Rounded Dynamic Programming for Tree-Structured Stochastic Network Design

Xiaojian Wu and **Daniel Sheldon** and **Shlomo Zilberstein**
School of Computer Science
University of Massachusetts Amherst
{xiaojian,sheldon,shlomo}@cs.umass.edu

Abstract

We develop a fast approximation algorithm called rounded dynamic programming (RDP) for stochastic network design problems on directed trees. The underlying model describes phenomena that spread away from the root of a tree, for example, the spread of influence in a hierarchical organization or fish in a river network. Actions can be taken to intervene in the network—for some cost—to increase the probability of propagation along an edge. Our algorithm selects a set of actions to maximize the overall spread in the network under a limited budget. We prove that the algorithm is a fully polynomial-time approximation scheme (FPTAS), that is, it finds $(1-\epsilon)$-optimal solutions in time polynomial in the input size and $1/\epsilon$. We apply the algorithm to the problem of allocating funds efficiently to remove barriers in a river network so fish can reach greater portions of their native range. Our experiments show that the algorithm is able to produce near-optimal solutions much faster than an existing technique.

Introduction

Many influence maximization problems can be formulated as *stochastic network design* problems—represented as directed graphs in which edges are present or absent stochastically (Sheldon et al. 2010). When an edge is present, it allows the flow of influence between two neighboring vertices. Actions can be taken to increase the probability of certain edges being present. Unlike the well-known *source-selection* influence maximization problem of Kempe, Kleinberg, and Tardos (2003), where actions determine where to *start* the spread of influence, actions in this model intervene directly to modify the dynamics of the model. For example, to facilitate the rollout of new policies within a large company, actions might provide incentives for managers to push new policies to their subordinates upon being notified by their own bosses. We assume that the actions are costly and that there is a budget that limits the total cost of the actions that can be taken. The objective is to maximize a total measure of reward associated with nodes that become influenced. So, in our example, one might focus incentives on less reliable managers or those with large units to reach as many employees as possible.

Many ecological sustainability problems can be modeled this way, for example, maximizing the spread of an endangered species in some geographical region over time (Sheldon et al. 2010; Golovin et al. 2011; Kumar, Wu, and Zilberstein 2012; Wu, Sheldon, and Zilberstein 2013).

The general stochastic network design problem is intractable. It includes source-selection influence maximization as a special case, and thus it is NP-hard to approximate within $1-1/e+\epsilon$ for any $\epsilon > 0$ (Kempe, Kleinberg, and Tardos 2003; 2005). One key difficulty is that, even when the actions are fixed, it is #P-hard to compute the objective function (Chen, Wang, and Wang 2010), a problem which is essentially equivalent to the #P-complete s-t reliability problem (Valiant 1979).

In this paper, we focus on the special case of stochastic network design in *tree-structured* networks, which can represent spreading phenomena in a hierarchical organization or a river network. Our motivating application is a conservation planning problem involving the removal of barriers in a river network that limit the accessibility of fish to their native habitats (O'Hanley and Tomberlin 2005). In a tree-structured network, a key aspect of the general problem becomes much easier: one can compute the probability of spreading to any vertex, and thus the objective function, in linear time. However, a simple reduction from the knapsack problem shows that the problem remains NP-hard. The overall message is positive, though: we prove the existence of an FPTAS for the tree-structured problem, which places it among the easiest of NP-hard problems, and clearly beats the $1-1/e+\epsilon$ inapproximability result for the general case.

A starting point for our work is that of O'Hanley and Tomberlin (2005), who formulate the barrier-removal problem and present a pseudo-polynomial dynamic programming algorithm to solve it. Their algorithm assumes that all budgets and costs are integers and its running time is quadratic in the budget size, which is a clear limitation. In practice, to use the algorithm with large budgets one needs to scale the budget and costs to much smaller values and round them to integers; examples show that no optimality guarantees can be made for such a cost-based rounding approach.

The main contribution of this work is to develop an alternate dynamic programming approach that is amenable to rounding and to prove that the resulting algorithm—*rounded dynamic programming* (RDP)—is an FPTAS for tree-structured stochastic network design. Specifically, in the worst case, the algorithm computes a $(1-\epsilon)$-optimal solution in time $O(n^2/\epsilon^2)$, where n is the number of vertices. Experiments show that the algorithm actually computes near-optimal policies much faster: most problem in-

stances on a large river network were solved within 1 minute and produced solutions that were within 10% of optimal.

In Section 2, we formulate the barrier-removal and stochastic network design problems. We introduce the RDP algorithm in Section 3, and prove that it is an FPTAS in Section 4. Section 5 discusses implementation issues. Experiments in Section 6 show the benefits of the algorithm.

Problem Formulation

Our work on tree-structured stochastic network design is motivated by the *fish barrier removal problem*, an important ecological sustainability problem proposed by O'Hanley and Tomberlin (2005) to combat dramatic population declines of wild fish over the past two centuries due to the presence of river barriers. Dams and other barriers such as culvert, floodgates and weirs harm populations by preventing fish from accessing or moving between parts of their historical habitat. The way to address this problem is to retrofit existing barriers or to replace them by new instream structures that make it easier for fish to pass. O'Hanley and Tomberlin formulated the optimization problem of selecting a subset of barriers to repair or remove so as to maximize the available *upstream* habitat for anadromous fish—species such as salmon that live part of the year in oceans but travel up rivers and streams to spawn.

In the barrier removal problem, fish swim upstream in a tree $\mathcal{T} = (V, E)$ from the root or source vertex s (the ocean) to access upstream habitat. Edges represents stream segments and vertices represent either barriers ($u \in B \subseteq V$) or junctions of multiple streams ($u \in V \setminus B$). Initially, fish are able to pass barrier $u \in B$ with probability p_u, the *passability* of vertex u. A finite set of possible repair actions A_u is associated with barrier u. Action $a \in A_u$ has cost $c_{u,a}$ and, if taken, increases the passability to $p_{u|a}$. We assume that A_u includes a default zero-cost "noop" action a_0 such that $p_{u|a_0} := p_u$. A policy π selects an action $\pi(u)$—either a repair or noop—for each barrier. We write $p_{u|\pi} := p_{u|\pi(u)}$ for the passability of vertex u under policy π.

Given a fixed policy π, the *accessibility* $P^+_{u|\pi}$ of node u is the probability that a fish swimming upstream from s can pass all the barriers up to and including u. It is useful to define this notion slightly more generally. Whenever v is an ancestor of u, define the v-u *accessibility* $P^+_{v\to u|\pi}$ as the product of the barrier probabilities on the unique path from v to u, including the probabilities $p_{v|\pi}$ and $p_{u|\pi}$ of the barriers at the path's endpoints. We now have $P^+_{u|\pi} := P^+_{s\to u|\pi}$.

Upon passing the barrier at u, fish are able to use the amount of habitat r_u between u and its nearest upstream neighbors (i.e., children in $\mathcal{T}$). We refer to r_u as the *reward* of vertex u. The objective to be maximized is the total accessibility-weighted reward:

$$z(\pi) = \sum_{u \in V} P^+_{u|\pi} \cdot r_u. \tag{1}$$

We refer to $z(\pi)$ either as the *value* or the *expected reward* (for reasons detailed below). The goal is to find a policy that maximizes $z(\pi)$ subject to a budget b on the total cost of all actions, that is, an optimal policy π^* satisfies $\pi^* \in \operatorname{argmax}_{\{\pi | c(\pi) \le b\}} z(\pi)$, where $c(\pi)$ is the total cost of actions taken by π.

Our algorithms will be based on a simple recurrence for the value of a policy. Let $\mathcal{T}_u$ be the subtree rooted at u and let $z_u(\pi) = \sum_{v \in \mathcal{T}_u} P^+_{u\to v|\pi} r_u$ denote the expected reward of policy π when applied to $\mathcal{T}_u$. For the barrier removal problem, this corresponds to the case where all fish start at u and swim upstream. The value $z_u(\pi)$ can be calculated by

$$z_u(\pi) = p_{u|\pi}\Big(r_u + \sum_{v \in \mathrm{Ch}(u)} z_v(\pi)\Big), \tag{2}$$

where $\mathrm{Ch}(u)$ is the set of children of u.

Stochastic Network Design The reader can easily verify that the objective $z(\pi)$ in Eq. (1) is equivalent to the expected reward in the following random experiment. For each barrier $u \in B$, an independent coin is flipped to determine whether u will remain in the graph (with probability $p_{u|\pi}$) or fail and be removed (with probability $1 - p_{u|\pi}$). In the resulting subgraph $\mathcal{T}'$, a reward of r_u is collected for each node u that is reachable from the source s. Thus, the fish barrier removal problem is equivalent to a stochastic network design problem where actions are taken to reduce failure probabilities at nodes in a directed tree, and a reward is collected for each node that remains reachable from the root. By the "live-edge" characterization of the Independent Cascade Model (Kempe, Kleinberg, and Tardos 2003), the problem is also equivalent to maximizing the spread of an Independent Cascade from the source s (see also Sheldon et al., 2010) by taking actions to increase the probability the cascade propagates at nodes in the tree.

Rounded Dynamic Programming

For the knapsack problem, a dynamic programming algorithm can be used to compute the optimal packing strategy (Vazirani 2003). In this section we describe how similar ideas can be applied to the barrier removal problem.

A key aspect of our algorithm will be to determine how to partition the total value $z_u(\pi)$ obtained from $\mathcal{T}_u$ among the children of u. To simplify this aspect of the algorithm and proofs, we assume without loss of generality the following.

Assumption 1. *Each node u in $\mathcal{T}$ has at most two children.*

Any problem instance can be converted to this form by replacing nodes u with more than two children by a sequence of nodes with exactly two children to achieve the same overall branching factor, where the newly added vertices are set to be fully passable and the original reward r_u is split equally among u and the newly added vertices. In this way, a policy for the modified tree can be mapped to a unique policy for the original tree with the same expected reward.

The basic idea of the DP algorithm is to write a recurrence for the quantity $C_u(z)$, which is the minimum cost needed to produce expected reward of exactly z for the subtree $\mathcal{T}_u$. This can be written as:

$$C_u(z) = \min_{a \in A_u} C_{u,a}(z),$$

where $C_{u,a}(z)$ is the minimum cost needed to produce value z from $\mathcal{T}_u$ given that action a is taken. To compute $C_{u,a}(z)$,

one must reason about how to split the overall reward z among u and its children. The only difficult case is when u has two children v and w, when we write:

$$C_{u,a}(z) = \min_{0 \le d \le z'_{u,a}} c_{u,a} + C_v(d) + C_w(z'_{u,a} - d) \quad (3)$$

Here, $z'_{u,a} = \frac{z}{p_{u|a}} - r_u$ encodes the total value that the children are required to contribute in order to get value z at their parent. When u has a single child v, the expression simplifies to $C_{u,a}(z) = c_{u,a} + C_v(z'_{u,a})$, and when u is a leaf we have the base case: $C_{u,a}(z) = c_{u,a}$ if $p_{u|a} r_u = z$, and $C_{u,a}(z) = +\infty$ otherwise, meaning that it is impossible to obtain value z. The optimal expected reward given the budget limit b is the maximum value that we can get at the subtree $\mathcal{T}_s$ with the cost at most b, namely $\max\{z : C_s(z) \le b\}$.

Unfortunately, this recurrence does not lead to an efficient algorithm because the z values are real numbers, and the number of z values that must be evaluated increases exponentially as the recurrence approaches the source. In the design of the FPTAS of the knapsack problem, this is addressed by scaling down the profits by some factor K and then rounding the scaled profits into integral values. In this way, the intermediate sum of profits only takes a small number of integral values and only the least costs corresponding to these values must be evaluated. Hence, the DP algorithm takes polynomial time to produce a near optimal solution. However, simply rounding the rewards of vertices does not address the difficulty in the barrier removal problem, because the accessibility-weighted rewards can have any real value even when the rewards are integral.

We propose a novel way to discretize the continuous space of the expected reward of each subtree. To do this, we introduce the concept of rounded expected rewards of subtrees, which are calculated by the following recurrence:

$$\hat{z}_u(\pi) = K_u \left\lfloor \frac{p_{u|\pi}\left(r_u + \sum_{v \in \mathrm{Ch}(u)} \hat{z}_v(\pi)\right)}{K_u} \right\rfloor \quad (4)$$

where K_u is a distinct scaling factor associated with subtree $\mathcal{T}_u$. Recurrence (4) is very similar to the recurrence (2), except that the value is first divided by a scaling factor K_u, rounded into an integer and then enlarged back by the same factor K_u. This effort discretizes the expected reward of each $\mathcal{T}_u$ into $\{0K_u, 1K_u, .., \ell_u K_u\}$ where ℓ_u is an integer with value being $\left\lceil \frac{\sum_{v \in \mathcal{T}_u} r_v}{K_u} \right\rceil$ so that $\ell_u K_u$ is an upper bound of $\hat{z}_u$. Our dynamic programming algorithm called rounded dynamic programming (RDP) looks for an optimal policy π' such that $\pi' = \max_{\{\pi : c(\pi) \le b\}} \hat{z}(\pi)$. Specifically, the algorithm finds the minimal cost needed to get each of the *discretized* values $\hat{z}_u$ using the following recurrence in place of (3):

$$C_{u,a}(\hat{z}) = \min_{(\hat{z}_v, \hat{z}_w) \in U_{u,a}(z)} c_{u,a} + C_v(\hat{z}_v) + C_w(\hat{z}_w) \quad (5)$$

where the set $U_{u,a}(\hat{z})$ defined by

$$\left\{ (\hat{z}_v, \hat{z}_w) \;\middle|\; \begin{array}{l} \hat{z} = K_u \left\lfloor \frac{p_{u|a}(r_u + \hat{z}_v + \hat{z}_w)}{K_u} \right\rfloor, \\ \hat{z}_v \in \{0, ..., K_v \ell_v\}, \hat{z}_w \in \{0, ..., K_w \ell_w\} \end{array} \right\} \quad (6)$$

contains, for action a, all pairs of integral values that subtrees $\mathcal{T}_v$ and $\mathcal{T}_w$ can take to get the rounded expected reward $\hat{z}$ at $\mathcal{T}_u$ by (4). Similarly, the optimal rounded expected reward of the whole river network is $\max\{\hat{z} : C_s(\hat{z}) \le b\}$ and the policy that achieves that value is the one we search for.

Theoretical Analysis of the Algorithm

In this section, we present our main result.

Theorem 1. *The Rounded Dynamic Programming (RDP) algorithm is an FPTAS. Specifically, let OPT be the value of the optimal policy. By assigning the scaling factors $\{K_v\}$ in a certain (described below), the RDP algorithm computes a policy with value at least $(1-\epsilon)$OPT and runs in time $O(\frac{n^2}{\epsilon^2})$ in the worst case.*

We prove Theorem 1 by first showing the approximation guarantee and then analyzing the running time of RDP.

Approximation Guarantee

Let π^* be the optimal policy and let π' be the policy returned by RDP. We wish to bound the value loss $z(\pi^*) - z(\pi')$.

The idea of the proof is to first bound the difference between the true objective value $z(\pi)$ and the RDP objective value $\hat{z}(\pi)$ for an arbitrary policy π, which can be done by analyzing the error incurred by the rounding operations in the recurrence of Eq. (4). By showing that the rounded objective function $\hat{z}$ is uniformly close to z for all policies π, it is straightforward to show that optimizing with respect to $\hat{z}$ provides a nearly-optimal policy with respect to z.

To analyze the error introduced by rounding, fix a policy π and let $\Delta_u(\pi) \in [0, 1)$ be the fractional part of the quantity that is rounded in Eq. (4), so that $K_u \Delta_u(\pi)$ is the total loss due to rounding when computing the recurrence for node u. Then it is straightforward to show that the total error is equal to the sum of the rounding errors at each node u weighted by the accesibility of the parent of node u under policy π.

Lemma 1. *For any policy π, the difference between the original and rounded objective functions is*

$$z(\pi) - \hat{z}(\pi) = \sum_{u \in V} P^+_{pa(u)|\pi} K_u \Delta_u(\pi) \quad (7)$$

where $pa(u)$ denotes the parent of u.

To make the above notation consistent for the case $u = s$, assume the existence of a distinguished vertex $s_0 = \mathrm{pa}(s) \notin V$ such that $r_{s_o} = 0$ and $P^+_{s_0|\pi} = 1$ for all π.

Now, to bound the optimality gap $z(\pi^*) - z(\pi')$, note that $z(\pi') \ge \hat{z}(\pi') \ge \hat{z}(\pi^*)$, where the first inequality holds because the rounded policy value always underestimates the true policy value, and the second inequality holds because π' is optimal with respect to $\hat{z}$. Thus we have

$$z(\pi^*) - z(\pi') \le z(\pi^*) - \hat{z}(\pi^*), \quad (8)$$

so it suffices to bound the gap between the original and rounded objective on the optimal policy π^* using Lemma 1.

Lemma 2. *The RDP policy π' has value at least $(1-\epsilon)$OPT if the following condition on the scaling factors $\{K_u\}$ holds:*

$$\sum_{u \in V} P^+_{pa(u)|\pi^*} K_u \le \epsilon z(\pi^*). \quad (9)$$

Proof. The left side of Eq. (9) is an upper bound on $z(\pi^*) - z(\pi')$, which can be seen by substituting Eq. (7) into Eq. (8) and then applying the bound $\Delta_u(\pi^*) \leq 1$. □

Lemma 2 is useful as a generic condition on the scaling factors $\{K_u\}$ for obtaining a $(1-\epsilon)$-optimal policy. There are different ways of setting the values so Eq. (9) is satified, and the particular choice will affect the running time of the algorithm. Indeed, note that a larger value of K_u leads to a coarser discretization of the value space at node u, and thus the RDP algorithm will take less time to evaluate the recurrence for all discretized values. Thus, in practice we would like to set the scaling factors as large as possible while still satisfying Eq. (9). We first present a particular way of setting the values that is rather coarse but lets us prove both the approximation guarantee and the worst-case running-time bound. In Section 5 we discuss practical improvements.

Lemma 3. *Let the scaling factor $K_u = K = \epsilon m/2$ for all nodes u, where $m = \min_{u \in V} r_u$ is the minimum reward of any node. Then the policy π' returned by RDP is a $(1-\epsilon)$-optimal policy.*

Proof. Rewrite the left side of Eq. (9) as

$$\sum_{u \in V \cup s_0 \setminus L} P^+_{u|\pi^*} \Big(\sum_{v \in \text{Ch}(u)} K_v \Big)$$

where L are the leaves of $\mathcal{T}$. By setting $K_u = \epsilon m/2$ and recalling that u has at most two children, we have $\sum_{v \in \text{Ch}(u)} K_v \leq \epsilon m \leq \epsilon r_u$, so the sum is bounded by

$$\sum_{u \in V \setminus L} P^+_{u|\pi^*} \epsilon r_u \leq \epsilon \sum_{v \in V} P^+_{u|\pi^*} r_u = \epsilon z(\pi^*)$$

□

From the proof it is clear that we can set the scaling factors larger while still proving the bound.

Lemma 4. *Let the scaling factor $K_u = \epsilon r_{pa(u)}/\deg(pa(u))$ where $\deg(v)$ is the number of children of v. Then the policy π' returned by RDP is a $(1-\epsilon)$-optimal policy.*

In practice, the scaling factors of Lemma 4 will always lead to faster running-times than those in Lemma 3. However, they do not improve the worst-case asymptotic bounds.

Runtime Analysis

In this subsection, we will prove that the running time of the RDP algorithm is $O(\frac{n^2}{\epsilon^2})$ in the worst case if the scaling factors K_u are assigned as in Lemma 3. First, it is reasonable to assume that the rewards are constant with respect to n.

Assumption 2. *There are universal constants m and M such that $m \leq r_u \leq M$ for all $u \in V$.*

For example, in our application the rewards are the lengths of accessible stream segments; these should not vary with the size of the stream network being modeled.

First, let's examine the time spent computing the recurrence for a single node u. The value of $C_{u,a}(\hat{z})$ is computed for all $a \in A_u$ and ℓ_u different value of $\hat{z}$. If u is a leaf or has only one child (denote these nodes by V_1), then it takes constant time to compute $C_{u,a}(\hat{z})$. Otherwise u has two children (denote these nodes by V_2), and the RDP recurrence (5) iterates over each pair in $U_{u,a}(\hat{z})$ to determine how to split the value $\hat{z}$. Thus, if we assume $|A_u|$ is constant, the time spent at a single level of the recurrence is

$$O(\ell_u),\ u \in V_1, \qquad O(\ell_u \gamma_u),\ u \in V_2,$$

where γ_u is any upper bound on $|U_{u,a}(\cdot)|$. We will argue the following two facts below.

Fact 1. *For a node u with two children v and w, the number of pairs $|U_{u,a}(\hat{z})|$ evaluated by the RDP recurrence is $O(\ell'_u)$ where $\ell'_u = \min\{\ell_v, \ell_w\}$.*

Fact 2. *Under the assumptions made so far, the number of discretized values ℓ_u at node u is equal to $O(n_u/\epsilon)$, where n_u is the number of nodes in $\mathcal{T}_u$.*

Using these two facts we can rewrite the running times above as $O(n_u/\epsilon)$ for $u \in V_1$ and as $O(n_u k_u/\epsilon^2)$ for $u \in V_2$ where $k_u = \min\{n_v : v \in \text{Ch}(u)\}$ is size of the smaller of u's subtrees. Now, if we define $T(n)$ as the maximum running time of RDP for any tree of size n, the following recurrence is satisfied for $n > 1$:

$$T(n) \leq \max_{0 \leq k \leq \lfloor n/2 \rfloor} \Big(O(n(k+1)/\epsilon^2) + T(k) + T(n-k-1) \Big),$$

with the base case of $T(1) = O(1)$. The case $k > 1$ captures a root node with two subtrees of size k and $n-k-1$, and the case $k = 0$ captures a root node with a single child, so the subtree size is $n-1$. It can be shown inductively that $T(n)$ is $O(n^2/\epsilon^2)$ (details are given in the appendix), which proves the main result. □

Proof of Facts 1 and 2. Fact 2 is a direct consequence of the definitions we have made. Let $\text{UB}_v = \sum_{u \in \mathcal{T}_v} r_v$. Then

$$\ell_u = \left\lceil \frac{\text{UB}_u}{K_u} \right\rceil \leq \left\lceil \frac{n_u M}{K_u} \right\rceil = O\Big(\frac{n_u}{\epsilon}\Big).$$

For Fact 1, we see by (6) that any pair $(\hat{z}_v, \hat{z}_w)$ in $U_{u,a}(\hat{z})$ satisfies

$$\frac{\hat{z}}{K_u} \leq \frac{p_{u|a}(r_u + \hat{z}_v + \hat{z}_w)}{K_u} < \frac{\hat{z}}{K_u} + 1$$

Without loss of generality, assume that $\ell'_u = \ell_v \leq \ell_w$. For each $\hat{z}_v$ in $\{0, ..., \ell_v K_v\}$, $\hat{z}_w$ satisfies

$$\frac{\hat{z}}{p_{u|a}} - r_u - \hat{z}_v \leq \hat{z}_w < \frac{\hat{z} + K_u}{p_{u|a}} - r_u - \hat{z}_v$$

The number of different integral values in this range is at most $\left\lfloor \frac{K_u}{p_{u|a} K_w} \right\rfloor + 1$ which is $O(1)$ and thus the total number of pairs is $O(\ell_v)$.

Implementation

We now discuss techniques to improve runtime in practice.

Setting the Scaling Factors Recall from Lemma 2 that the scaling factors must satisfy $\sum_{u \in V} P^+_{\text{pa}(u)|\pi^*} K_u \leq \epsilon z(\pi^*)$ to give the approximation guarantee. To prove the FPTAS, it suffices to set $K_u = \epsilon m/2$ for all u, but in Lemma 4 we gave a different way of setting K_u on a per-node basis that provides the same guarantees and is always faster. However, *in practice*, we observe that setting the scaling factors to a constant value α that is larger than $\epsilon m/2$ still finds near-optimal policies and runs much faster.

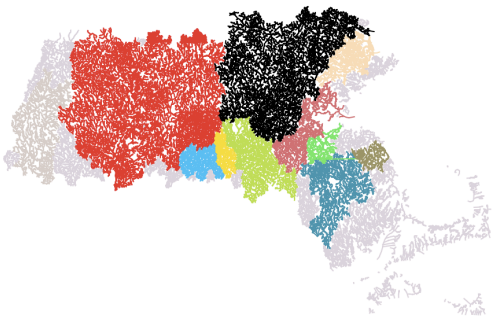


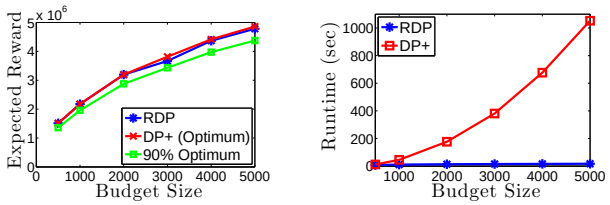
Figure 1: River networks in Massachusetts

We briefly explain this apparent contradiction. First, note that we only wish to bound the sum $\sum_{u \in V} P^{+}_{\text{pa}(u)|\pi^*} K_u$, so we can trade off the magnitude of the individual terms to achieve the same overall bound. For the individual terms, we want to keep K_u small whenever $P^{+}_{\text{pa}(u)|\pi^*}$ is large, which makes sense intuitively: we want a finer resolution at node u when it is more accessible under the optimal policy. Finally, by the reasoning of the previous section, the dominant component of the running time can be approximated as $\sum_{u \in V_2} \ell_u \ell'_u \approx \sum_{u \in V_2} (\text{UB}_u / K_u)^2$. Here we assume $\ell_u \approx \ell'_u$. From this it is clear that *nodes with very small values of K_u have a disproportionately large effect on the running time*. That said, the value of K_u at those nodes may have a minor effect on the approximation bound, especially if $P^{+}_{u|\pi^*}$ is small. Overall, an obvious way to reduce the running time is simply to increase K_u at those nodes to be in line with the values elsewhere, knowing there is some "play" in the approximation bound. This naturally leads back to a constant setting of K_u to some value higher than $\epsilon m/2$. By doing this we *are relaxing the approximation guarantee* to achieve better empirical performance.

Detecting Infeasible Policies The algorithm can be made faster by exploiting the budget limit b. The idea is to ignore pairs $(\hat{z}_v, \hat{z}_w)$ in the RDP recurrence when the cost to obtain either $\hat{z}_v$ or $\hat{z}_w$ in the subtree already exceeds b, because these will lead to infeasible policies. This technique speeds up the algorithm especially when the budget limit is small. In our experiments, we observe that even for the subtrees at intermediate depths, a large percentage of values are pruned.

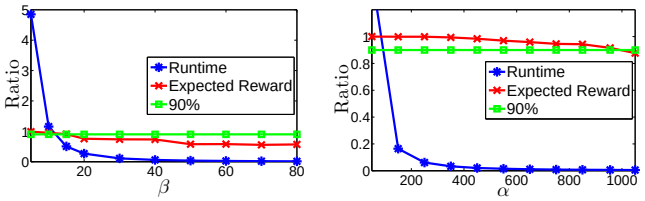
Experiments

In our experiments, we use data from the CAPS project (McGarigal et al. 2011) for the Connecticut River watershed in Massachusetts (shown in red in Fig. 1), which has 18550 vertices including 596 dams and 7566 crossings that include different types of small barriers. We assigned passage probabilities to dams and road-stream crossings based on techniques developed in the CAPS project. For dams, the structural height of the dam is a proxy for passability, which maps through a logistic function to a probability value. A subset of road-stream crossings were directly assessed by a field protocol; the remaining crossings were assigned passage scores based on a fitted predictive model. Passage scores were then transformed to probabilities, resulting in a typical range of $[0.7, 1.0]$ for road-crossings and $[0, 0.15]$ for dams.



(a) Optimality w.r.t. budget (b) Runtime w.r.t. budget

Figure 2: Solution quality and runtime for different budgets



(a) Time & value for method (1) (b) Time & value for method (2)

Figure 3: Impact of different methods to set K_u

Using this dataset, we compared our RDP algorithm with the dynamic programming algorithm, called DP+, of O'Hanley and Tomberlin (2005), which assumes that the costs of actions and the budget b are integral values. DP+ is optimal under this assumption. However, unlike RDP, DP+ is not scalable to large action costs. Therefore, to perform the evaluation we used relatively small integral costs.

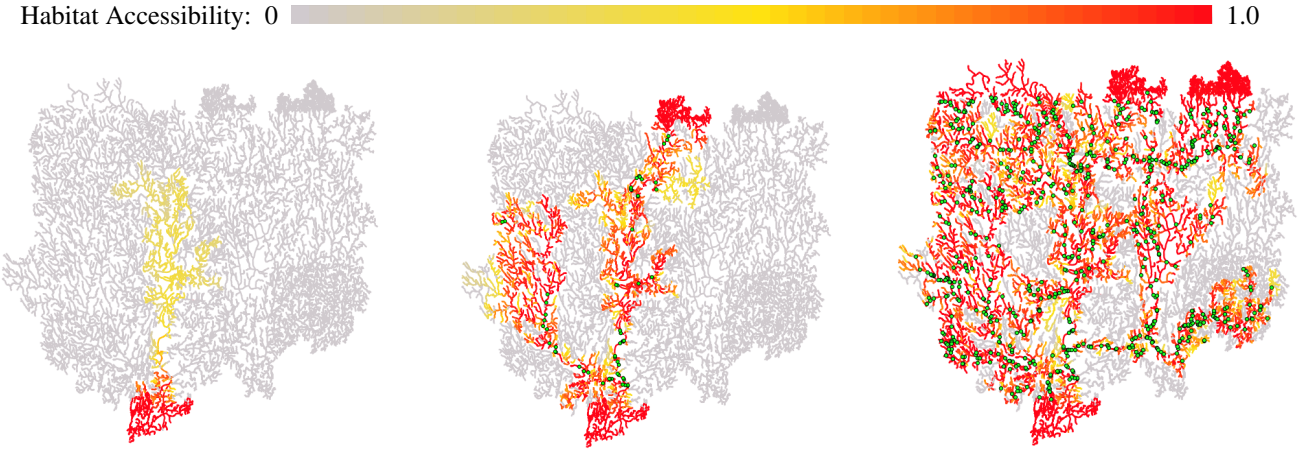
For road-crossings, most of the probabilities are close to 1 to start with, and relatively cheap actions can be taken to clear out the crossing completely. For example, we use $A_u = \{a_1\}$ with $(p_{u|a_1} = 1.0, c_{u|a_1} = 20)$. In contrast, it is relatively difficult and expensive to remove dams completely, so multiple strategies must be considered to improve the passability of dams. For example, we may have $A_u = \{a_1, a_2, a_3\}$ with $(p_{u|a_1} = 0.2, c_{u|a_1} = 20)$, $(p_{u|a_2} = 0.5, c_{u|a_2} = 40)$ and $(p_{u|a_3} = 1.0, c_{u|a_3} = 100)$.

Experimental Results

Approximation Quality Fig. 2(a) shows the expected reward of the computed policy for different budgets, as well as OPT (the value of the optimal policy) and 90%OPT. In these experiments, we set all K_u to be a constant $\alpha = 450$. We see that the actual expected reward of the computed policy is very close to the optimal value and guaranteed to be greater than the 90%OPT .

Runtime Fig. 2(b) shows the computation time of our algorithm (with the optimization of detecting infeasible policies) compared with DP+ over a range of budget sizes. We see that RDP runs much faster than the optimal algorithm DP+. Moreover, the runtime of DP+ increases quadratically with the budget size while RDP's runtime remains essentially constant, even for very large budget sizes. For example, when the budget size is 5000, DP+ takes about 20 minutes while RDP takes only 20 seconds.

Fig. 2(a) and 2(b) together show that by setting $K_u = 450$, RDP runs much faster than DP and produces a near-optimal policy (within 90%).



(a) budget = 0, $z = 5.4 \times 10^5$ (b) budget = 1000, $z = 2.2 \times 10^6$ (c) budget = 10000, $z = 5.8 \times 10^6$

Figure 4: Visualization of several barrier removal policies

Different Settings of K_u We compared the two different ways of setting K_u as discussed in Section 5: (1) setting $K_u = \beta K'_u$ where K'_u are the values specified in Lemma 4, and (2) $K_u = \alpha$. We used the constants α and β to relax the optimality guarantee and study the effect on runtime/quality (Fig. 3). The runtime ratio is the ratio of RDP's runtime to DP+'s runtime. The value ratio is the ratio of the expected reward of the computed policy obtained by RDP to the optimal expected reward obtained by DP+. We found that for method (1) that before the expected reward becomes worse than 90% of optimal, RDP takes more time than DP+. For method (2), as α increases up to 950, the quality of the policy remains above 90%, but the runtime ratio is less than 0.1. This matches the intuitive explanation provided in Section 5.

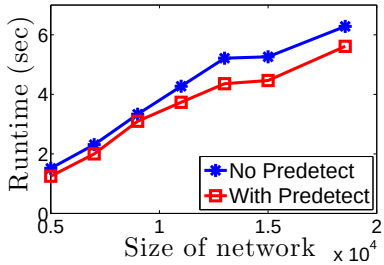


Figure 5: RDP's runtime achieving 90% optimality

Runtime Curve Theoretically, we proved that in the worse case the complexity of RDP is $O(\frac{n^2}{\epsilon^2})$. However, in practice, we can get a better computation time by using the techniques described in Section 5. By applying the technique of detecting infeasible policies, runtime was reduced by at least 20% over a range of budget sizes. When the budget size is small, many computations can be pruned and runtime is reduced by up to 55%. Moreover, as just shown, the value of K_u can be selected in a better way to further reduce the computation time dramatically. Fig. 5 shows the minimum time needed to produce a 90% optimal policy as a function of network size. Subnetworks of different sizes were extracted from the original network for these experiments. The minimum time is obtained by applying method (2) and choosing the largest constant α that produces the desired quality. Surprisingly, both curves—with and without detecting the infeasible policies—are nearly linear except for some small fluctuation in the middle.

Visualizing the resulting policies To give some sense of how the policies improve the ability of fish to access their habitats, Fig. 4 illustrates the accessibility of each stream segment by a distinct color from the color bar at the top. Barriers that are repaired by some action are designated by green circles, regardless of the specific repair action. The budget in (c) is 10 times larger relative to (b), leading to a substantial increase in the number of repaired crossings as they are much cheaper to repair compared to dams, and resulting in significantly better overall accessibility.

Conclusion

We developed a fast approximation algorithm that uses rounded dynamic programming (RDP) for solving the fish barrier removal problem. The algorithm can be used to compute near optimal policies for tree-structured stochastic network design problems. We analyzed the characteristics of the algorithm and proved that it is an FPTAS. When applied to the Connecticut River watershed in Massachusetts, RDP can produce near-optimal results within a small fraction of the runtime of the benchmark optimal algorithm DP+. Moreover, RDP can easily scale up to larger action costs and budget sizes, while DP+ is not scalable. These results provide an effective planning tool for restoring accessibility of native fish habitat in large river networks.

A promising line of future work is to better exploit the interplay between running time and solution quality in RDP (cf. Section 5). For example, one could potentially derive better upper bounds UB_u using properties of the problem instance, or use an iterative approach that first computes an optimal policy at a coarse discretization, and then makes the discretization finer at nodes with high accessibility.

Acknowledgments

We thank Brad Compton and Scott Jackson for providing the river network and barrier data and helping with the interpretation of the data.

Appendix

Fact 3. *$T(n)$ is $O(n^2/\epsilon^2)$.*

Proof. We drop the constant factor $1/\epsilon^2$, which only serves to scale the final running time by $1/\epsilon^2$. Assume that

$$T(n) \leq \max_{0 \leq k \leq \lfloor n/2 \rfloor} \Big(cn(k+1) + T(k) + T(n-k-1) \Big)$$

and $T(1) = O(1)$. We will show that $T(n) \leq 2cn^2$ for all $n \geq 1$. Assume inductively that $T(m) \leq 2cm^2$ for all $m < n$. Then we get

$$\begin{aligned} T(n) &\leq \max_{0 \leq k \leq \lfloor n/2 \rfloor} \Big(cn(k+1) + 2ck^2 + 2c(n-k-1)^2 \Big) \\ &= \max_{0 \leq k \leq \lfloor n/2 \rfloor} \Big(2cn^2 + c(4k^2 - 3nk + 4k - 3n + 2) \Big) \\ &= 2cn^2 + c \max_{0 \leq k \leq \lfloor n/2 \rfloor} f_n(k) \end{aligned}$$

The function $f_n(k) = 4k^2 - 3nk + 4k - 3n + 2$ is convex and hence

$$\max_{0 \leq k \leq \lfloor n/2 \rfloor} f_n(k) \leq \max_{k \in [0, n/2]} f_n(k) = \max\{f_n(0), f_n(n/2)\}$$

But

$$f_n(0) = -3n + 2, \quad f_n(n/2) = -\frac{1}{2}n^2 - n + 2.$$

Each of these is non-positive when $n \geq 1$.
Thus $T(n) \leq 2cn^2$, and the result is proved. □

References

Chen, W.; Wang, C.; and Wang, Y. 2010. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1029–1038.

Golovin, D.; Krause, A.; Gardner, B.; Converse, S. J.; and Morey, S. 2011. Dynamic resource allocation in conservation planning. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, 1331–1336.

Kempe, D.; Kleinberg, J.; and Tardos, É. 2003. Maximizing the spread of influence through a social network. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 137–146.

Kempe, D.; Kleinberg, J.; and Tardos, É. 2005. Influential nodes in a diffusion model for social networks. In *Automata, languages and programming*. Springer. 1127–1138.

Kumar, A.; Wu, X.; and Zilberstein, S. 2012. Lagrangian relaxation techniques for scalable spatial conservation planning. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, 309–315.

McGarigal, K.; Compton, B. W.; Jackson, S. D.; Plunkett, E.; Rolih, K.; Portante, T.; and Ene, E. 2011. Conservation assessment and prioritization system (CAPS). Technical report, Department of Environmental Conservation, Univ. of Massachusetts Amherst.

O'Hanley, J. R., and Tomberlin, D. 2005. Optimizing the removal of small fish passage barriers. *Environmental Modeling and Assessment* 10(2):85–98.

Sheldon, D.; Dilkina, B.; Elmachtoub, A.; Finseth, R.; Sabharwal, A.; Conrad, J.; Gomes, C.; Shmoys, D.; Allen, W.; Amundsen, O.; and Vaughan, W. 2010. Maximizing the spread of cascades using network design. In *Proceedings of the 26th Conference on Uncertainty in Artificial Intelligence*, 517–526.

Valiant, L. G. 1979. The complexity of enumeration and reliability problems. *SIAM Journal on Computing* 8(2):410–421.

Vazirani, V. 2003. *Approximation Algorithms*. Springer.

Wu, X.; Sheldon, D.; and Zilberstein, S. 2013. Stochastic network design for river networks. In *NIPS Workshop on Machine Learning for Sustainability*.