) + J_N(x_N)\right] \\ &s.t.\ x_{k+1} = \phi(x_k, u_k, \omega_k), \\ &\qquad u_k \in U(x_k), k = 0, 1, \ldots, N-1. \end{aligned} \tag{3}$$

We use the notation $J_0(x_0)$ to represent an optimal cost-to-go function that accumulates the expected optimal cost starting at stage 0 with the initial state x_0. Moreover, if we extend the definition to a general stage, the cost-to-go function J_j defined at jth stage is expressed in a recursive pattern,

$$\begin{aligned} &J_j(x_j) \\ &= \min \mathbb{E}\left[\sum_{k=j}^{N-1} g(x_k, x_{k+1}, u_k) + J_N(x_N) \middle| \begin{array}{l} x_{k+1} = \phi(x_k, u_k, \omega_k), \\ u_k \in U(x_k), k = j, \ldots, N-1 \end{array}\right] \\ &= \min \mathbb{E}\left[g(x_j, x_{j+1}, u_j) + J_{j+1}(x_{j+1}) \mid x_{j+1} = \phi(x_j, u_j, \omega_j), u_j \in U(x_j)\right]. \end{aligned}$$

For ease of exposition, we assume that the final cost function is a linear combination of the absolute differences between the current dose and the ideal target dose at each voxel, that is

$$J_N(x_N) = \sum_{i \in \mathcal{T}} p(i)|x_N(i) - T(i)|. \tag{4}$$

Here, $T(i), i \in \mathcal{T}$, in voxel i represents the required final dosage on the target, and the vector p weights the importance of hitting the ideal value for each voxel. We typically set $p(i) = 10$, for $i \in \mathcal{T}$, and $p(i) = 1$ elsewhere, in our problem to emphasize the importance of target volume. Other forms of final cost function could be used, such as the sum of least squares error [19].

A key issue to note is that the controls are nonnegative since dose cannot be removed from the patient. The immediate cost g at each stage is the amount of dose delivered outside of the target volume due to the random shift,

$$g(x_k, x_{k+1}, u_k) = \sum_{i+w_k \notin \mathcal{T}} p(i+\omega_k)u_k(i+\omega_k). \tag{5}$$

It is clear that the immediate cost is only associated with the control u_k and the random term ω_k. If there is no displacement error ($\omega_k = 0$), the immediate cost is 0, corresponding to the case of accurate delivery.

The control most commonly used in the clinic is the constant policy, which delivers

$$u_k = T/N$$

at each stage and ignores the errors and uncertainties. (As mentioned in the introduction, when the planner knows the probability distribution, an optimal off-line planning strategy calculates a total dose profile D, which is later divided by N and delivered using the constant policy, so that the expected delivery after N stages is close to T.) We propose an on-line planning strategy that attempts to compensate for the error over the remaining time stages. At each time stage, we divide the residual dose required by the remaining time stages:

$$u_k = \max(0, T - x_k)/(N - k).$$

Since the reactive policy takes into consideration the residual at each time stage, we expect this reactive policy to outperform the constant policy. Note the reactive policy requires knowledge of the cumulative dose x_k and replanning at every stage – a significant additional computation burden over current practice.

We show later in this paper how the constant and reactive heuristic policies perform on several examples. We also show the NDP approach improves upon these results. The NDP makes decisions on several candidate policies (so called modified reactive policies), which account for a variation of intensities on the reactive policy. At each stage, given an amplifying parameter a on the overall intensity level, the policy delivers

$$u_k = a \cdot \max(0, T - x_k)/(N - k).$$

We will show that the amplifying range of $a > 1$ is preferable to $a = 1$, which is equivalent to the standard reactive policy. The parameter a should be confined with an upper bound, so that the total delivery does not exceed the tolerance level of normal tissue.

Note that we assume these idealized policies u_k (the constant, reactive and modified reactive policies) are valid and deliverable in our model. However, in practice they are not because u_k has to be a combination of dose profiles of beamlets fired from a gantry. In Voelker's thesis [22], some techniques to approximate u_k are provided. Furthermore, as delivering devices and planning tools become more sophisticated such policies will become attainable.

So far, the fractionation problem is formulated in a *finite horizon*[3] dynamic programming framework [1, 4, 13]. Numerous techniques for such problems can be applied to compute optimal decision policies. But unfortunately, because of the immensity of these state spaces (Bellman's "curse of dimensionality"), the classical dynamic programming algorithm is inapplicable. For instance, in a simple one-dimensional problem with only ten voxels involving 6 time stages, the DP solution times are around one-half hour. To address these complex problems, we design sub-optimal solutions using approximate DP algorithms – *neuro-dynamic programming* [3, 11].

3 Neuro-Dynamic Programming

3.1 Introduction

Neuro-dynamic programming is a class of reinforcement learning methods that approximate the optimal cost-to-go function. Bertsekas and Tsitsiklis [3] coined the term neuro-dynamic programming because it is associated with building and tuning a neural network via simulation results. The idea of an approximate cost function helps NDP avoid the curse of dimensionality and distinguishes the NDP methods from earlier approximation versions of DP methods. Sub-optimal DP solutions are obtained at significantly smaller computational cost.

The central issue we consider is the evaluation and approximation of the reduced optimal cost function J_k in the setting of the radiation fractionation problem – a finite horizon problem with N periods. We will approximate a total of N optimal cost-to-go functions $J_k, k = 0, 1, \ldots, N-1$, by simulation and training of a neural network. We replace the optimal cost $J_k(\cdot)$ with an approximate function $\tilde{J}_k(\cdot, r_k)$ (all of the $\tilde{J}_k(\cdot, r_k)$ have the same parametric form), where r_k is a vector of parameters to be ascertained from a training process. The function $\tilde{J}_k(\cdot, r_k)$ is called a scoring function, and the value $\tilde{J}_k(x, r_k)$ is called the score of state x. We use the optimal control $\hat{u}_k$ that

[3] finite horizon means finite number of stages

solves the minimum problem in the (approximation of the) right-hand side of Bellman's equation defined using

$$\hat{u}_k(x_k) \in \\ \operatorname*{argmin}_{u_k \in U(x_k)} \mathbb{E}[g(x_k, x_{k+1}, u_k) + \tilde{J}_{k+1}(x_{k+1}, r_{k+1}) | \, x_{k+1} = \phi(x_k, u_k, \omega_k)]. \quad (6)$$

The policy set $U(x_k)$ is a finite set, so the best $\hat{u}_k$ is found by the direct comparison of a set of values. In general, the approximate function $\tilde{J}_k(\cdot, r_k)$ has a simple form and is easy to evaluate. Several practical architectures of $\tilde{J}_k(\cdot, r_k)$ are described below.

3.2 Approximation Architectures

Designing and selecting suitable approximation architectures are important issues in NDP. For a given state, several representative features are extracted and serve as input to the approximation architecture. The output is usually a linear combination of features or a transformation via a neural network structure. We propose using the following three types of architecture:

1. *A neural network/multilayer perceptron architecture.* The input state x is encoded into a feature vector f with components $f_l(x), l = 1, 2, \ldots, L$, which represent the essential characteristics of the state. For example, in the fractionation radiotherapy problem, the average dose distribution and standard deviation of dose distribution are two important components of the feature vector associated with the state x, and it is a common practice to add the constant 1 as an additional feature. A concrete example of such a feature vector is given in Sect. 4.1.
   The feature vector is then linearly mapped with coefficients $r(j, l)$ to P 'hidden units' in a hidden layer,
   $$\sum_{l=1}^{L} r(j, l) f_l(x), \; j = 1, 2, \ldots, P, \quad (7)$$
   as depicted in Fig. 2.
   The values of each hidden unit are then input to a *sigmoidal function* that is differentiable and monotonically increasing. For example, the hyperbolic tangent function
   $$\sigma(\xi) = \tanh(\xi) = \frac{e^{\xi} - e^{-\xi}}{e^{\xi} + e^{-\xi}},$$
   or the logistic function
   $$\sigma(\xi) = \frac{1}{1 + e^{-\xi}}$$
   can be used. The sigmoidal functions should satisfy
   $$-\infty < \lim_{\xi \to -\infty} \sigma(\xi) < \lim_{\xi \to \infty} \sigma(\xi) < \infty.$$

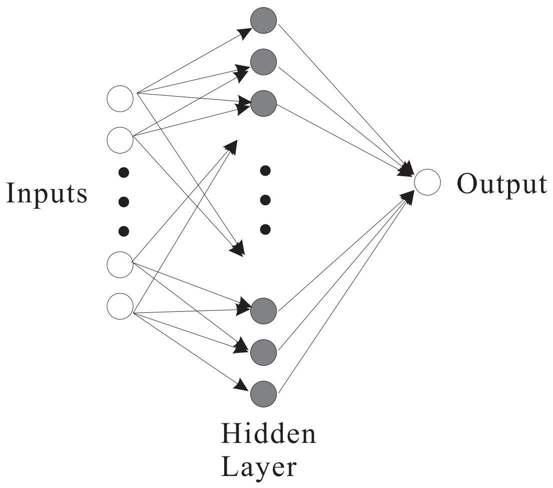


Fig. 2. An example of the structure of a neural network mapping

The output scalars of the sigmoidal function are linearly mapped again to generate one output value of the overall architecture,

$$\tilde{J}(x,r) = \sum_{j=1}^{P} r(j)\sigma\left(\sum_{l=1}^{L} r(j,l) f_l(x)\right). \tag{8}$$

Coefficients $r(j)$ and $r(j,l)$ in (7) are called the weights of the network. The weights are obtained from the training process of the algorithm.

2. *A feature extraction mapping.* An alternative architecture directly combines the feature vector $f(x)$ in a linear fashion, without using a neural network. The output of the architecture involves coefficients $r(l), l = 0,1,2,\dots,L$,

$$\tilde{J}(x,r) = r(0) + \sum_{l=1}^{L} r(l) f_l(x). \tag{9}$$

An application of NDP that deals with playing strategies in a Tetris game involves such an architecture [2]. While this is attractive due to its simplicity, we did not find this architecture effective in our setting. The principal difficulty was that the iterative technique we used to determine r failed to converge.

3. *A heuristic mapping.* A third way to construct the approximate structure is based on existing heuristic controls. Heuristic controls are easy to implement and produce decent solutions in a reasonable amount of time. Although not optimal, some of the heuristic costs $H_u(x)$ are likely to be fairly close to the optimal cost function $J(x)$. $H_u(x)$ is evaluated by averaging results of simulations, in which policy u is applied in every

stage. In the heuristic mapping architecture, the heuristic costs are suitably weighted to obtain a good approximation of J. Given a state x and heuristic controls $u_i, i = 1, 2, \dots, I$, the approximate form of J is

$$\tilde{J}(x, r) = r(0) + \sum_{i=1}^{I} r(i) H_{u_i}(x), \tag{10}$$

where r is the overall tunable parameter vector of the architecture.
The more heuristic policies that are included in the training, the more accurate the approximation is expected to be. With proper tuning of the parameter vector r, we hope to obtain a policy that performs better than all of the heuristic policies. However, each evaluation of $H_{u_i}(x)$ is potentially expensive.

3.3 Approximate Policy Iteration Using Monte-Carlo Simulation.

The method we consider in this subsection is an approximate version of policy iteration. A sequence of policies $\{u_k\}$ is generated and the corresponding approximate cost functions $\tilde{J}(x, r)$ are used in place of $J(x)$. The NDP algorithms are based on the architectures described previously. The training of the parameter vector r for the architecture is performed using a combination of Monte-Carlo simulation and least squares fitting.

The NDP algorithm we use is called *approximate policy iteration* (API) using Monte-Carlo simulation. API alternates between approximate policy evaluation steps (simulation) and policy improvement steps (training). Policies are iteratively updated from the outcomes of simulation. We expect the policies will converge after several iterations, but there is no theoretical guarantee of the convergence. Such an iteration process is illustrated in Fig. 3.

Simulation Step

Simulating sample trajectories starts with an initial state $x_0 = 0$, corresponding to no dose delivery. At the kth stage, an approximate cost-to-go function $\tilde{J}_{k+1}(x_{k+1}, r_{k+1})$ for the next stage determines the policy $\hat{u}_k$ via the equation (6), using the knowledge of the transition probabilities. We can then simulate x_{k+1} using the calculated $\hat{u}_k$ and a realization of ω_k. This process can be repeated to generate a collection of sample trajectories. In this simulation step, the parameter vectors $r_k, k = 0, 1, \dots, N-1$, (which induce the policy $\hat{u}_k$) remain fixed as all the sample trajectories are generated.

Simulation generates sample trajectories $\{x_{0,i} = 0, x_{1,i}, \dots, x_{N,i}\}, i = 1, 2, \dots, M$. The corresponding sample cost-to-go for every transition state is equal to the cumulative instantaneous costs plus a final cost,

$$c(x_{k,i}) = \sum_{j=k}^{N-1} g(x_{j,i}, x_{j+1,i}, \hat{u}_j) + J_N(x_{N_i}).$$

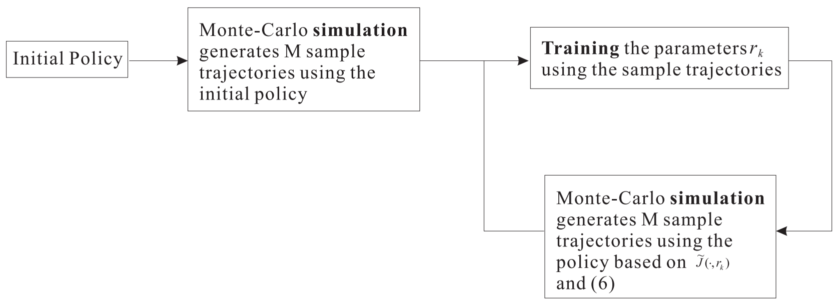

Initial Policy
Monte-Carlo **simulation** generates M sample trajectories using the initial policy
Training the parameters r_k using the sample trajectories
Monte-Carlo **simulation** generates M sample trajectories using the policy based on $\tilde{J}(\cdot, r_k)$ and (6)

Fig. 3. Simulation and training in API. Starting with an initial policy, the Monte-Carlo simulation generates a number of sample trajectories. The sample costs at each stage are input into the training unit in which r_k's are updated by minimizing the least squares error. New sample trajectories are simulated using the policy based on the approximate structure $\tilde{J}(\cdot, r_k)$ and (6). This process is repeated.

Training Step

In the training process, we evaluate the cost, and update the r_k by solving a least squares problem at each stage $k = 0, 1, \dots, N-1$,

$$\min_{r_k} \frac{1}{2} \sum_{i=1}^{M} |\tilde{J}_k(x_{k,i}, r_k) - c(x_{k,i})|^2. \tag{11}$$

The least squares problem (11) penalizes the difference of approximate cost-to-go estimation $\tilde{J}_k(x_{k,i}, r_k)$ and sample cost-to-go value $c(x_{k,i})$. It can be solved in various ways.

In practice, we divide the M generated trajectories into M_1 batches, with each batch containing M_2 trajectories.

$$M = M_1 * M_2.$$

The least squares formulation (11) is equivalently written as

$$\min_{r_k} \sum_{m=1}^{M_1} \left(\frac{1}{2} \sum_{x_{k,i} \in \text{Batch}_m} |\tilde{J}_k(x_{k,i}, r_k) - c(x_{k,i})|^2 \right). \tag{12}$$

We use a gradient-like method that processes each least squares term

$$\frac{1}{2} \sum_{x_{k,i} \in \text{Batch}_m} |\tilde{J}_k(x_{k,i}, r_k) - c(x_{k,i})|^2 \tag{13}$$

incrementally. The algorithm works as follows: Given a batch of sample state trajectories (M_2 trajectories), the parameter vector r_k is updated by

$$r_k := r_k - \gamma \sum_{x_{k,i} \in \text{Batch}_m} \nabla \tilde{J}_k(x_{k,i}, r_k) \left(\tilde{J}(x_{k,i}, r_k) - c(x_{k,i}) \right),$$
$$k = 0, 1, \ldots, N-1. \tag{14}$$

Here γ is a stepsize length that should decrease monotonically as the number of batches used increases (see Proposition 3.8 in [3]). A suitable step length choice is $\gamma = \alpha/m, m = 1, 2, \ldots, M_1$, in the mth batch, where α is a constant scalar. The summation in the right-hand side of (14) is a gradient evaluation corresponding to (13) in the least squares formulation. The parametric vectors r_k are updated via the iteration (14), as a batch of trajectories become available. The incremental updating scheme is motivated by the stochastic gradient algorithm (more details are given in [3]).

In API, the r_k's are kept fixed until all the M sample trajectories are generated. In contrast to this, another form of the NDP algorithm, called *optimistic policy iteration* (OPI), updates the r_k more frequently, immediately after a batch of trajectories are generated. The intuition behind OPI is that the new changes on policies are incorporated rapidly. This 'optimistic' way of updating r_k is subject to further investigation.

Ferris and Voelker [10] applied a rollout policy to solve this same problem. The approximation is built by applying the particular control u at stage k and a control (base) policy at all future stages. This procedure ignores the training part of our algorithm. The rollout policy essentially suggests a simple form of

$$\tilde{J}(x) = H_{base}(x).$$

The simplification results in a biased estimation of $J(x)$, because the optimal cost-to-go function strictly satisfies: $J(x) \leq H_{base}(x)$. In our new approach, we use an approximate functional architecture for the cost-to-go function, and the training process will determine the parameters in the architecture.

4 Computational Experimentation

4.1 A Simple Example

We first experiment on a simple one dimensional fractionation problem with several variations of the approximating architectures described in the preceding section. The setting consists of a total of 15 voxels $\{1, 2, \ldots, 15\}$, where the target voxel set, $\mathcal{T} = \{3, 4, \ldots, 13\}$ is located in the center. Dose is delivered to the target voxels, and due to the random positioning error of the patient, a portion of dose is delivered outside of the target. We assume a maximum shift of 2 voxels to the left or right.

In describing the cost function, our weighting scheme assigns relatively high weights on the target, and low weights elsewhere:

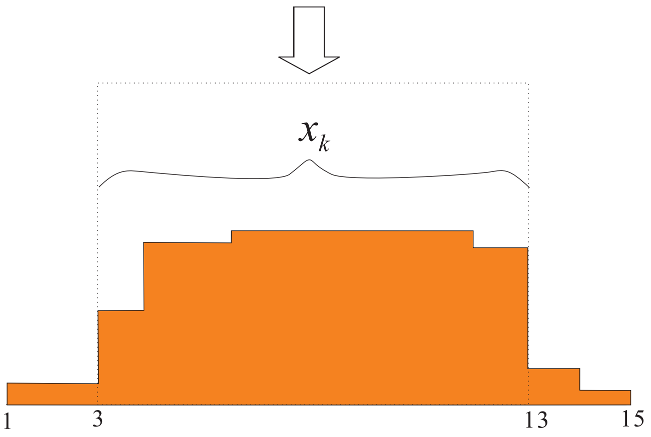


Fig. 4. A simple one-dimension problem. x_k is the dose distribution over voxels in the target: voxels $3, 4, \ldots, 13$.

$$p(i) = \begin{cases} 10, \ i \in \mathcal{T}; \\ 1, \quad i \notin \mathcal{T}. \end{cases}$$

Definitions of final error and one step error refer to (4) and (5).

For the target volume above, we also consider two different probability distributions for the random shift ω_k. In the low volatility examples, we have

$$\omega_k = \begin{cases} -2, \text{ with probability } 0.02 \\ -1, \text{ with probability } 0.08 \\ 0, \quad \text{with probability } 0.8 \\ 1, \quad \text{with probability } 0.08 \\ 2, \quad \text{with probability } 0.02, \end{cases}$$

for every stage k. The high volatility examples have

$$\omega_k = \begin{cases} -2, \text{ with probability } 0.05 \\ -1, \text{ with probability } 0.25 \\ 0, \quad \text{with probability } 0.4 \\ 1, \quad \text{with probability } 0.25 \\ 2, \quad \text{with probability } 0.05, \end{cases}$$

for every stage k. While it is hard to estimate the volatilities present in the given application, the results are fairly insensitive to these choices.

To apply the NDP approach, we should provide a rich collection of policies for the set $U(x_k)$. In our case, $U(x_k)$ consists of a total number of A modified reactive policies,

$$U(x_k) = \{u_{k,1}, u_{k,2}, \ldots, u_{k,A} | \; u_{k,i} = a_i \cdot \max(0, T - x_k)/(N - k)\}, \qquad (15)$$

where a_i is a numerical scalar indicating an augmentation level to the standard reactive policy delivery; here $A = 5$ and

$$\boldsymbol{a} = \{1, 1.4, 1.8, 2.2, 2.6\}.$$

We apply two of the approximation architectures in Sect. 3.2, the neural network/multilayer (NN) perceptron architecture, and linear architecture using a heuristic mapping. The details follow.

1. API using Monte-Carlo simulation and neural network architecture.
   For the NN architecture, after experimentation with several different sets of features, we used the following six features $f_j(x), j = 1, 2, \ldots, 6$:
   a) Average dose distribution in the left rind of the target organ:
   $$\text{mean of } \{x(i), i = 3, 4, 5\}.$$
   b) Average dose distribution in the center of the target organ:
   $$\text{mean of } \{x(i), i = 6, 7, \ldots, 10\}.$$
   c) Average dose distribution in the right rind of the target organ:
   $$\text{mean of } \{x(i), i = 11, 12, 13\}.$$
   d) Standard deviation of the overall dose distribution in the target.
   e) Curvature of the dose distribution. The curvature is obtained by fitting a quadratic curve over the values $\{x_i, i = 3, 4, \ldots, 13\}$, and extracting the curvature.
   f) A constant feature $f_6(x) = 1$.

   In features (a)–(c), we distinguish the average dose on different parts of the structure, because the edges commonly have both underdose and overdose issues, while the center is delivered more accurately.
   In the construction of neural network formulation, a hyperbolic tangent function was used as the sigmoidal mapping function. The neural network has 6 inputs (6 features), 8 hidden sigmoidal units, and 1 output, such that weight of neural network r_k is a vector of length 56.
   In each simulation, a total of 10 policy iterations were performed. Running more policy iterations did not show further improvement. The initial policy used was the standard reactive policy $\boldsymbol{u}$: $u_k = \max(0, T - x_k)/(N - k)$. Each iteration involved $M_1 = 15$ batches of sample trajectories, with $M_2 = 20$ trajectories in each batch to train the neural network.
   To train the r_k in this approximate architecture, we started with $r_{k,0}$ as a vector of ones, and used an initial step length $\gamma = 0.5$.

2. API using Monte-Carlo simulation and the linear architecture of heuristic mapping.
   Three heuristic policies were involved as base policies: (1) constant policy $\boldsymbol{u}_1$: $u_{1,k} = T/N$, for all k; (2) standard reactive policy $\boldsymbol{u}_2$: $u_{2,k} = \max(0, T - x_k)/(N - k)$, for all k; (3) modified reactive policy $\boldsymbol{u}_3$ with the amplifying parameter $a = 2$ applied at all stages except the last one. For the stage $k = N - 1$, it simply delivers the residual dose:
$$u_{3,k} = \begin{cases} 2 \cdot \max(0, T - x_k)/(N - k), & k = 0, 1, \ldots, N - 2, \\ \max(0, T - x_k)/(N - k), & k = N - 1. \end{cases}$$
   This third choice facilitates a more aggressive treatment in early stages. To evaluate the heuristic cost $H_{u_i}(x_k), i = 1, 2, 3$, 100 sub-trajectories starting with x_k were generated for periods k to N. The training scheme was analogous to above method. A total of 10 policy iterations were performed. The policy used in the first iteration was the standard reactive policy. All iterations involved $M_1 = 15$ batches of sample trajectories, with $M_2 = 20$ trajectories in each batch, a total of 300 trajectories.
   Running the heuristic mapping architecture requires a great deal of computation, because it requires evaluating the heuristic costs by sub-simulations.

The fractionation radiotherapy problem is solved using both techniques with $N = 3, 4, 5, 10, 14$ and 20 stages. Fig. 5 shows performance of API using a heuristic mapping architecture, in a low volatility case. The starting policy is the standard reactive policy, that has expected error (cost) of 0.48 (over $M = 300$ sample trajectories). The policies u_k converge after around 7 policy iterations, taking around 20 minutes on a PIII 1.4GHz machine. After the training, the expected error decreases to 0.30, which is reduced by about 40% compared to the standard reactive policy.

The main results of training and simulation with two probability distributions are plotted in Fig. 6. This one-dimension example is small, but the revealed patterns are informative. For each plot, the results of the constant policy, reactive policy and NDP policy are displayed. Due to the significant randomness in the high volatility case, it is more likely to induce underdose in the rind of target, which is penalized heavily with our weighting scheme. Thus as volatility increases, so does the error. Note that in this one-dimensional problem, an ideal total amount of dose delivered to target is 11, which can be compared with the values on the vertical axes of the plots (which are multiplied by the vector p).

Comparing the figures, we note the remarkable similarities. Common to all examples is the poor performance of the constant policy. The reactive policy performs better than the constant policy, but not as well as the NDP policy in either architecture. The constant policy does not change much with number of total fractions. The level of improvement depends upon which NDP approximate structure to use. The NN architecture performs better than the heuristic mapping architecture, when N is small. While N is large, they do not show significant difference.

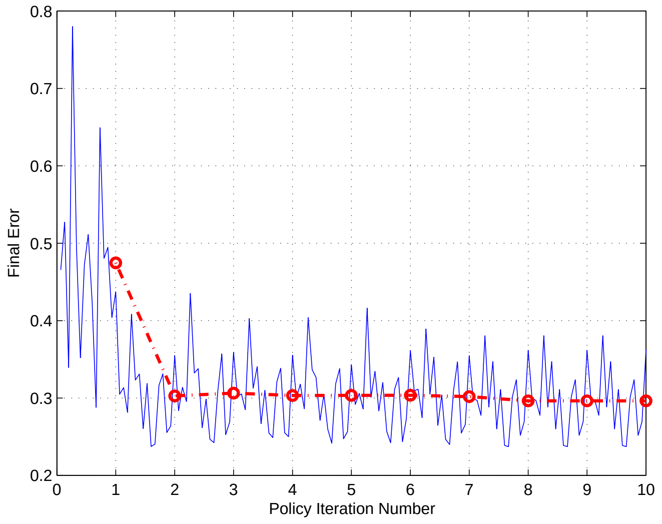


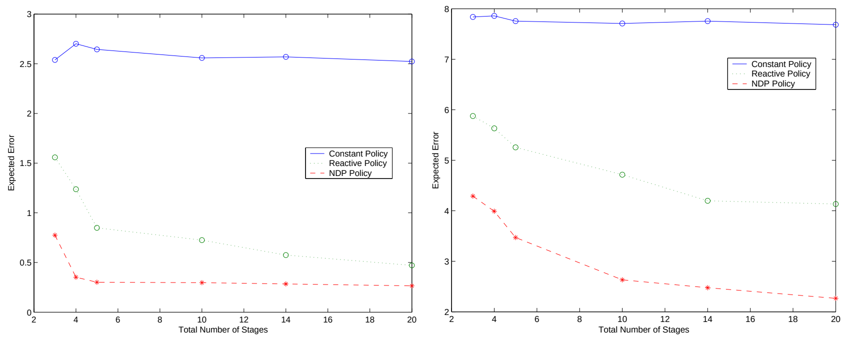
Fig. 5. Performance of API using heuristic cost mapping architecture, $N = 20$. For every iteration, we plot the average (over $M_2 = 20$ trajectories) of each of $M_1 = 15$ batches. The broken line represents the mean cost in each iteration.

4.2 A Real Patient Example: Head & Neck Tumor

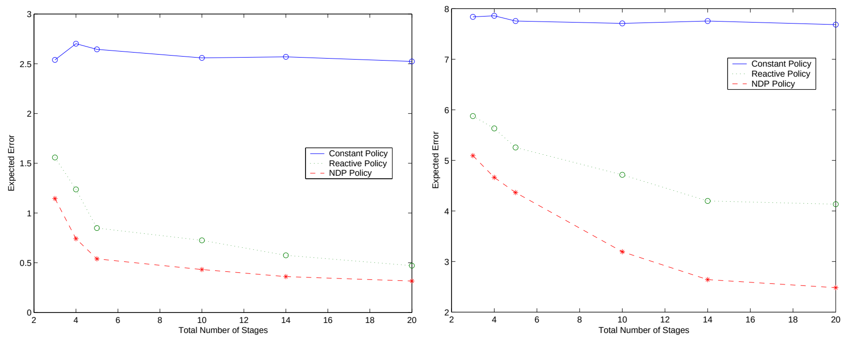
In this subsection, we apply our NDP techniques to a real patient problem – a head & neck tumor. In the head & neck tumor scenario, the tumor volume covers a total of 984 voxels in space. As noted in Fig. 7, the tumor is circumscribed by two critical organs: the mandible and the spinal cord. We will perform analogous techniques as in the above simple example. The weight setting is the same:

$$p(i) = \begin{cases} 10, \; i \in \mathcal{T}; \\ 1, \quad i \notin \mathcal{T}. \end{cases}$$

In our problem setting, we do not distinguish between critical organs and other normal tissue. In reality, a physician also takes in account radiation damage to the surrounding critical organs. For this reason, a higher penalty weight is usually assigned on these organs.



(a) NN architecture in low volatility (b) NN architecture in high volatility



(c) Heuristic mapping architecture in low volatility (d) Heuristic mapping architecture in high volatility

Fig. 6. Comparing the constant, reactive and NDP policies in low and high volatility cases

ω_k are now three-dimension random vectors. By assumption of independence of each component direction, we have

$$Pr(\omega_k = [i, j, k]) = Pr(\omega_{k,x} = i) \cdot Pr(\omega_{k,y} = j) \cdot Pr(\omega_{k,z} = k) \tag{16}$$

In the low and high volatility cases, each component of ω_k follows a discrete distribution (also with a maximum shift of two voxels),

$$\omega_{k,i} = \begin{cases} -2, \text{ with probability } 0.01 \\ -1, \text{ with probability } 0.06 \\ 0, \quad \text{with probability } 0.86 \\ 1, \quad \text{with probability } 0.06 \\ 2, \quad \text{with probability } 0.01, \end{cases}$$

and

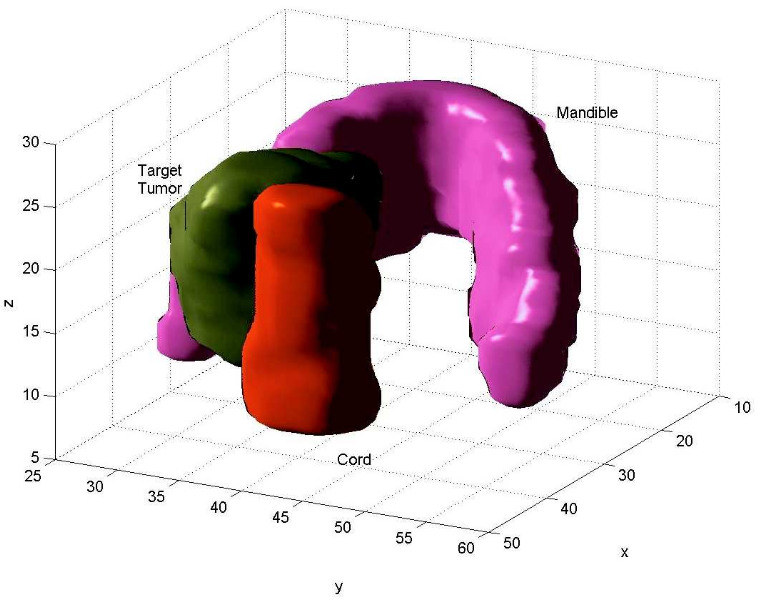


Fig. 7. Target tumor, cord and mandible in the head & neck problem scenario

$$\omega_{k,i} = \begin{cases} -2, \text{ with probability } 0.05 \\ -1, \text{ with probability } 0.1 \\ 0, \quad \text{with probability } 0.7 \\ 1, \quad \text{with probability } 0.1 \\ 2, \quad \text{with probability } 0.05, \end{cases}$$

We adjust the $\omega_{k,i}$ by smaller amounts than in the one dimension problem, because the overall probability is the product of each component (16); the resulting volatility therefore grows.

For each stage, $U(x_k)$ is a set of modified reactive policies, whose augmentation levels include

$$\boldsymbol{a} = \{1, 1.5, 2, 2.5, 3\}.$$

For the stage $k = N - 1$ (when there are two stages to go), setting the augmentation level $a > 2$ is equivalent to delivering more than the residual dose, which is unnecessary for treatment. In fact, the NDP algorithm will ignore these choices.

The approximate policy iteration algorithm uses the same two architectures as in Sect. 4.1. However, for the neural network architecture we need an extended 12 dimensional input feature space:

(a) Features 1–7 are the mean value of the dose distribution of the left, right, up, down, front, back and center part of the tumor.
(b) Feature 8 is the standard deviation of dose distribution in the tumor volume.
(c) Feature 9–11. We extract the dose distribution on 3 lines through center of the tumor. Lines are from left to right, from up to down, and from front to back. Features 9–11 are the estimated curvature of the dose distribution on the three lines.
(d) Feature 12 is a constant feature, set as 1.

In the neural network architecture, we build 1 hidden layer, with 16 hidden sigmoidal units. Therefore, each r_k for $\tilde{J}(x, r_k)$ is of length 208.

We still use 10 policy iterations. (Later experimentation shows that 5 policy iterations are enough for policy convergence.) In each iteration, simulation generates a total of 300 sample trajectories, that are grouped in $M_1 = 15$ batches of sample trajectories, with $M_2 = 20$ in each batch, to train the parameter r_k.

One thing worth mentioning here is, the initial step length scaler γ in (14) is set to a much smaller value in the 3D problem. In the head & neck case, we set $\gamma = 0.00005$ as compared to $\gamma = 0.5$ in the one dimension example. A plot, Fig. 8, shows the reduction of expected error as the number of policy iteration increases.

The alternative architecture for $\tilde{J}(x, r)$ using a linear combination of heuristic costs is implemented precisely as in the one dimension example.

The overall performance of this second architecture is very slow, due to the large amount of work in evaluation of the heuristic costs. It spends a considerable time in the simulation process generating sample sub-trajectories. To save computation time, we propose an approximate way of evaluating each candidate policy in (6). The expected cost associated with policy u_k is

$$\begin{aligned}&\mathbb{E}[g(x_k, x_{k+1}, u_k) + \tilde{J}_{k+1}(x_{k+1}, r_{k+1})] \\ &= \sum_{\omega_{k,1}=-2}^{2} \sum_{\omega_{k,2}=-2}^{2} \sum_{\omega_{k,2}=-2}^{2} Pr(\omega_k)[g(x_k, x_{k+1}, u_k) + \tilde{J}_{k+1}(x_{k+1}, r_{k+1})].\end{aligned}$$

For a large portion of ω_k, the value of $Pr(\omega_k)$ almost vanishes to zero when it makes a two-voxel shift in each direction. Thus, we only compute the sum of cost over a subset of possible ω_k,

$$\sum_{\omega_{k,1}=-1}^{1} \sum_{\omega_{k,2}=-1}^{1} \sum_{\omega_{k,2}=-1}^{1} Pr(\omega_k)[g(x_k, x_{k+1}, u_k) + \tilde{J}_{k+1}(x_{k+1}, r_{k+1})].$$

A straightforward calculation shows that we reduce a total of $125(= 5^3)$ evaluations of state x_{k+1} to $27(= 3^3)$. The final time involved in training the architecture is around 10 hours.

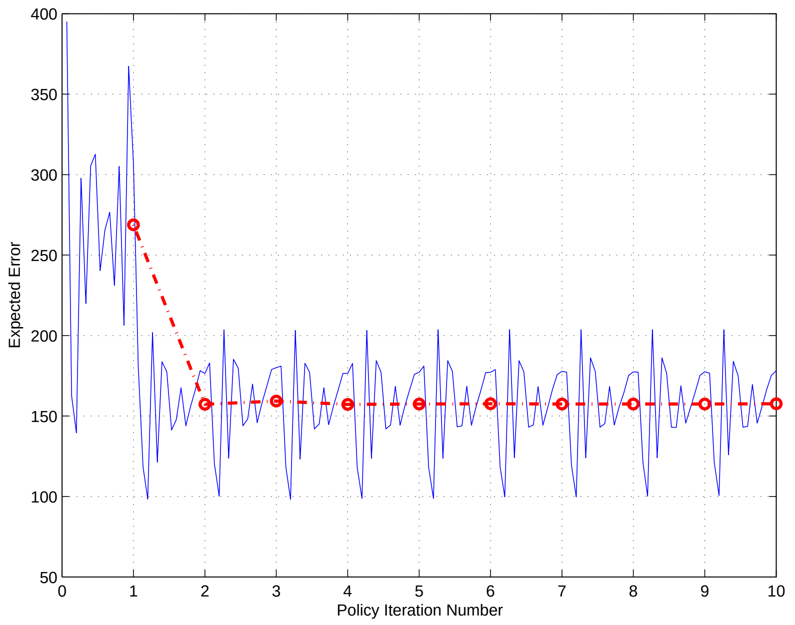
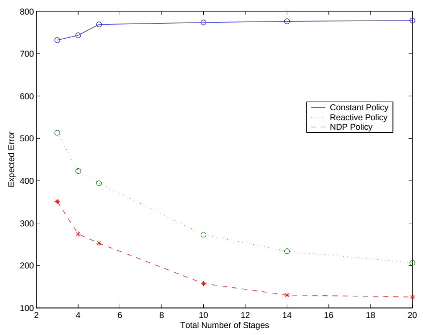


Fig. 8. Performance of API using neural-network architecture, $N = 11$. For every iteration, we plot the average (over $M_2 = 20$ trajectories) of each of $M_1 = 15$ batches. The broken line represents the mean cost in each policy iteration.

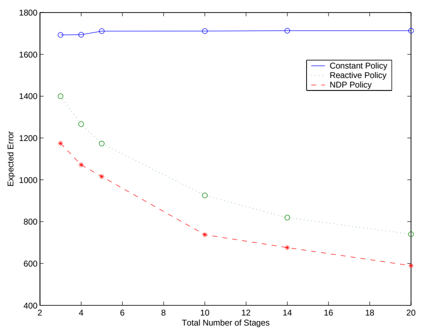
Again, we plot the results of constant policy, reactive policy and NDP policy in the same figure. We still investigate on the cases where $N = 3, 4, 5, 14, 20$. As we can observe in all sub-figures in Fig. 9, the constant policy still performs the worst in both high and low volatility cases. The reactive policy is better and the best policy is the NDP policy. As the total number of stages increases, the constant policy remains almost at the same level, but the reactive and NDP continue to improve. The poor constant policy is a consequence of significant underdose near the edge of the target.

The two approximating architectures perform more or less the same, though the heuristic mapping architecture takes significantly more time to train. Focusing on the low volatility cases, Fig. 9 (a) and (c), we see the heuristic mapping architecture outperforms the NN architecture when N is small, i.e. $N = 3, 4, 5, 10$. When $N = 20$, the expected error is reduced to the lowest, about 50% from reactive policy to NDP policy. When N is small, the improvement ranges from 30% to 50%. When the volatility is high, it undoubtedly induces more error than in low volatility. Not only the expected error, but the variance escalates to a large value as well.

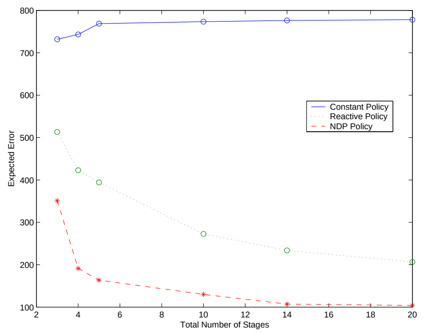
For the early fractions of the treatment, the NDP algorithm intends to select aggressive policies, i.e., the augmentation level $a > 2$, while in the later stage time, it intends to choose more conservative polices. As the weighting



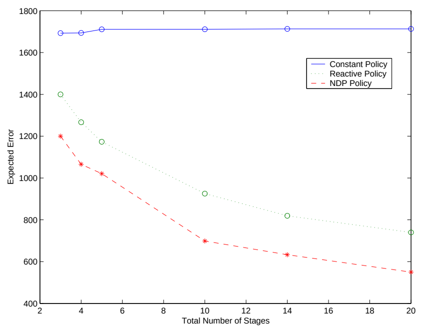
(a) NN architecture in low volatility



(b) NN architecture in high volatility



(c) Heuristic mapping architecture in low volatility



(d) Heuristic mapping architecture in high volatility

Fig. 9. Head & neck problem – comparing constant, reactive and NDP policies in two probability distributions

factor for target voxels is 10, aggressive policies are preferred in the early stage because it leaves room to correct the delivery error on the target in the later stages. However, it may be more likely to cause delivery error on the normal tissue.

4.3 Discussion

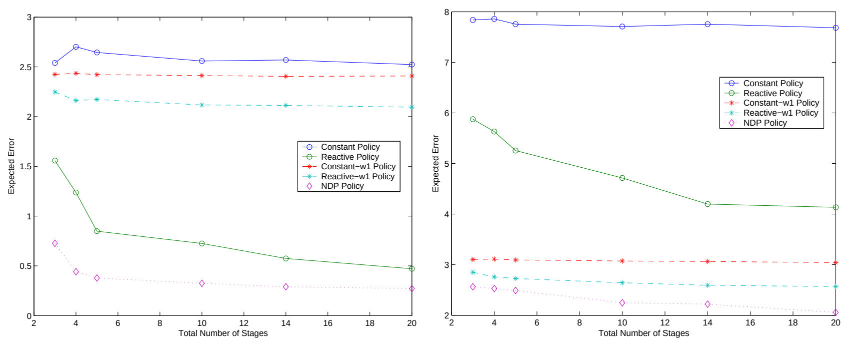
The number of candidate policies used in training is small. Once we have the optimal r_k after simulation and training procedures, we can select u_k from an extended set of policies $U(x_k)$ (via (6)) using the approximate cost-to-go functions $\tilde{J}(x, r_k)$, improving upon the current results.

For instance, we can introduce a new class of policies that cover a wider delivery region. This class of clinically favored policies include a safety margin around the target. The policies deliver the same dose to voxels in the margin as that delivered to the nearest voxels in the target. As an example policy in the class, a constant-w1 policy (where ‘w1’ means ‘1 voxel wider’) is an extension of the constant policy, covering a 1-voxel thick margin around the target. As in the one-dimensional example in Sect. 4.1, the constant-w1 policy is defined as:

$$u_k(i) = \begin{cases} T(i)/N, & \text{for } i \in \mathcal{T}, \\ T(3)/N = T(13)/N, & \text{for } i = 2 \text{ or } 14, \\ 0, & \text{elsewhere,} \end{cases}$$

where the voxel set $\{2, 14\}$ represents the margin of the target. The reactive-w1 policies and the modified reactive-w1 policies are defined accordingly. (We prefer to use 'w1' policies rather than 'w2' policies because 'w1' policies are observed to be uniformly better.)

The class of 'w1' policies are preferable to apply in the high volatility case, but not in the low volatility case (see Fig. 10). For the high volatility case, the policies reduce the underdose error significantly, which is penalized 10 times as heavy as the overdose error, easily compensating for the overdose error they introduce outside of the target. In the low volatility case, when the underdose is not as severe, they inevitably introduce redundant overdose error.



(a) Heuristic mapping architecture in low volatility (b) Heuristic mapping architecture in high volatility

Fig. 10. In the one-dimensional problem, NDP policies with extended policy set $U(x_k)$

The NDP technique was applied to an enriched policy set $U(x_k)$, including the constant, constant-w1, reactive, reactive-w1, modified reactive and modified reactive-w1 policies. It automatically selected an appropriate policy at each stage based on the approximated cost-to-go function, and outperformed every component policy in the policy set. In Fig. 10, we show the result of the one-dimensional example using the heuristic mapping architecture for NDP. As we have observed, in the low volatility case, the NDP policy tends to be the reactive or the modified reactive policy, while in the high volatility case, it is more likely to be the reactive-w1 or the modified reactive-w1 policy. Comparing to the NDP policies in Fig. 6, we see that increasing the choices of policies in $U(x_k)$, the NDP policy generates lower expected error.

Another question concerns about how much difference occurs when switching to another weighting scheme. Setting a high weighting factor on the target is rather arbitrary. This will also influence the NDP in selecting policies. In addition, we changed the setting of weighting scheme to

$$p(i) = \begin{cases} 1, \, i \in \mathcal{T}; \\ 1, \, i \notin \mathcal{T}. \end{cases}$$

and ran the experiment on the real example (Sect. 4.2) again. In Fig. 11, we discovered the same pattern of results, while this time, all the error curves were scaled down accordingly. The difference between constant and reactive policy decreased. The NDP policy showed an improvement of around 12% over the reactive policy when $N = 10$.

We even tested the weighting scheme

$$p(i) = \begin{cases} 1, \quad i \in \mathcal{T}; \\ 10, \, i \notin \mathcal{T}, \end{cases}$$

which reverted the importance of the target and the surrounding tissue. It resulted in a very small amount of delivered dose in the earlier stages, and at the end the target was severely underdosed. The result was reasonable because the NDP policy was cautious to deliver any dose outside of the target at each stage.

5 Conclusion

Solving an optimal on-line planning strategy in fractionated radiation treatment is quite complex. In this paper, we set up a dynamic model for the day-to-day planning problem. We assume that the probability distribution of patient motion can be estimated by means of prior inspection. In fact, our experimentation on both high and low volatility cases display very similar patterns.

Although methods such as dynamic programming obtain exact solutions, the computation is intractable. We exploit neuro-dynamic programming tools to derive approximate DP solutions, that can be solved with much fewer computational resources. The API algorithm we apply iteratively switches between Monte-Carlo simulation steps and training steps, whereby the feature based approximating architectures of the cost-to-go function are enhanced as

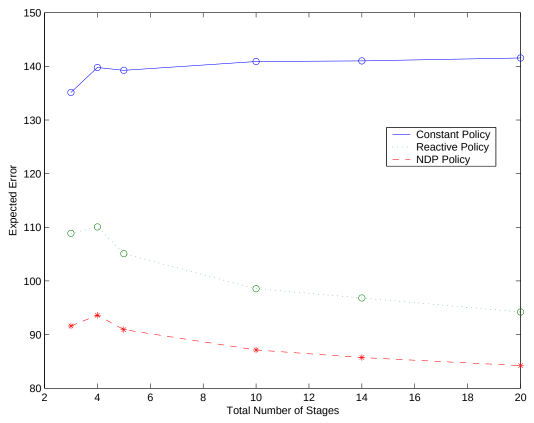


Fig. 11. Head & neck problem. Using API with a neural network architecture, in a low volatility case, with identical weight on the target and normal tissue.

the algorithm proceeds. The computational results are based on a finite policy set for training. In fact, the final approximate cost-to-go structures can be used to facilitate selecting from a larger set of candidate policies, extended from the training set.

We jointly compare the on-line policies with an off-line constant policy, that simply delivers a fixed dose amount in each fraction of treatment. The on-line policies are shown to be significantly better than the constant policy, in terms of total expected delivery error. In most of the cases, the expected error is reduced more than a half. The NDP policy performs preferentially, enhancing the reactive policy for all our tests. Future work needs to address further timing improvement.

We have tested two approximation architectures. One uses a neural network and the other is based on existing heuristic policies, both of which perform similarly. The heuristic mapping architecture is slightly better than the neural network based architecture, but it takes significantly more computational time to evaluate. As these examples have demonstrated, neuro-dynamic programming is a promising supplement to heuristics in discrete dynamic optimization.

References

1. D. P. Bertsekas. *Dynamic Programming and Optimal Control.* Athena Scientific, Belmont MA, 1995.
2. D. P. Bertsekas and S. Ioffe. Temporal differences-based policy iteration and applications in neuro-dynamic programming. *Report LIDS-P-2349, Lab. for Information and Decision Systems, MIT*, 1996.
3. D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, Belmont, Massachusetts, 1996.

4. J. R. Birge and R. Louveaux. *Introduction to Stochastic Programming.* Springer, New York, 1997.
5. M. Birkner, D. Yan, M. Alber, J. Liang, and F. Nusslin. Adapting inverse planning to patient and organ geometrical variation: Algorithm and implementation. *Medical Physics*, 30(10):2822–31, 2003.
6. Th. Bortfeld. Current status of IMRT: physical and technological aspects. *Radiotherapy and Oncology*, 61(2):291–304, 2001.
7. C. L. Creutzberg, G. V. Althof, M. de Hooh, A. G. Visser, H. Huizenga, A. Wijnmaalen, and P. C. Levendag. A quality control study of the accuracy of patient positioning in irradiation of pelvic fields. *International Journal of Radiation Oncology, Biology and Physics*, 34:697–708, 1996.
8. M. C. Ferris, J.-H. Lim, and D. M. Shepard. Optimization approaches for treatment planning on a Gamma Knife. *SIAM Journal on Optimization*, 13:921–937, 2003.
9. M. C. Ferris, J.-H. Lim, and D. M. Shepard. Radiosurgery treatment planning via nonlinear programming. *Annals of Operations Research*, 119:247–260, 2003.
10. M. C. Ferris and M. M. Voelker. Fractionation in radiation treatment planning. *Mathematical Programming B*, 102:387–413, 2004.
11. A. Gosavi. *Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning.* Kluwer Academic Publishers, Norwell, MA, USA, 2003.
12. M. A. Hunt, T. E. Schultheiss, G. E. Desobry, M Hakki, and G. E. Hanks. An evaluation of setup uncertainties for patients treated to pelvic fields. *International Journal of Radiation Oncology, Biology and Physics*, 32:227–33, 1995.
13. P. Kall and S. W. Wallace. *Stochastic Programming.* John Wiley & Sons, Chichester, 1994.
14. K. M. Langen and T. L. Jones. Organ motion and its management. *International Journal of Radiation Oncology, Biology and Physics*, 50:265–278, 2001.
15. J. G. Li and L. Xing. Inverse planning incorporating organ motion. *Medical Physics*, 27(7):1573–1578, July 2000.
16. A. Niemierko. Optimization of 3D radiation therapy with both physical and biological end points and constraints. *International Journal of Radiation Oncology, Biology and Physics*, 23:99–108, 1992.
17. W. Schlegel and A. Mahr, editors. *3D Conformal Radiation Therapy - A Multimedia Introduction to Methods and Techniques.* Springer-Verlag, Berlin, 2001.
18. D. M. Shepard, M. C. Ferris, G. Olivera, and T. R. Mackie. Optimizing the delivery of radiation to cancer patients. *SIAM Review*, 41:721–744, 1999.
19. J. Unkelback and U. Oelfke. Inclusion of organ movements in IMRT treatment planning via inverse planning based on probability distributions. *Institute of Physics Publishing, Physics in Medicine and Biology*, 49:4005–4029, 2004.
20. J. Unkelback and U. Oelfke. Incorporating organ movements in inverse planning: Assessing dose uncertainties by Bayesian inference. *Institute of Physics Publishing, Physics in Medicine and Biology*, 50:121–139, 2005.
21. L. J. Verhey. Immobilizing and positioning patients for radiotherapy. *Seminars in Radiation Oncology*, 5(2):100–113, 1995.
22. M. M. Voelker. *Optimization of Slice Models.* PhD thesis, University of Wisconsin, Madison, Wisconsin, December 2002.
23. S. Webb. *The Physics of Conformal Radiotherapy: Advances in Technology.* Institute of Physics Publishing Ltd., 1997.