European Journal of Operational Research 263 (2017) 664–678

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Innovative Applications of O.R.

A comparison of Monte Carlo tree search and rolling horizon optimization for large-scale dynamic resource allocation problems

Dimitris Bertsimas[a], J. Daniel Griffith[b], Vishal Gupta[c], Mykel J. Kochenderfer[d], Velibor V. Mišić[e,*]

[a] Sloan School of Management and Operations Research Center, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139, USA
[b] Lincoln Laboratory, Massachusetts Institute of Technology, 244 Wood Street, Lexington, MA 02420, USA
[c] Department of Data Sciences and Operations, Marshall School of Business, University of Southern California, 3670 Trousdale Parkway, Los Angeles, CA 90089, USA
[d] Department of Aeronautics and Astronautics, Stanford University, 496 Lomita Mall, Stanford, CA 94305, USA
[e] Anderson School of Management, University of California, Los Angeles, 110 Westwood Plaza, Los Angeles, CA 90024, USA

ARTICLE INFO

Article history:
Received 7 June 2016
Accepted 15 May 2017
Available online 19 May 2017

Keywords:
Dynamic resource allocation
Monte Carlo tree search
Rolling horizon optimization
Wildfire management
Queueing control

ABSTRACT

Dynamic resource allocation (DRA) problems constitute an important class of dynamic stochastic optimization problems that arise in many real-world applications. DRA problems are notoriously difficult to solve since they combine stochastic dynamics with intractably large state and action spaces. Although the artificial intelligence and operations research communities have independently proposed two successful frameworks for solving such problems—Monte Carlo tree search (MCTS) and rolling horizon optimization (RHO), respectively—the relative merits of these two approaches are not well understood. In this paper, we adapt MCTS and RHO to two problems – a problem inspired by tactical wildfire management and a classical problem involving the control of queueing networks – and undertake an extensive computational study comparing the two methods on large scale instances of both problems in terms of both the state and the action spaces. Both methods are able to greatly improve on a baseline, problem-specific heuristic. On smaller instances, the MCTS and RHO approaches perform comparably, but RHO outperforms MCTS as the size of the problem increases for a fixed computational budget.

1. Introduction

Dynamic resource allocation (DRA) problems are problems where one must assign resources to tasks over some finite time horizon. Many important real-world problems can be cast as DRA problems, including applications in air traffic control (Bertsimas & Stock Patterson, 1998), scheduling (Bertsimas, Gupta, & Lulli, 2014) and logistics, transportation and fulfillment (Acimovic & Graves, 2012). DRA problems are notoriously difficult to solve exactly since they typically exhibit stochasticity and extremely large state and action spaces. The artificial intelligence (AI) and operations research (OR) communities have sought more sophisticated techniques for addressing DRA and other dynamic stochastic optimization problems.

Within the AI community, one approach for dynamic stochastic optimization problems that has received increasing attention in the last 15 years is Monte Carlo tree search (MCTS) (Browne et al., 2012; Coulom, 2007). In any dynamic stochastic optimization problem, one can represent the possible trajectories of the system—the state and the action taken at each decision epoch—as a tree, where the root represents the initial state. In MCTS, one iteratively builds an approximation to this tree and uses it to inform the choice of action. MCTS's effectiveness stems from two key features: (1) bandit upper confidence bounds (see Auer, Cesa-Bianchi, & Fischer, 2002; Kocsis & Szepesvári, 2006) can be used to balance exploration and exploitation in learning, and (2) application-specific heuristics and knowledge can be used to customize the base algorithm (Browne et al., 2012). Moreover, MCTS can easily be tailored to a variety of problems. Indeed, the only prerequisite for implementing MCTS is a generative model that, given a state and an action at a given decision epoch, generates a new state for the next epoch. This flexibility makes MCTS particularly attractive as a general purpose methodology.

Most importantly, MCTS has been extremely successful in a number of applications, particularly in designing expert computer

* Corresponding author.
E-mail addresses: dbertsim@mit.edu (D. Bertsimas), dan.griffith@ll.mit.edu (J.D. Griffith), guptavis@usc.edu (V. Gupta), mykel@stanford.edu (M.J. Kochenderfer), velibor.misic@anderson.ucla.edu (V.V. Mišić).

http://dx.doi.org/10.1016/j.ejor.2017.05.032
0377-2217/

players for difficult games such as Go (Enzenberger, Muller, Arneson, Segal, 2010; Gelly & Silver, 2011), Hex (Arneson, Hayward, & Henderson, 2010), Kriegspiel (Ciancarini & Favini, 2010), and Poker (Rubin & Watson, 2011). Although MCTS is one of the top-performing algorithms for this class of games, games like Go and Hex are qualitatively different from DRAs: unlike typical DRA problems, the state of these games does not evolve stochastically, and the size of the feasible action space is often much smaller. For example, in the Go instances of Gelly and Silver (2011), the action branching factor is at most 81, whereas in one DRA instance we consider, a typical branching factor is approximately 230 million (cf. Eq. (12)). While MCTS has been applied to probabilistic problems (Eyerich, Keller, & Helmert, 2010) and problems with large action spaces (Couëtoux, Hoock, Sokolovska, Teytaud, & Bonnard, 2011), there is relatively little experience with MCTS in DRA-like problems.

On the other hand, within the OR community, the study of DRAs has proceeded along different lines. A prominent stream of research is based upon mathematical optimization (MO). In contrast to MCTS which only requires access to a generative model, MO approaches model the dynamics of the system *explicitly* via a constrained optimization problem. The solution to this optimization problem then yields a control policy for the system. We consider a specific MO-based approach that is sometimes called *rolling horizon optimization* (RHO). Specifically, we replace uncertain parameters in a MO formulation with their expected values and periodically re-solve the formulation for an updated policy as the true system evolves. This paradigm goes by many other names such as fluid approximation, certainty equivalent control or model predictive control. It is known to have excellent practical performance in applications like queueing (Avram, Bertsimas, & Ricard, 1995) and network revenue management (Ciocan & Farias, 2012), and in some special cases, also enjoys strong theoretical guarantees (e.g., Ciocan & Farias, 2012; Gallego & van Ryzin, 1994).

The widespread use and success of RHO approaches for DRAs contrasts strongly with a lack of computational experience with MCTS for DRAs. Furthermore, the two methods differ philosophically. MCTS involves directly simulating the *true* system and efficiently searching through the tree of state-action trajectories. In contrast, RHO involves first constructing an *approximation* of the true system and then solving an optimization problem based on this approximation to determine a policy; this policy is generally not guaranteed to be optimal for the true system. MCTS and RHO also differ in their informational requirements. MCTS only requires a generative model for simulating transitions, and one can interact with this model in a "black-box" fashion, without being able to precisely and compactly describe its dynamics. On the other hand, RHO requires one to know something about the dynamics of the system in order to specify the underlying MO model.

In this paper, we aim to understand the relative merits of MCTS and RHO by applying them to two challenging DRA problems:

1. **Tactical wildfire management.** The decision maker controls the spread of a fire on a discrete grid (representing a wildland area) by deploying suppression resources to cells on this grid. This problem is computationally intractable: each cell on the grid may be burning or not burning, resulting in an exponentially large state space, while the allocation decision involves choosing a subset of the burning cells to extinguish, resulting in an exponentially large action space. This problem is also of practical importance: for example, in the US, increasing wildfire severity has resulted in increased government spending on wildfire management, amounting to $3.5 billion in 2013 (Bracmort, 2013).
2. **Queueing network control.** The decision maker controls a network of servers that serve jobs of different classes and at each decision epoch, must decide which job class each server should process so as to minimize the average long-run number of jobs in the system. The system state is encoded by the number of jobs of each class and is exponential in the number of job classes. The action is to decide which class each server should service, and is also exponential in the number of servers. The problem thus constitutes a challenging DRA. At the same time, queueing networks arise in many domains such as manufacturing (Buzacott & Shanthikumar, 1993), computer systems (Harchol-Balter, 2013) and neuroscience (Mišić, Sporns, & McIntosh, 2014).

We summarize our contributions as follows:

1. We develop an MCTS approach for the tactical wildfire management problem and the queueing network control problem. To the best of our knowledge, this represents the first application of MCTS to challenging DRA problems motivated by real-world applications. Towards this end, we combine a number of classical features of MCTS, such as bandit upper confidence bounds, with new features such as double progressive widening (Couëtoux, Hoock, Sokolovska, Teytaud, & Bonnard, 2011). For the wildfire problem, we also propose a novel action generation approach to cope with the size of the state and action spaces of the DRA.
2. We propose an RHO approach based on a mixed-integer optimization (MIO) model of the wildfire problem that approximates the original discrete and stochastic elements of the MDP by suitable continuous and deterministic counterparts. This particular formulation incorporates elements of a linear dynamical system which may be of independent interest in other DRA problems. For the queueing control problem, we apply an existing fluid optimization approach (Avram, Bertsimas, & Ricard, 1995).
3. Through extensive computational experiments in both problems, we show the following:
   (a) MCTS and RHO both produce high-quality solutions, generally performing as well or better than a baseline heuristic. MCTS and RHO perform comparably when the problem instance is small. With a fixed computational budget, however, the RHO approach begins to outperform the MCTS approach as the size of the problem instance grows, either in state space or action space. Indeed, in the wildfire problem, MCTS can begin to perform worse than our baseline heuristic when the action space grows very large; the RHO approach, by comparison, still performs quite well. Similarly, for queueing network control, MCTS with an informed rollout policy (the $c\mu$ rule) often performs worse than the same rollout policy on its own for larger queueing systems.
   (b) The choice of hyperparameters in MCTS—such as the exploration bonus and the progressive widening parameters—can significantly affect its overall performance. The interdependence between these parameters is complex and in general, they cannot be chosen independently. Some care must be taken to appropriately tune the algorithm to a specific DRA.

In tactical wildfire management, there have been a number of empirically validated, deterministic models for wildfire spread proposed (e.g., Tymstra, Bryce, Wotton, Taylor, & Armitage, 2010). However, there have been fewer works that incorporate the stochastic elements of fire spread (Boychuck, Braun, Kulperger, Krougly, & Stanford, 2008; Fried, Gilless, & Spero, 2006; Ntaimo, Arrubla, Stripling, Young, & Spencer, 2012). Most works focus on developing simulation models; few consider the associated problem of managing suppression resources. A notable exception is the research stream of Hu and Ntaimo (2009) and Ntaimo et al. (2013), which considers the problem of determining how many and what

kind of suppression resources to allocate to a wildfire, but not the tactical DRA problem of *optimally* dynamically allocating suppression resources. In queueing network control, a variety of approaches have been proposed, including ones based on heavy traffic approximations (Harrison, 1988), fluid approximations (Avram, Bertsimas, & Ricard, 1995) and characterizing the achievable performance region (Bertsimas, Paschalidis, & Tsitsiklis, 1994). To the best of our knowledge, MCTS has not been previously applied in this domain.

The rest of this paper is organized as follows. In Section 2, we introduce our MDP formulations of the tactical wildfire management and the queueing network control problems that we use to compare the MCTS and RHO approaches. In Sections 3 and 4, we describe the MCTS and RHO approaches, respectively. In Sections 5 and 6, we describe our computational experiments for the wildfire and queueing control problems, respectively, and report on the results of these experiments. Finally, in Section 7, we summarize the main contributions of the work and highlight promising directions for future research.

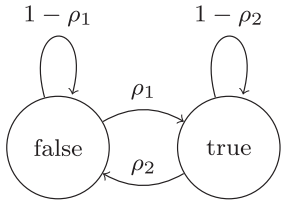


Fig. 1. $B(x)$ transition model.

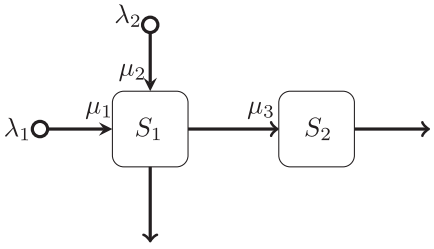


Fig. 2. Criss-cross network from Bertsimas, Nasrabadi, and Paschalidis (2015).

2. Problem definition

We begin by describing the dynamics of the two problems that we will use to compare MCTS and RHO.

2.1. Tactical wildfire management

Specifically inspired by the stochastic model of Boychuck, Braun, Kulperger, Krougly, and Stanford (2008) for wildland fire simulation, we propose a Markov decision process model of tactical wildfire management. We partition the landscape into a grid of cells $\mathcal{X}$. Each cell $x \in \mathcal{X}$ is endowed with two attributes: $B(x)$, a Boolean (0/1) variable indicating whether the cell is currently burning, and $F(x)$, an integer variable indicating how much fuel is remaining in the cell. The collection of these two attributes over all cells in the grid represents the state of the Markov decision process.

We assume a set of suppression teams $\mathcal{I}$. To simplify the model, we will treat all suppression teams as identical, but it is straightforward to extend to the case of heterogeneous teams. The decisions of our MDP correspond to assigning teams to cells. For each $i \in \mathcal{I}$, let $a^{(i)} \in \mathcal{X}$ denote the cell to which we assign suppression team i. We assume that any team can be assigned to any cell at any time step, i.e., that the travel time between cells is negligibly small compared to the time between each period.

Once ignited, a cell consumes fuel at a constant rate. Once the fuel is exhausted, the cell extinguishes. Since fuel consumption occurs at a constant rate, without loss of generality, we can rescale the units of fuel to make this rate equal to unity. Thus, we model the evolution of fuel in the model by

$$F_{t+1}(x) = \begin{cases} F_t(x) & \text{if } \neg B_t(x) \vee F_t(x) = 0 \\ F_t(x) - 1 & \text{otherwise,} \end{cases} \tag{1}$$

where $\neg$ denotes the Boolean complement ($\neg 0 = 1$; $\neg 1 = 0$) and $\vee$ denotes the Boolean "Or". Note that this evolution is deterministic given $B_t(x)$.

The evolution of $B_t(x)$, however, is stochastic. Fig. 1 shows the probabilistic transition model for $B_t(x)$ where

$$\rho_1 = \begin{cases} 1 - \prod_y (1 - P(x,y)B_t(y)) & \text{if } F_t(x) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

and

$$\rho_2 = \begin{cases} 1 & \text{if } F_t(x) = 0 \\ 1 - \prod_i (1 - S(x)\delta_x(a^{(i)})) & \text{otherwise.} \end{cases} \tag{3}$$

Here, $P(x, y)$ is the probability that a fire in cell y ignites a fire in cell x. Generally, only the neighbors of x can ignite x, and so we expect $P(x, y)$ to be sparse. The specification of $P(x, y)$ can capture the tendency of a fire to propagate primarily in one direction due to wind or sloping terrain. $S(x)$ is the probability that a suppression effort on cell x successfully extinguishes the cell, while $\delta_x(a^{(i)})$ is an expression that is 1 if $a^{(i)} = x$ (suppression team i is allocated to cell x) and 0 otherwise. We assume that the probability of success for multiple attempts on the same cell are independent. Under these dynamics, cells that have been previously extinguished may later reignite.

The reward for a cell burning is $R(x)$ (always negative) and the total reward received at the tth step is $\Sigma_x B_t(x)R(x)$. We can vary the reward across the grid to represent a higher cost of fire in particular areas (e.g., urban zones).

2.2. Control of queueing networks

In this problem, there is a network of servers and each server is fed by one or more queues/buffers that correspond to different job classes. Each job class is served by only one server, but a server may serve several different job classes. When a server becomes free, it can choose to process an available job from one of its classes. Once the server finishes, the job either exits the system, or proceeds to another server; at the new server, it becomes a job of a different class and waits in the queue for the class until the new server begins processing it. The time each server takes to process a job is random, with the distribution determined by the job's class. Jobs of some classes may arrive exogenously to the system. For the problem we will study, each such arrival process is an independent Poisson process, and the service time for each class is exponential.

As an example, let us consider the example network in Fig. 2, taken from Bertsimas, Nasrabadi, and Paschalidis (2015). In this example, there are two servers and three job classes. Jobs of class 1 and 2 arrive exogenously to the system with rates λ_1 and λ_2 respectively, and are serviced by server S_1 with rates μ_1 and μ_2 respectively. When a job of class 2 is completed, it exits the system. When a job of class 1 is completed, it becomes a job of class 3, and enters the corresponding buffer, where it awaits service from server S_2. The service time of a class 3 job in S_2 is exponential with rate μ_3; once a class 3 job completes service at S_2, it exits the system.

The queue sizes for the different job classes fluctuate over time as jobs arrive, enter service and are routed through the system. Whenever a server completes a job and becomes free, we must decide which queue (i.e., which job class) the server will draw its next job from. The problem of queueing network control is to decide at each such decision epoch which job class will be served so as to minimize the expected weighted long-run average number of jobs in the system.

To define the queueing network control problem, we use n to denote the number of job classes and m to denote the number of servers. We use $s(i)$ to denote the server of job class i. We use λ_i to denote the rate of the exogenous Poisson arrival process to class i (note that λ_i may be 0, in which case there are no exogenous arrivals to class i). We use μ_i to denote the exponential service time rate for job class i when it is served by server $s(i)$. We use g_{ij} to denote routing; more specifically, $g_{ij} = 1$ if class i jobs become class j jobs after service at server $s(i)$, and 0 otherwise. Note that $g_{ij} = 0$ for every j denotes the fact that jobs of class i, once processed by $s(i)$, exit the system, and do not join a queue at a different server. We use $\mathbf{c} = (c_1, \ldots, c_n)$ to denote the vector of objective function weights for each class. Finally, we let T denote the time horizon for the queuing problem.

We let $\mathbf{x}(t) = (x_1(t), x_2(t), \ldots, x_n(t))$ denote the number of jobs of class i in the system (including a job that may be in service) at time t. The objective is to minimize the expected weighted long-run average number of jobs in the system from $t = 0$ to $t = T$, which is given by the following performance metric:

$$\frac{1}{T} \cdot \mathbb{E}_\pi \left[\int_0^T \mathbf{c}^T \mathbf{x}(t)\, dt \right],$$

where π denotes the policy for deciding which class to serve at each epoch.

3. A rolling horizon optimization approach

We now present rolling horizon optimization approaches for the tactical wildfire management problem and for the queueing network control problem.

3.1. Approach for tactical wildfire management

In this section, we present an optimization-based solution approach for the tactical wildfire management problem. In this approach, we formulate a deterministic optimization problem that approximates the original MDP. At each decision step, we re-solve this approximation based on the current state of the process and select the first prescribed allocation. The key feature of the formulation is the use of a deterministic, "smoothed" version of the dynamics presented in Section 2. Rather than modeling the state with a discrete level of fuel and binary state of burning, we model fuel as a continuous quantity and model a new (continuous) *intensity* level of each cell representing the rate at which fuel is consumed. Then, rather than modeling the evolution of cells being burning/not burning using probabilistic transitions (cf. Fig. 1), we model the evolution of intensity through one-step linear dynamics. Other authors have used similar ideas when motivating various fluid models in the OR literature (see, e.g., Gallego & van Ryzin, 1994; Avram, Bertsimas, & Ricard, 1995).

Smoothing the dynamics in this way results in a tractable optimization model that can be solved to obtain an allocation at each time step. We wish to emphasize that this formulation is *not* identical to the original problem. As a result, the derived allocation at each period is not guaranteed to be the same as the optimal allocation of the original MDP in Section 2.1. Nevertheless, given the wide application of fluid models to other, similarly intractable problems in the OR literature (such as Gallego & van Ryzin, 1994; Avram, Bertsimas, & Ricard, 1995), it seems reasonable to expect that our approach may still yield good policies for the original MDP in Section 2.1.

Let $A_t(x, i)$ be a binary variable that is 1 if suppression team $i \in \mathcal{I}$ is assigned to cell x at time t and 0 otherwise; this is the main decision variable of the problem. Let $I_t(x)$ represent the intensity of the fire in cell $x \in \mathcal{X}$ at time t, which is a continuous, nonnegative decision variable. Finally, let $F_t(x)$ denote the amount of fuel available at the start of period t in cell x. Due to the continuous nature of $I_t(x)$, $F_t(x)$ does not directly correspond to fuel in the original MDP formulation. In the online supplement, we discuss how to calibrate the initial fuel values $F_0(x)$.

Some additional notation is required to describe the evolution of $I_t(x)$. Define $\mathcal{N}(x)$ as the set of cells that are neighbors of x in the sense of fire transmission, i.e., $\mathcal{N}(x) = \{y : P(x, y) > 0\}$. Let $\zeta_t(y, x) \in [0, 1]$ be the rate at which the intensity $I_{t-1}(y)$ of cell y at time $t-1$ contributes to the intensity $I_t(x)$ of cell x at time t. Furthermore, let $\tilde{\zeta}_t(x, i) \in [0, 1]$ be the relative reduction in intensity when suppression team i is assigned to cell x at time t. We discuss how ζ and $\tilde{\zeta}$ are calibrated in the online supplement. Finally, let $I_0(x) = 1$ if cell x is burning at time 0, and $I_0(x) = 0$ if cell x is not burning. Note that these values are simply the initial values of the intensity variables and $I_t(x)$ for $t > 0$ can take on any nonnegative value.

With this notation, our formulation is

$$\text{minimize} \quad \sum_{x \in \mathcal{X}} \sum_{t=0}^{T} -R(x) I_t(X) \tag{4a}$$

$$\begin{aligned} \text{subject to} \quad & I_t(x) \geq I_{t-1}(x) + \sum_{y \in \mathcal{N}(x)} I_{t-1}(y) \cdot \zeta_t(y, x) \\ & - \sum_{i \in \mathcal{I}} A_{t-1}(x, i) \cdot \bar{I}_t(x) \cdot \tilde{\zeta}_t(x, i) \\ & - \left(F_0(x) + \sum_{y \in \mathcal{N}(x)} F_0(y) \right) \cdot z_{t-1}(x), \\ & \forall x \in \mathcal{X},\ t \in \{1, \ldots, T\}, \end{aligned} \tag{4b}$$

$$F_t(x) = F_0(x) - \sum_{t'=0}^{t-1} I_{t'}(x), \quad \forall x \in \mathcal{X},\ t \in \{0, \ldots, T\}, \tag{4c}$$

$$F_t(x) \geq \delta \cdot (1 - z_t(x)), \quad \forall x \in \mathcal{X},\ t \in \{0, \ldots, T\}, \tag{4d}$$

$$F_t(x) \leq \delta \cdot z_t(x) + F_0(x) \cdot (1 - z_t(x)), \quad \forall x \in \mathcal{X},\ t \in \{0, \ldots, T\}, \tag{4e}$$

$$I_{t+1}(x) \leq F_0(x) \cdot (1 - z_t(x)), \quad \forall x \in \mathcal{X},\ t \in \{0, \ldots, T\}, \tag{4f}$$

$$\sum_{x \in \mathcal{X}} A_t(x, i) \leq 1, \quad \forall t \in \{0, \ldots, T\},\ i \in \mathcal{I}, \tag{4g}$$

$$I_t(x),\ F_t(x) \geq 0, \quad \forall x \in \mathcal{X},\ t \in \{0, \ldots, T\}, \tag{4h}$$

$$z_t(x) \in \{0, 1\}, \quad \forall x \in \mathcal{X},\ t \in \{0, \ldots, T\}, \tag{4i}$$

$$A_t(x, i) \in \{0, 1\}, \quad \forall x \in \mathcal{X},\ i \in \mathcal{I},\ t \in \{0, \ldots, T\}. \tag{4j}$$

Here, $\delta > 0$ is a small threshold chosen so that a cell with less than δ units of fuel cannot burn. Consequently, the binary decision variable $z_t(x)$ indicates whether the fuel at time t in cell x is below δ or not. Finally in the spirit of "big-M" constraints, $\bar{I}_t(x)$ is an upper bound on the maximal value attainable by $I_t(x)$. We discuss how δ and $\bar{I}_t(x)$ are set in the online supplement.

We now describe the constraints. Constraint (4b) is the main constraint, which expresses the one-step dynamics of the fire intensity in region x at time t. Although the constraint is in inequality form, it is not difficult to see that in an optimal solution, the constraint will always be satisfied at equality since the

objective is a sum of the intensities over all periods and regions weighted by the (positive) importance factors. The first two terms of the right-hand side represent that—without intervention and without regard for fuel—the intensity of a cell one step into the future is the current intensity ($I_{t-1}(x)$) plus the sum of the intensities of the neighboring cells weighted by the transmission rates ($\sum_{y\in\mathcal{N}(x)} I_{t-1}(y)\cdot\zeta_t(y,x)$). If suppression team i is assigned to cell x at $t-1$, then $A_t(x,i)=1$ and the intensity is reduced by $\tilde{\zeta}_t(x,i)\cdot\tilde{I}_t(x)$. If the cell's fuel is below δ at time $t-1$, then $z_t(x)=1$, and the intensity is reduced by $F_0(x)+\sum_{y\in\mathcal{N}(x)}F_0(y)$; since the intensity of a cell is upper bounded by the initial fuel of that cell, the term $-\left(F_0(x)+\sum_{y\in\mathcal{N}(x)}F_0(y)\right)\cdot z_{t-1}(x)$ ensures that whenever the fuel $F_t(x)$ drops below δ, this constraint becomes vacuous.

With regard to the rest of the model, constraint (4c) is the equation for the remaining fuel at a given period as a function of the intensities (intensity is assumed to be the fuel burned in a particular time period). Constraints (4d) and (4e) are forcing constraints that force $F_t(x)$ to be between δ and $F_0(x)$ if $z(t)=0$, and between 0 and δ if $z(t)=1$. Constraint (4f) ensures that if there is insufficient fuel in cell x at period t, then the intensity in the next period is zero. If there is sufficient fuel, then the constraint is vacuous (the intensity is at most $F_0(x)$, which is already implied in the formulation). Constraint (4g) ensures that each team in each period is assigned to at most one cell. The remaining constraints ensure that the fuel and intensity are continuous nonnegative variables, and that the sufficient fuel and team assignment variables are binary. Finally, the objective (4a) is the sum of the intensities over all time periods and all cells, weighted by the importance factor of each cell in each period.

Problem (4) is a mixed-integer optimization (MIO) model. Although MIO problems are not solvable in polynomial time, there exist high-quality solvers that are able to solve such problems efficiently in practice. In resource-constrained environments when it is not possible to solve the above model to optimality, we can still obtain good approximate solutions by relaxing the $A_t(x,i)$ variables to be continuous within the unit interval [0, 1]. Then, given an optimal solution with fractional values for the $A_t(x,i)$ variables at $t=0$, we can compute a score $v(x)$ for each cell x as $v(x)=\sum_{i\in\mathcal{I}}A_0(x,i)$. We then assign suppression teams to the $|\mathcal{I}|$ cells with the highest values of the index v. Indeed, we will follow this strategy in Section 5.

3.2. Approach for queueing network control

In this section, we describe the fluid optimization approach that we will use for the queueing network control problem; the approach was first developed in Avram, Bertsimas, and Ricard (1995) and later extended in Bertsimas, Nasrabadi, and Paschalidis (2015). The key idea is to replace the probabilistic arrival and service dynamics with a system of deterministic differential equations. As for the deterministic model in Section 3.1, we emphasize that this new optimization problem is not equivalent to the true MDP in Section 2.2 and as such, the resulting decision of which queue each server should work on is not necessarily optimal for the true MDP. However, it is well-known that fluid approximations of queueing networks are closely related to the true queueing network. For example, a queueing network is stable if its fluid approximation is stable (Dai, 1995). Moreover, fluid approaches perform very well in simulations (Avram, Bertsimas, & Ricard, 1995; Bertsimas, Nasrabadi, & Paschalidis, 2015).

The decision variables are $\mathbf{x}(t)=(x_1(t),\ldots,x_n(t))$ for $t\in[0,T]$, where $x_i(t)$ indicates the *fluid* of class i at time t, and $\mathbf{u}(t)=(u_1(t),\ldots,u_n(t))$ where $u_i(t)$ denotes the effort (rate of service) that is devoted to queue i by its server $s(i)$.

We use the same data in Section 2.2. For convenience, we define the matrix $\mathbf{A}$ as

$$a_{ij}=\begin{cases}1 & \text{if } i=j,\\ -1 & \text{if class } i \text{ jobs become class } j \text{ jobs upon service by server } s(i),\\ 0 & \text{otherwise.}\end{cases}$$

The dynamics of the class fluids can be written in matrix form as

$$\dot{\mathbf{x}}(t)=\boldsymbol{\lambda}-\mathbf{A}\mathbf{u}(t),\quad \forall t\in[0,T]. \tag{5}$$

By integrating on both sides for every t, the constraint can be rewritten as

$$\int_0^t \mathbf{A}u(s)\,ds+\mathbf{x}(t)=\mathbf{x}(0)+\boldsymbol{\lambda}t,\quad \forall t\in[0,T] \tag{6}$$

where $\mathbf{x}(0)$ is a vector of the number of jobs of each class present at time 0.

The variable $u_i(t)$ denotes the rate at which class i is being serviced by server $s(i)$. These rates are constrained as follows: for a given server s, we must have

$$\sum_{i:s(i)=s}\frac{u_i(t)}{\mu_i}\le 1,\quad \forall t\in[0,T]. \tag{7}$$

In words, $u_i(t)/\mu_i$ represents the rate that class i is being serviced as a fraction of server s's (maximum) rate for class i, and so the constraint ensures that in a fluid sense, the server is serving at most one job per time unit. By defining the matrix $\mathbf{H}$ componentwise as

$$h_{si}=\begin{cases}1/\mu_i & \text{if } s(i)=s,\\ 0 & \text{otherwise,}\end{cases}$$

the constraint can be written in matrix form as

$$\mathbf{H}\mathbf{u}(t)\le\mathbf{1},\quad \forall t\in[0,T]. \tag{8}$$

where $\mathbf{1}$ is a column vector of m ones.

With these definitions, the fluid optimization problem can be written as

$$\text{minimize}\quad \frac{1}{T}\int_0^T \mathbf{c}^T\mathbf{x}(t)\,dt \tag{9a}$$

$$\text{subject to}\quad \int_0^t \mathbf{A}u(s)\,ds+\mathbf{x}(t)=\mathbf{x}(0)+\boldsymbol{\lambda}t,\quad \forall t\in[0,T], \tag{9b}$$

$$\mathbf{H}\mathbf{u}(t)\le\mathbf{1},\quad \forall t\in[0,T], \tag{9c}$$

$$\mathbf{u}(t),\mathbf{x}(t)\ge 0,\quad \forall t\in[0,T], \tag{9d}$$

where the objective is the long-run average weighted fluid in the system and the last two constraints require that the effort devoted to each class and the fluid of each class are never below zero.

The problem can be further re-formulated to eliminate the $\mathbf{x}(t)$ variable:

$$\text{minimize}\quad \frac{1}{T}\int_0^T (T-t)\mathbf{c}^T(\boldsymbol{\lambda}-\mathbf{A}\mathbf{u}(t))\,dt \tag{10a}$$

$$\text{subject to}\quad \int_0^t \mathbf{A}u(s)\,ds\le\mathbf{x}(0)+\boldsymbol{\lambda}t,\quad \forall t\in[0,T], \tag{10b}$$

$$\mathbf{H}\mathbf{u}(t)\le\mathbf{1},\quad \forall t\in[0,T], \tag{10c}$$

$$\mathbf{u}(t)\ge 0,\quad \forall t\in[0,T]. \tag{10d}$$

This problem is a semi-continuous linear programming (SCLP) problem that has been studied extensively (see, e.g., Pullan, 1993). Although the problem appears to be intractable (having an infinite number of variables and constraints), it turns out that it is possible to characterize the structure of the optimal solution (namely, the

control $\mathbf{u}(t)$ is a piecewise constant function) and there exist efficient solution algorithms that exploit this (e.g., Luo & Bertsimas, 1998).

The fluid control approach to the queueing network control problem is, at each decision epoch, to do as follows:

1. Set $\mathbf{x}(0)$ to reflect the current number of jobs of each class in the system.
2. Solve problem (10) to obtain the optimal control $\mathbf{u}^*(t)$.
3. At each server s that is idle, serve the non-empty class i of that server with the highest value of $u_i^*(0)$.

4. Monte Carlo tree search

Our review of MCTS is necessarily brief; the reader is referred to Browne et al. (2012) for a more thorough survey. In particular, we will focus on a variation of MCTS that employs *double progressive widening*, a technique that explicitly controls the branching factor of the search tree (Couëtoux, Hoock, Sokolovska, Teytaud, & Bonnard, 2011). This variation is specifically necessary when the action space is continuous or so large that all actions cannot possibly be explored. This is frequently the case in most DRA problems. We note that there exist other strategies for managing the branching factor, such as the pruning strategy employed in the "bandit algorithm for smooth trees" (BAST) method of Coquelin and Munos (2007); we do not consider these here.

We first describe MCTS with double progressive widening in general, and then discuss two particular modifications we have made to the algorithm to tailor it to DRAs.

4.1. MCTS with double progressive widening

Algorithm 1 provides a pseudocode description of the MCTS algorithm. This algorithm involves running many simulations from the current state while updating an estimate of the state-action value function $Q(s, a)$. Each simulation from the current state is executed to a depth of d. We use a generative model G to produce samples of the next state s' and reward r given the current state s and action a. All of the information about the state transitions and rewards is represented by G; the transition probabilities and rewards are not used directly. There are three stages in each simulation: search, expansion, and rollout.

Search. If the current state in the simulation is in the set T (initially empty), we enter the search stage. Otherwise, we proceed to the expansion stage. During the search stage, we update $Q(s, a)$ for the states and actions visited and tried in our search. We also keep track of the number of times we have visited a state $N(s)$ and the number of times we have taken an action from a state $N(s, a)$.

During the search, the first progressive widening controls the number of actions considered from a state. To do this, we generate a new action if $|A(s)| < kN(s)^\alpha$, where k and α are parameters that control the number of actions considered from the current state and $A(s)$ is the set of actions tried from s. When generating a new action, we add it to the set $A(s)$, and initialize $N(s, a)$ and $Q(s, a)$ with $N_0(s, a)$ and $Q_0(s, a)$, respectively. The functions N_0 and Q_0 can be based on prior expert knowledge of the problem; if none is available, then they can both be initialized to 0. We also initialize the empty set $V(s, a)$, which contains the set of states s' transitioned to from s when selecting action a. A default strategy for generating new actions is to randomly sample from candidate actions. After potentially generating new actions, we execute the action that maximizes

$$Q(s, a) + c\sqrt{\frac{\log N(s)}{N(s, a)}},$$

Algorithm 1 Monte Carlo tree search with double progressive widening.

```
function MonteCarloTreeSearch(s, d)
    loop
        Simulate(s, d)
    return arg max_a Q(s, a)
function Simulate(s, d)
    if d = 0 then
        return 0
    if s ∉ T then
        T = T ∪ {s}
        N(s) ← N_0(s)
        return Rollout(s, d)
    N(s) ← N(s) + 1
    if |A(s)| < kN(s)^α then
        a ← GetNext(s, Q)
        (N(s, a), Q(s, a), V(s, a)) ← (N_0(s, a), Q_0(s, a), V(s, a))
        A(s) = A(s) ∪ {a}
    a ← arg max_a Q(s, a) + c sqrt(log N(s) / N(s, a))
    if |V(s, a)| < k'N(s, a)^α' then
        (s', r) ~ G(s, a)
        if s' ∉ V(s, a) then
            V(s, a) = V(s, a) ∪ {s'}
            N(s, a, s') ← N_0(s, a, s')
        else
            N(s, a, s') ← N(s, a, s') + 1
    else
        s' ← Sample(N(s, a, ·))
    q ← R(s, a, s') + γ Simulate(s', d − 1)
    N(s, a) ← N(s, a) + 1
    Q(s, a) ← Q(s, a) + (q − Q(s, a)) / N(s, a)
    return q
function Rollout(s, d)
    if d = 0 then
        return 0
    a ~ π_0(s)
    (s', r) ~ G(s, a)
    return r + γ Rollout(s', a, d − 1)
```

where c is a parameter that controls the amount of exploration in the search. The second term is an *exploration bonus* that encourages selecting actions that have not been tried as frequently.

Next, we draw a sample $(s', r) \sim G(s, a)$, if $|V(s, a)| < k'N(s, a)^{\alpha'}$. In this second progressive widening step, the parameters k' and α' control the number of states transitioned to from s. If s' is not a member of $V(s, a)$, we add it to the set $V(s, a)$, initialize $R(s, a, s')$ to r, and initialize $N(s, a, s')$ with $N_0(s, a, s')$. If s' is in $V(s, a)$, then we increment $N(s, a, s')$. However, if $|V(s, a)| \geq k'N(s, a)^{\alpha'}$, then we select s' from $V(s, a)$ proportionally to $N(s, a, s')$.

Expansion. Once we reach a state not in T, we initialize $N(s, a)$ with $N_0(s, a)$, add the current state to the set T and proceed to the rollout stage.

Rollout. After the expansion stage, we simply select actions according to some *rollout* (or default) policy π_0 until the desired depth is reached. Typically, rollout policies are stochastic, and so the action to execute is sampled $a \sim \pi_0(s)$. The rollout policy does not have to be close to optimal, but it is a way for an expert to bias the search into promising areas. The expected value is returned and is used to update the value for $Q(s, a)$ used by the search phase.

Simulations are run until some stopping criterion is met, often simply a fixed number of iterations. We then execute the action

that maximizes $Q(s, a)$. Once that action has been executed, we can rerun MCTS to select the next action.

4.2. Tailoring MCTS to DRAs

We next discuss two modifications we have found to be critical when applying MCTS to DRAs.

Action generation. As previously mentioned, the default strategy for generating new actions during the search stage of MCTS involves randomly sampling from all candidate actions. In DRAs where the action space may be very large, this strategy is inefficient; we may need to search many actions before identifying a high-quality choice.

We now describe an alternate scheme for generating actions. Specifically, consider MCTS after several iterations. The current values of $Q(s, a)$ provide a (noisy) estimate of the value function and hence, can be used to approximately identify promising actions. Consequently, we use these estimates to bias our sampling procedure through a sampling scheme inspired by genetic algorithm search heuristics (Whitley, 1994). Our strategy, described in Algorithm 2, involves generating actions using one of three approaches: with probability u' an existing action in the search tree is mutated, with probability u'' two existing actions in the search tree are recombined, or with probability $1 - u' - u''$, a new action is generated from the default strategy. Mutating involves randomly changing the allocation of one or resources. Recombining involves selecting a subset of allocations from two actions and combining the two subsets. When mutating or recombining, we select the existing action (or actions) from $A(s)$ using tournament select where the fitness for each action is proportional to $Q(s, a)$. Our numerical experiments confirm that our proposed action generation approach significantly outperforms the default strategies.

Algorithm 2 Action generation.

```
function GetNext(s, Q)
    u ~ U(0, 1)
    if u < u' then
        a' ← Sample(Q(s, ·))
        a ← Mutate(a')
    else if u < u' + u'' then
        a' ← Sample(Q(s, ·))
        a'' ← Sample(Q(s, ·))
        a ← Recombine(a', a'')
    else
        a ~ π0(s)
    return a
```

Rollout policy. In many papers treating MCTS, it is argued that even if the heuristic used for the rollout policy is highly suboptimal, given enough time the algorithm will converge to the correct state-action value function $Q(s, a)$. In DRAs with combinatorial structure (and hence, huge state and action spaces), it may take an extremely long time for this convergence; similar behavior was observed in Coquelin and Munos (2007). Indeed, we have observed for DRAs that having a good initial rollout policy makes a material difference in the performance of MCTS.

For the tactical wildfire management problem, we consider a heuristic that involves assigning a weight $W(x)$ to each cell x, defined as

$$W(x) = \sum_y \frac{R(y)}{D(x, y)}, \tag{11}$$

where $D(x, y)$ is the shortest path between x and y assuming that the distance between adjacent cells is $P(x, y)$. We compute the values $D(x, y)$ offline using the Floyd–Warshall algorithm (Floyd, 1962), and consequently term this heuristic the FW heuristic. The heuristic performs generally well because it prioritizes allocating teams to cells that are near large negative reward cells. The rollout policy involves selecting the cells that are burning and assigning teams to the highest weighted cells. We are also able to randomly sample from the weights to generate candidate actions. In our experiments, we have observed that the heuristic performs fairly well and consequently, may be of independent interest to the wildfire suppression community.

For the queueing network control problem, in addition to a random policy, we also consider the so-called $c\mu$ heuristic. Under this heuristic, when a server becomes idle, the class it draws its next job from is the class i with the highest value of $c_i\mu_i$ (the service rate weighted by the cost of the class). This type of policy is simple to implement and in many special cases, is known to be either optimal or asymptotically optimal (see, for example, Buyukkoc, Varaiya, & Walrand, 1985; Van Mieghem, 1995).

Table 1
Default parameters for algorithms.

Method	Parameter	Value
MCTS	Time limit per Iteration	60 seconds
	Exploration bonus c	50
	Rollout policy	FW Heuristic
	Depth d	10
	Progressive widening, action space α	0.5
	Progressive widening, state space α'	0.2
	Progressive widening, action space k	40
	Progressive widening, state space k'	40
	Algorithm 2 (u', u'')	(0.3, 0.3)
RHO	Time limit per iteration	60 seconds
	Horizon length	10

5. Numerical comparisons for tactical wildfire management

This section presents experiments comparing Monte Carlo tree search (MCTS) and the rolling horizon optimization (RHO) approach. We seek to understand their relative strengths and weaknesses as well as how the user-defined parameters of each approach, such as the exploration bonus c for MCTS, affect the performance of the algorithm. Before proceeding to the details of our experiments, we summarize our main insights here:

- Overall, RHO performs as well as or better than MCTS. For even moderate computational budgets, RHO generates high quality solutions.
- Although the MCTS approach works well for certain smaller cases, its performance can degrade for larger cases (with a fixed, computational budget). Moreover, the dependence of the algorithm on its underlying hyperparameters is complex. The optimal choice of the exploration bonus and progressive widening factors depends both on the rollout heuristic as well as the available computational budget.

5.1. Algorithmic parameters and basic experimental setup

In what follows, we use a custom implementation of MCTS written in C++ and use the mixed-integer optimization software Gurobi 5.0 (Gurobi Optimization, Inc., 2013) to solve the RHO formulation. All experiments were conducted on a computational grid with 2.2GHz cores with 4GB of RAM in a single-threaded environment.

Many of our experiments will study the effect of varying various hyperparameters (like the time limit per iteration) on the solution quality. Unless otherwise specified in the specific experiment, all hyperparameters are set to their baseline values in Table 1.

To ease comparison in what follows, we generally present the performance of each our algorithms relative to the performance of a randomized suppression heuristic. At each time step, the randomized suppression heuristic chooses $|\mathcal{I}|$ cells without replacement from those cells which are currently burning and assigns suppression teams to them. This heuristic should be seen as a naive "straw man" for comparisons only. We will also often include the performance of our more tailored heuristic, the Floyd–Warshall (FW) heuristic, as a more sophisticated straw man.

In our experiments, we consider a $k \times k$ grid with a varying reward function. There is a negative one reward received when the lower left cell is burning and the reward for a cell burning decreases by one when traversing one cell up or to the right across the grid. The fire propagates as described in Section 2 with

$$P(x, y) = \begin{cases} p, & \text{if } y \in \mathcal{N}(x) \\ 0, & \text{otherwise,} \end{cases}$$

where $p = 0.06$. We also assume for this experiment that suppression efforts are successful with an 80% probability—that is, $Q(x) = 0.8$ for all $x \in \mathcal{X}$.

For a single simulation we randomly generate an initial fire configuration—that is, whether or not each cell is burning and the fuel level in each cell. After generating an initial fire, suppression then begins according to one of our four approaches with $|\mathcal{I}|$ teams. The simulation and suppression efforts continue until the fire is extinguished or the entire area is burned out. A typical experiment will repeat this simulation many times with different randomly generate initial fire configurations and aggregate the results.

Our process for initializing the fire in a simulation is as follows:

1. Initialize all of the cells with a fuel level of $\lfloor k/2p \rfloor$ and seed a fire in the lower left hand cell.
2. Allow the fire to randomly propagate for $\lfloor k/2p \rfloor$ steps. Note that the lower left hand cell will naturally extinguish at the point.
3. Next, scale the fuel levels by a factor of $k^{-0.25}$. We scale the fuel levels to reduce the length of experiments where the number of teams is insufficient to successfully fight the fire.

5.2. Tuning Hyperparameters for the MCTS Methodology

The MCTS approach includes a number of hyperparameters to control the performance of the algorithm. In this section, we explore the effects of some of these parameters with $k = 20$ and 4 assets, and a time limit of 10 s per iteration. We vary the exploration bonus $c \in \{0, 10, 50, 100\}$, the progressive widening factors $\alpha = \alpha' \in \{1, 0.5, 0.2\}$, the depth $d \in \{1, 5, 10\}$, whether or not we use Algorithm 2 (based on genetic algorithm search heuristics) in action generation, and whether we use the random suppression heuristic or the FW heuristic in the rollout. For each combination of hyperparameters, we run 256 simulations and aggregate the results. Fig. 3 presents box plots of the cumulative reward; for simplicity, we focus on depth $d = 5$, as the insights are qualitatively similar for the two other values of d. The top panel groups these box plots by the exploration bonus c, while the bottom one groups the same plots by the heuristic used.

Several features are noticeable. First, the effect of the exploration bonus c is small and depends on the heuristic used. For the FW heuristic, the effect is negligible. One explanation for this features is that the FW heuristic is fairly strong on its own. Hence, the benefits from local changes to this policy are somewhat limited. For the random suppression heuristic, there may be some effect, but it seems to vary with α. When $\alpha = 0.2$, increased exploration benefits the random suppression heuristic, but when $\alpha = 0.5$, less exploration is preferred.

Table 2
Estimated effects for the MCTS hyperparameters.

	FW Heuristic		Random suppression	
(Intercept)	−353903.78***	(0.00)	−408969.13***	(0.00)
$\alpha = 0.5$	−3073.68	(0.37)	5335.56	(0.16)
$\alpha = 0.2$	7292.33*	(0.03)	4616.83	(0.23)
Depth = 5	34551.16***	(0.00)	4211.96	(0.14)
Depth = 10	35375.98***	(0.00)	3952.83	(0.17)
A2	−40434.84***	(0.00)	−976.72	(0.71)
$c = 10$			2857.04	(0.32)
$c = 50$			6900.73*	(0.02)
$c = 100$			9366.90**	(0.00)
$\alpha = 0.5$: Depth = 5	4412.72	(0.29)	58653.80***	(0.00)
$\alpha = 0.2$: Depth = 5	−11279.75**	(0.01)	41290.86***	(0.00)
$\alpha = 0.5$: Depth = 10	2989.4	(0.48)	65456.71***	(0.00)
$\alpha = 0.2$: Depth = 10	−11282.11**	(0.01)	47508.01***	(0.00)
$\alpha = 0.5$: A2	8467.03*	(0.01)	6960.33*	(0.02)
$\alpha = 0.2$: A2	20363.68***	(0.00)	−1457.66	(0.61)
Depth = 5 : A2	23627.58***	(0.00)		
Depth = 10 : A2	24100.29***	(0.00)		
$\alpha = 0.5$: $c = 10$			−2543.39	(0.53)
$\alpha = 0.2$: $c = 10$			−983.12	(0.81)
$\alpha = 0.5$: $c = 50$			−10139.50*	(0.01)
$\alpha = 0.2$: $c = 50$			−1250.75	(0.76)
$\alpha = 0.5$: $c = 100$			−16684.02***	(0.00)
$\alpha = 0.2$: $c = 100$			−674.51	(0.87)
Depth = 5 : A2			12733.89***	(0.00)
Depth = 10 : A2			12608.70***	(0.00)
R^2	0.06		0.14	
adj. R^2	0.06		0.14	

† Significant at $p < 0.10$.
*$p < 0.05$.
**$p < 0.01$.
***$p < 0.001$.
See Section 5.2 for details. Baseline should be interpreted as the value of $\alpha = 1$, Depth of 1, $c = 0$, without Algorithm 2 (based on genetic algorithm search heuristics).

Second, from the second panel, in all cases it seems that MCTS with the FW heuristic outperforms MCTS with the random suppression heuristic. The benefit of Algorithm 2 in action generation, however, is less clear. Using Algorithm 2 does seem to improve the performance of the random suppression heuristic in some cases. For the FW heuristic, there seems to be little benefit, likely because the FW heuristic already generates fairly good actions on its own.

To assess the statistical significance of these differences, we fit two separate linear regression model, one for the random suppression heuristic and one for the FW heuristic. We simplify the models using backward stepwise variable deletion, beginning with the full model with second-order interactions. Table 2 displays the results. In each column of values, the first value indicates the size of the effect, while the parenthesized value indicates the p-value of the effect (how statistically significant the effect is). One can check that the features we observed graphically from the box plots are indeed significant. Moreover, the random suppression heuristic demonstrates interesting second-order effects between the depth and α. The performance improves substantially when the depth is greater than one and we restrict the size of the searched actions. One explanation is that both parameters serve to increase the quality of the search tree, i.e., its depth, and the accuracy of the estimate at each of the searched nodes.

5.3. State space size

We first study the performance of our algorithms as the size of the state space grows. We simulate 256 runs of each of our methods with either 4 or 8 suppression teams, using our default values of the hyperparameters and varying $k \in \{8, 12, 16, 20, 30\}$.

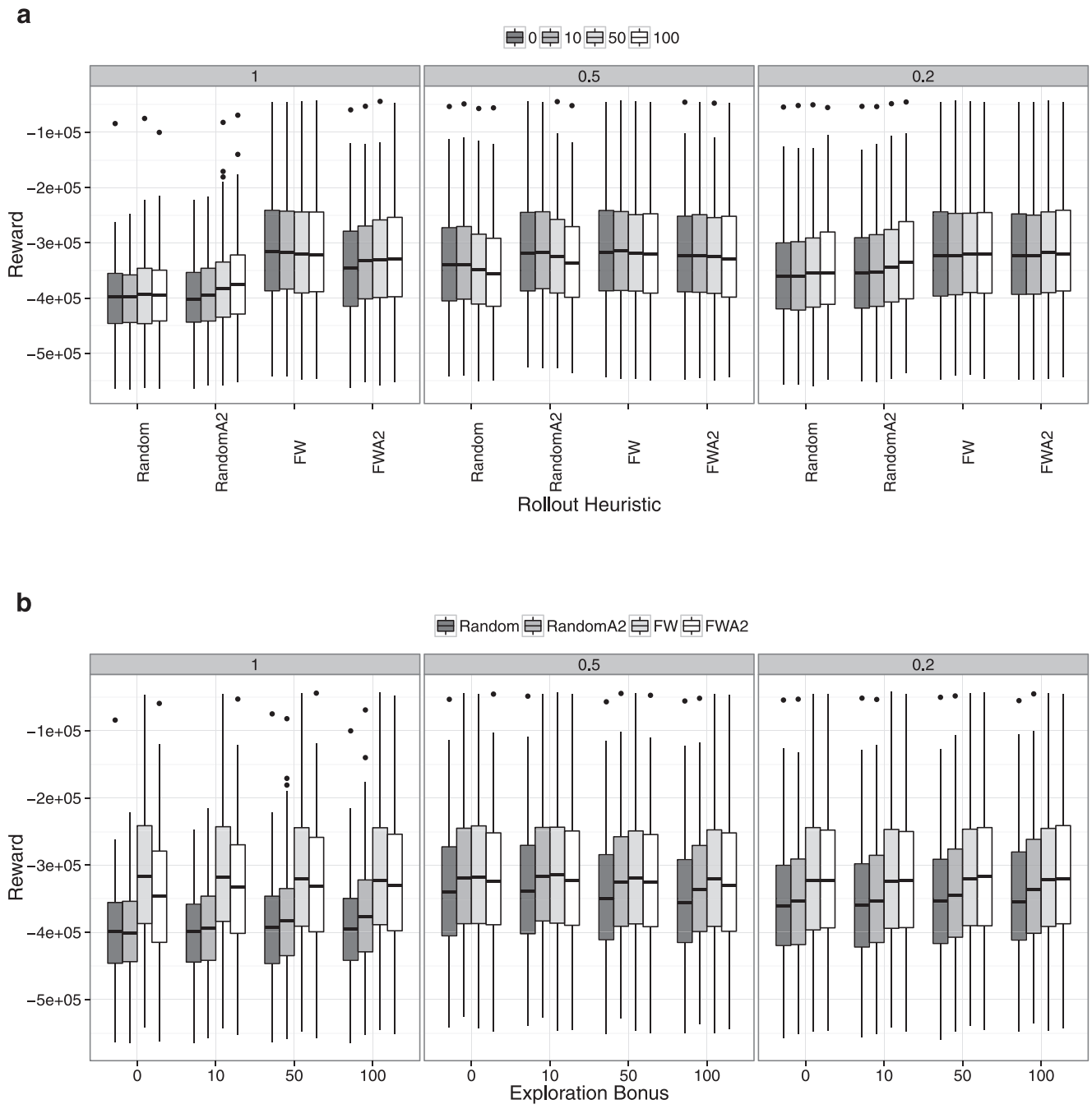


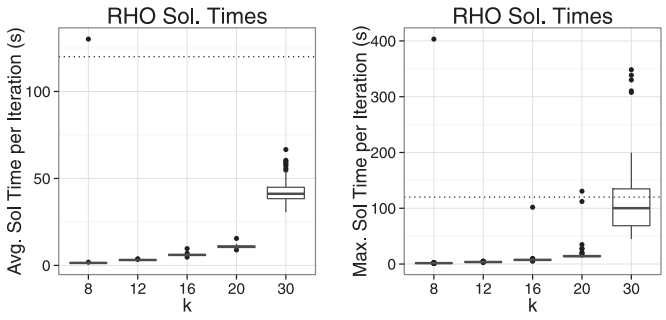
Fig. 3. The cumulative reward of the MCTS algorithm for various values organized by the exploration parameter c and the rollout heuristic. "Random" and "FW" refer to the random burn and Floyd–Warshall heuristics. "A2" indicates that we additionally use our Algorithm 2 (based on genetic algorithm search heuristics) in the action generation phase.

Figs. 4a and b show the average and maximum solution time per iteration of the RHO methodology when requesting at most 120 seconds of computation time. Notice that for most instances, the average time is well below the threshold – in these instances the underlying mixed-integer program is solved to optimality. For some instances, though, there are a few iterations which require much longer to find a feasible integer solution (cf. the long upper-tail in Fig. 4b). Consequently, we compare our RHO method to MCTS with 60 seconds, 90 seconds and 120 seconds of computation time.

A summary of the results is seen in Fig. 5a and b. Several features are evident in the plot. First, all three methods seem to outperform the FW heuristic, but there seems to only be a small difference between the three MCTS runs. The RHO method does seem to outperform MCTS method, especially for tail-hard instances. Namely, the lower whisker on the RHO plot is often shorter than the corresponding whisker on the MCTS plots.

To assess some of the statistical significance of these differences, we fit additive effects models for four and eight team separately. In both cases, there are no significant second-order interactions. The coefficients of the first-order interactions and corresponding p-values are shown in Table 3. The values suggest that the three MCTS methods are very similar and that the RHO method with a time limit of 60 seconds does outperform both.

In summary, these results suggest that MCTS and RHO perform comparably for small state spaces, but as the state space grows,



(a) Average (b) Maximum

Fig. 4. Average and maximum iteration solution time with 8 teams and a desired time limit of 120 seconds (dotted line). Instances which exceed their allotted time are typically not solved to optimality.

RHO begins to outperform MCTS, with the magnitude of the edge growing with the state space size.

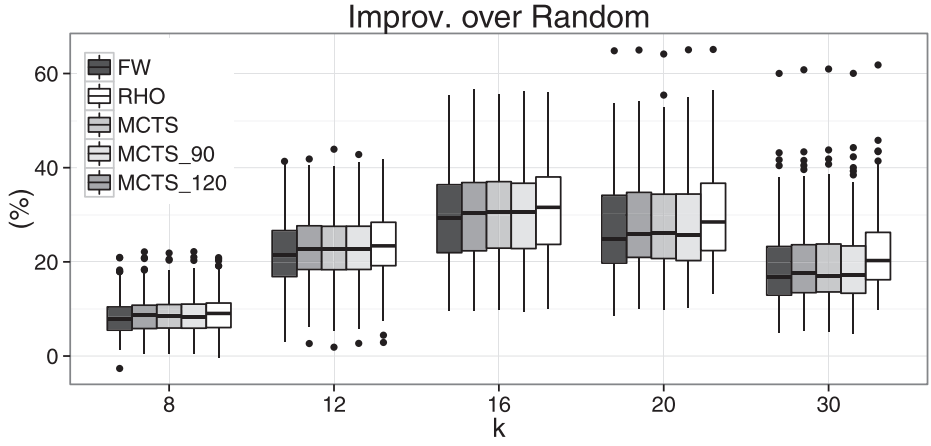
Table 3
Estimated effects for the percentage improvement relative to the random heuristic.

	$\|\mathcal{I}\| = 8$		$\|\mathcal{I}\| = 8$	
	Coefficient	p-value	Coefficient	p-value
(Intercept)	7.80***	(0.00)	16.94***	(0.00)
$k = 12$	14.48***	(0.00)	12.02***	(0.00)
$k = 16$	21.89***	(0.00)	9.76***	(0.00)
$k = 20$	19.48***	(0.00)	6.08***	(0.00)
$k = 30$	10.86***	(0.00)	−2.22***	(0.00)
MCTS (120 seconds)	0.87*	(0.03)	3.03***	(0.00)
MCTS (90 seconds)	0.88*	(0.03)	2.89***	(0.00)
MCTS (60 seconds)	0.83*	(0.04)	2.74***	(0.00)
RHO	2.22***	(0.00)	3.80***	(0.00)
R^2	0.47		0.21	
adj. R^2	0.47		0.21	

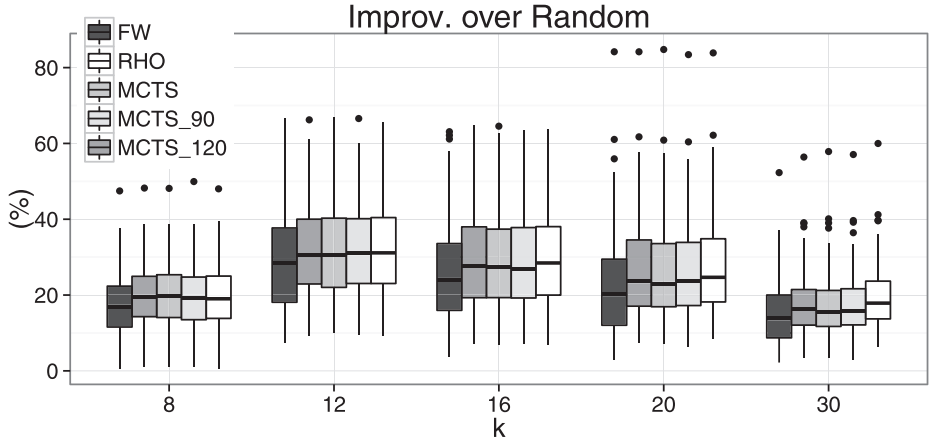
† Significant at $p < 0.10$.
*$p < 0.05$.
**$p < 0.01$.
***$p < 0.001$.
For details, see Section 5.3. The intercept should be interpreted as baseline of $k = 8$ with the FW heuristic.

5.4. Action space size

In this set of experiments, we are interested in comparing RHO and MCTS as the size of the grid (and thus the state space) is fixed, and the number of teams (and thus the action space size) increases. We will see that as the action space size grows large, RHO performs increasingly better than MCTS; moreover, the rate of increase is greater for larger grids.

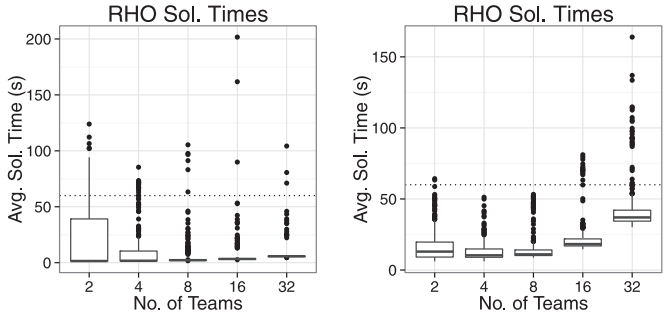


(a) $|\mathcal{I}| = 8$.



(b) $|\mathcal{I}| = 4$.

Fig. 5. Performance as a function of state space size.



(a) $k = 10$ (b) $k = 20$

Fig. 6. RHO average solution times.

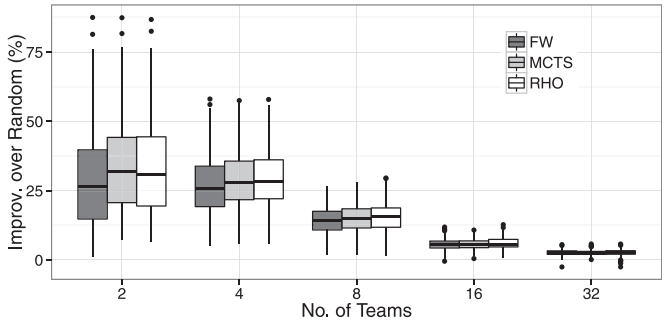


Fig. 7. Performance as function of number of suppression teams, $k = 10$.

Intuition suggests that the performance of the MCTS algorithm is highly dependent on the magnitude of the action branching factor, i.e., the number of actions available from any given state. As discussed in Section 4, without progressive widening, when the action branching factor is larger than the number of iterations, the MCTS algorithm will only expand the search tree to depth one. Even with progressive widening, choosing good candidate actions is critical to growing the search tree in relevant directions. A simple calculation using Stirling's approximation confirms that for the MDP outlined in Section 2, the action branching factor at time t is given by

$$\frac{N_B(t)^{|\mathcal{I}|}}{|\mathcal{I}|!} \approx \frac{\left(e \cdot \frac{N_B(t)}{|\mathcal{I}|}\right)^{|\mathcal{I}|}}{\sqrt{2\pi\,|\mathcal{I}|}}, \tag{12}$$

where $N_B(t)$ is the number of cells that are burning at time t. For even medium sized-instances, this number is extremely large. Consequently, in this section we study the performance of our algorithms with respect to the action branching factor.

We have already seen initial results in Section 5.2 suggesting that both our progressive widening and Algorithm 2 for action generation improve upon the base MCTS algorithm. It remains to see how MCTS with these refinements compares to RHO. We compute the relative improvement of the MCTS and RHO approaches over the randomized suppression heuristic with $k = 10$.

Recall that it is not possible to control for the exact time used by the RHO algorithm. Fig. 6a shows a box-plot of the average time per iteration for the RHO approach. Based on this plot, we feel that comparing the results to the MCTS algorithm with 60 seconds of computational time is a fair comparison. Fig. 7 summarizes the average relative improvement for each our methods for $k = 10$ as the number of teams varies. From this plot, we can see that the relative performance of all our methods degrades as the number of teams become large. This is principally because the randomized suppression heuristic improves with more teams. Although the FW heuristic is clearly inferior, the remaining approaches appear to perform similarly.

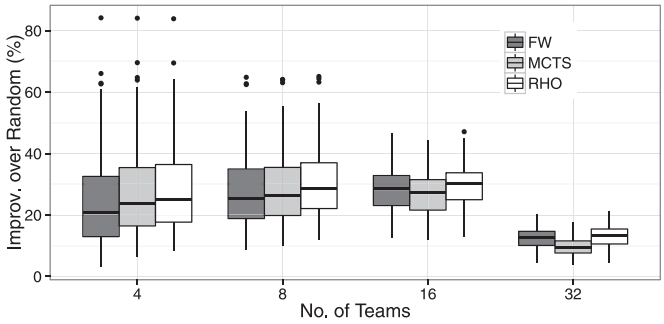


Fig. 8. Performance as a function of number of suppression teams, $k = 20$.

Table 4
Estimated effects for MCTS and RHO with varying number of teams.

	Coefficients	p-value
(Intercept)	23.40***	(0.00)
8 Teams	3.92***	(0.00)
16 Teams	4.79***	(0.00)
32 Teams	−10.84***	(0.00)
MCTS	3.01***	(0.00)
RHO	4.55***	(0.00)
8 Teams : MCTS	−1.93†	(0.10)
16 Teams : MCTS	−4.36***	(0.00)
32 Teams : MCTS	−5.86***	(0.00)
8 Teams : RHO	−1.49	(0.20)
16 Teams : RHO	−3.19**	(0.01)
32 Teams : RHO	−4.02***	(0.00)
R^2	0.36	
adj. R^2	0.36	

† Significant at $p < 0.10$.
*$p < 0.05$.
**$p < 0.01$.
***$p < 0.001$.
See also Section 5.4. The intercept should be interpreted as the performance of four teams under the FW algorithm.

To try and isolate more significant differences between the methodologies, we re-run the above experiment with $k = 20$. The results can be seen in Fig. 8 and the average solution times in Fig. 6b. In contrast with the previous experiment, RHO appears to outperform MCTS. Interestingly, although MCTS seems to outperform the FW heuristic for a small number of teams, it performs worse for more teams. To test the significance of these differences, we fit a linear regression model for the improvement over the randomized suppression heuristic as a function of the number of teams, the algorithm used, and potential interactions between the number of teams and the algorithm used. The results are in Table 4. The intercept value is the baseline, fitted value of the FW heuristic policy for four teams. From Table 4, we see that RHO outperforms FW for all choices of numbers of teams with statistical significance, but MCTS is statistically worse than FW with 16 or 32 teams.

In summary, the differences between RHO and MCTS become visible only when the grid size k is large, i.e., when the instances are sufficiently "difficult" to solve. It appears that although progressive widening and Algorithm 2 for action selection partially address the challenges of a large action state branching factor, the RHO approach is better suited to these instances.

5.5. Inexact calibration

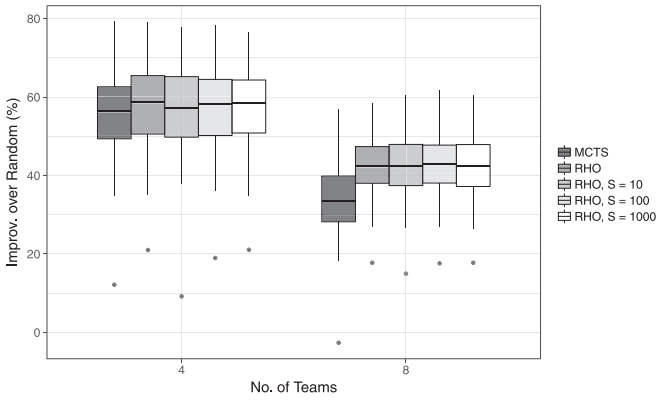
A major qualitative difference between RHO and MCTS is in their information requirements. MCTS only requires access to a

generative model, from which one samples transitions. On the other hand, RHO requires one to be able to specify certain parameters (e.g., the transition probabilities); in some applications, this may be difficult. In the prior experiments, the $\zeta_t(x, i)$ parameters of the RHO problem (4), which model the transmission rates of fire, were calibrated using the true transition probabilities (see the online supplement). One could argue that calibrating RHO using the true transition probabilities might unrealistically endow RHO with better performance and that it might be an unfair comparison to MCTS, which never works directly with the true transition probabilities.

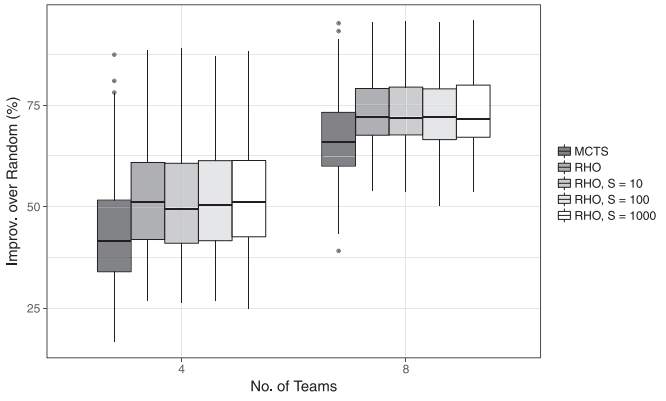
To investigate this, we consider another set of experiments where we compare MCTS and RHO against an "imperfect" version of RHO that is calibrated inexactly, using sampled transitions. We fix a number of samples S, we sample S transitions according to $P(x, y)$, and compute a sample-based estimate $\hat{P}(x, y)$ of $P(x, y)$, which we then use to calibrate $\zeta_t(x, y)$ as described in the online supplement. We test sample sizes $S \in \{10, 100, 1000\}$. We test grid sizes $k \in \{8, 16\}$ and suppression team sizes $|\mathcal{I}| \in \{4, 8\}$, and conduct 100 repetitions. We measure the performance of each policy relative to the randomized suppression heuristic. Note that the inexact RHO policies are exposed to the true dynamics, as are the exact RHO and MCTS policies; the difference now is that the exact RHO policy uses the true transition probabilities, whereas the sample-based RHO policy bases its decisions on the inexact transition probabilities. We note that in each repetition, the sample-based RHO policy is only trained once before the start of the problem horizon; the estimates $\hat{P}(x, y)$ are not dynamically updated with each period of the problem. We also emphasize that MCTS, as in the previous experiments, still has access to the exact generative model and is tested in an offline manner; it does not update its generative model with each period.

Fig. 9 shows the performance of MCTS, RHO and the inexactly calibrated RHO. From this figure, we can see that although there are some cases where the performance of RHO deteriorates when S is low, in general there is very little variation between different values of S, and the sample-based RHO exhibits similar performance to the exact RHO. To understand the reason behind this phenomenon, we examined the actions prescribed by the RHO policy. Empirically, we often observed that the solution to the MO problem (4) would assign suppression teams to burning cells that are on the boundary of the fire, as opposed to cells in the interior of the fire. Intuitively, such a policy should do well, irrespective of the precise transition dynamics: allocating a suppression team to a burning cell with non-burning neighbors is helpful because it prevents further new cells from igniting; on the other hand, allocating a suppression team to a burning cell completely surrounded by other burning cells is not helpful, because even if the team succeeds in suppressing the fire, the neighboring burning cells make it likely that the cell will re-ignite in the next period. This property of problem (4) seems to be insensitive to the transition probabilities used to calibrate the $\zeta_t(x, y)$ parameters and appears to be a feature of the deterministic continuous intensity dynamics used in problem (4). As a result, RHO can still perform well, even though the transition probabilities are not known with full accuracy.

Overall, the results of this experiment suggest that, in this particular DRA application, the requirement of exact knowledge of the transition dynamics can be relaxed with virtually no loss in performance. Of course, in other DRA problems one may observe reduced performance from using a sample-based RHO approach compared to the exact RHO approach. A more in-depth comparison of MCTS and RHO in this regime is an interesting direction for future research.



(a) $k = 8$.



(b) $k = 16$.

Fig. 9. Performance (measured as improvement over the random suppression heuristic) for MCTS, exact RHO and inexact RHO with $S \in \{10, 100, 1000\}$.

Table 5
Default parameters for MCTS.

Parameter	Value
Num. trajectories	20
Exploration bonus	20
Depth	10
Progressive widening, action space α	0.01
Progressive widening, state space α'	0.01
Progressive widening, action space k	5
Progressive widening, state space k'	5

6. Numerical comparisons for queueing network control

6.1. Background

For our experiments, we use the same custom implementation of MCTS in C++ used in Section 5 and for the fluid (RHO) approach, we use the results from Bertsimas, Nasrabadi, and Paschalidis (2015), which were obtained using a custom implementation of the solution approach of Luo and Bertsimas (1998) in C. All MCTS experiments were conducted on a 2.6 gigahertz quad-core computer with 16 gigabytes of RAM in a single-threaded environment.

In Table 5, we specify the parameters of the MCTS approach. These parameters were chosen through preliminary testing that suggested good performance and comparable timing behavior to the fluid method. For action generation, we consider a random action generation approach. For the rollout policy, we will consider the random policy and the $c\mu$ policy described in Section 4.2.

Table 6
Average long-run number of jobs in system for the criss-cross queueing network under different methods.

Parameter set	Random	$c\mu$	MCTS-random	MCTS-$c\mu$	Fluid
I.L.	0.678	0.679	0.681	0.677	0.678
B.L.	0.858	0.859	0.856	0.856	0.857
I.M.	2.167	2.170	2.168	2.160	2.162
B.M.	3.005	3.006	3.012	2.983	2.965
I.H.	10.359	10.470	10.430	10.546	10.398
B.H.	18.157	18.097	17.910	18.229	18.430

Table 7
Average long-run number of jobs in system for six-class queueing network under different methods.

Parameter set	Random	$c\mu$	MCTS-random	MCTS-$c\mu$	Fluid
I.L.	0.780	0.748	0.756	0.757	0.750
B.L.	0.965	0.928	0.938	0.939	0.923
I.M.	2.728	2.355	2.433	2.442	2.301
B.M.	3.807	3.235	3.319	3.320	3.024
I.H.	14.654	9.452	9.561	9.736	9.435
B.H.	24.695	18.167	18.425	17.882	15.670

In what follows, we will compare the MCTS and the fluid optimization approaches on the four example network studied in Bertsimas, Nasrabadi, and Paschalidis (2015). The insights that emerge from this computational study can be summarized as follows:

- MCTS and the fluid approach perform comparably on the criss-cross network (two servers, three job classes) and six-class network (two servers, three job classes per server).
- For the extended six-class network, MCTS tends to slightly outperform the fluid approach, while for the reentrant network, the fluid approach significantly outperforms MCTS. Overall, it seems that on large networks with complicated topologies, the fluid approach exhibits an edge over MCTS.
- In general, MCTS with the random rollout is able to significantly improve on the random policy, but MCTS with the $c\mu$ rollout is unable to improve on the basic $c\mu$ policy.

6.2. Criss-cross network

The first network that we study is the example criss-cross network shown in Fig. 2 in Section 2.2. We consider the six different sets of values from Bertsimas, Nasrabadi, and Paschalidis (2015) for the arrival rates λ_1 and λ_2 and the service rates μ_1, μ_2 and μ_3.

Table 7 displays the results. For this network, we can see that the methods are in general comparable and achieve very similar performance. In the **B.H.** case, both MCTS methods outperform the fluid method – for example, MCTS with random rollout attains a value of 17.910 compared to 18.430 for the fluid method, which is a difference of over 2% relative to the fluid method. In general, each MCTS variant does not appear to perform significantly differently from its underlying rollout policy; in some cases the MCTS method performs worse than its underlying rollout policy (e.g., MCTS-$c\mu$ for **I.H.** performs worse than $c\mu$ for the same parameter set), while in some cases MCTS offers a considerable improvement over the underlying rollout (e.g., MCTS-random for **B.H.** performs better than the random policy alone).

6.3. Six-class network

In this experiment, we consider the six-class network from Bertsimas, Nasrabadi, and Paschalidis (2015). The structure of the network is shown in Fig. 10. We consider the six different sets of parameter values from Bertsimas, Nasrabadi, and Paschalidis (2015).

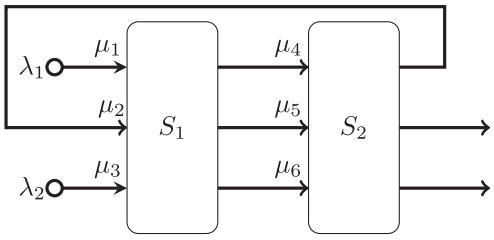


Fig. 10. Six-class network.

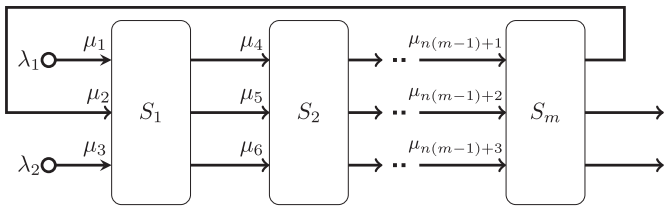


Fig. 11. The extended six-class network.

Table 8
Average long-run number of jobs in system for extended six-class queueing network under different methods.

Num. servers m	Random	$c\mu$	MCTS-random	MCTS-$c\mu$	Fluid
2	24.315	18.296	19.005	17.948	15.422
3	38.891	25.425	27.784	25.861	26.140
4	51.453	35.721	36.060	36.452	38.085
5	67.118	43.314	44.905	45.727	45.962
6	78.830	53.801	55.581	55.315	56.857
7	91.218	60.743	65.638	66.614	64.713

Table 7 displays the results. From this table, we can see that the fluid method generally performs better than MCTS. The most significant example is **B.H.** where the fluid method achieves an average number of jobs of 15.670 compared to 18.425 and 17.882 for MCTS-random and MCTS-$c\mu$, respectively. In general, MCTS with the random rollout significantly outperforms the random policy, but MCTS with the $c\mu$ rollout does not always improve on the performance of the $c\mu$ policy on its own and in general is slightly worse.

6.4. Extended six-class network

We now study a more complicated extension of the six class network. In particular, we assume that the servers repeat, as shown in Fig. 11. We assume that each server is associated with 3 classes (i.e., in Fig. 11, $n = 3$). The only external arrivals to the system are at classes 1 and 3, and the arrival rates for those classes are the same as in parameter set **B.H.** (see Bertsimas, Nasrabadi, & Paschalidis, 2015) for the six-class network. The service rates also follow the structure given in Bertsimas, Nasrabadi, and Paschalidis (2015).

Table 8 compares the different methods as the number of servers m varies from 2 to 7. From this table, we can see that aside from $m = 2$ and $m = 7$, MCTS with either the random or the $c\mu$ rollout policy performs better than the fluid method; for $m = 2$ and $m = 7$, the fluid method performs better. As in the six-class network, MCTS-random significantly improves on the random policy, but MCTS-$c\mu$ generally performs slightly worse than the basic $c\mu$ policy. It turns out for this network that the $c\mu$ policy, with the exception of $m = 2$, generally performs the best of all the methods.

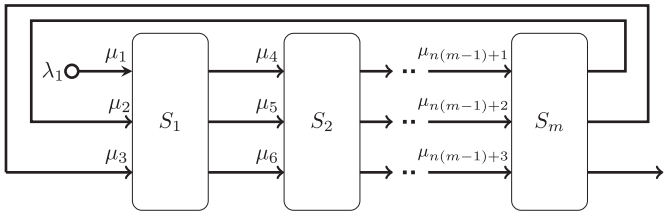


Fig. 12. Reentrant queueing network.

Table 9
Average long-run number of jobs in system for reentrant queueing network under different methods.

Num. servers m	Random	$c\mu$	MCTS-random	MCTS-$c\mu$	Fluid
2	25.076	18.199	17.991	17.796	15.422
3	37.908	26.937	27.233	26.257	25.955
4	50.940	34.934	37.194	36.734	32.014
5	64.273	42.387	45.629	44.866	40.113
6	79.165	51.945	57.189	54.682	48.781
7	92.672	59.958	67.531	63.696	54.711

6.5. Reentrant network

Finally, we consider a reentrant network, where after a job passes through the m servers, it re-enters the system at the first server and is processed again two more times (as a class 2 job and a class 3 job) before exiting the system. Fig. 12 displays the topology of this queueing network. The arrival rate of class 1 and the service rates of the classes are the same as for the extended six-class network of the previous section. We vary the number of servers from $m = 2$ to $m = 7$.

Table 9 displays the average number of jobs in the system under the different types of policies as the number of servers m varies from 2 to 7. For all values of m, the fluid policy performs the best, and significantly outperforms both versions of MCTS. For example, for $m = 7$, MCTS-$c\mu$ achieves a value of 63.696 while the fluid method achieves a value of 54.711 – a difference of over 14% relative to MCTS-$c\mu$. As in the extended six-class network, MCTS-random is able to provide a significant improvement over the baseline random policy, but MCTS-$c\mu$ generally does not improve on the basic $c\mu$ policy, especially for higher values of m.

7. Conclusion

In this study, we consider two dynamic resource allocation problems: one concerning tactical wildfire management, in which fire spreads stochastically on a finite grid and the decision maker must allocate resources at discrete epochs to suppress it, and one involving queueing network control, where a decision maker must determine which jobs are to be served by the servers as jobs arrive and pass through the network. We study two different solution approaches: one based on Monte Carlo tree search, and one based on rolling horizon optimization.

Our study makes two broad methodological contributions. The first of these contributions is to the understanding of MCTS: to the best of our knowledge, this study is the first application of MCTS to dynamic resource allocation problems. Our numerical results uncover some interesting insights into how MCTS behaves in relation to parameters such as the exploration bonus, the progressive widening parameters and others, as well as larger components such as the method of action generation and the rollout heuristic. Our results show that these components are highly interdependent—for example, our results show that the choices of action generation method and progressive widening factor become very important when the rollout heuristic is not strong on its own (e.g., a random heuristic) but are less valuable when the rollout heuristic is strong to begin with (e.g., the Floyd–Warshall heuristic in the case of the wildfire management problem). These insights will be valuable for practitioners interested in applying MCTS to other problems.

The second broad methodological contribution is towards the understanding of the relative merits of MCTS and RHO. Our results show that while both methodologies exhibit comparable performance for smaller instances, for larger instances, the mathematical optimization approach exhibits a significant edge. Initial evidence suggests this edge may be related more closely to action branching factor than the state space branching factor.

Acknowledgments

The authors thank the two anonymous referees for their helpful comments which have greatly improved the paper. This work is sponsored by the Assistant Secretary of Defense for Research and Engineering, ASD(R&E), under Air Force Contract #FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Government. The authors acknowledge Robert Moss for excellent research assistance. The work of the fifth author was supported by a PGS-D award from the Natural Sciences and Engineering Research Council (NSERC) of Canada.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.ejor.2017.05.032.

References

Acimovic, J., & Graves, S. (2012). Making better fulfillment decisions on the fly in an online retail environment. *Technical Report Working Paper*. Boston, MA: Massachusetts Institute of Technology.
Arneson, B., Hayward, R. B., & Henderson, P. (2010). Monte Carlo tree search in hex. *IEEE Transactions on Computational Intelligence and AI in Games, 2*(4), 251–258.
Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning, 47*(2–3), 235–256.
Avram, F., Bertsimas, D., & Ricard, M. (1995). Fluid models of sequencing problems in open queueing networks: An optimal control approach. *Institute for Mathematics and its Applications*, 199–234.
Bertsimas, D., Gupta, S., & Lulli, G. (2014). Dynamic resource allocation: A flexible and tractable modeling framework. *European Journal of Operational Research, 236*(1), 14–26.
Bertsimas, D., Nasrabadi, E., & Paschalidis, I. C. (2015). Robust fluid processing networks. *IEEE Transactions on Automatic Control, 60*(3), 715–728.
Bertsimas, D., Paschalidis, I. C., & Tsitsiklis, J. N. (1994). Optimization of multiclass queueing networks: Polyhedral and nonlinear characterizations of achievable performance. *The Annals of Applied Probability*, 43–75.
Bertsimas, D., & Stock Patterson, S. (1998). The air traffic flow management problem with enroute capacities. *Operations Research, 46*(3), 406–422.
Boychuck, D., Braun, W. J., Kulperger, R. J., Krougly, Z. L., & Stanford, D. A. (2008). A stochastic forest fire growth model. *Environmental and Ecological Statistics, 1*(1), 1–19.
Bracmort, K. (2013). Wildfire management: Federal funding and related statistics. *Technical Report*. Congressional Research Service.
Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., et al. (2012). A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games, 4*(1), 1–43.
Buyukkoc, C., Varaiya, P., & Walrand, J. (1985). The $c\mu$ rule revisited. *Advances in Applied Probability, 17*(1), 237–238.
Buzacott, J. A., & Shanthikumar, J. G. (1993). *Stochastic models of manufacturing systems*. Englewood Cliffs, NJ: Prentice Hall.
Ciancarini, P., & Favini, G. P. (2010). Monte Carlo tree search in Kriegspiel. *Artificial Intelligence, 174*(11), 670–684.
Ciocan, D. F., & Farias, V. (2012). Model predictive control for dynamic resource allocation. *Mathematics of Operations Research, 37*(3), 501–525.
Coquelin, P.-A., & Munos, R. (2007). Bandit algorithms for tree search. In *Proceedings of the twenty-third conference annual conference on uncertainty in artificial intelligence (UAI-07)* (pp. 67–74). Corvallis, Oregon: AUAI Press.
Couëtoux, A., Hoock, J.-B., Sokolovska, N., Teytaud, O., & Bonnard, N. (2011). Continuous upper confidence trees. In *Proceedings of the international conference on learning and intelligent optimization* (pp. 433–445). doi:10.1007/978-3-642-25566-3_32.

Coulom, R. (2007). Efficient selectivity and backup operators in Monte-Carlo tree search. In *Proceedings of the Computers and games* (pp. 72–83). Springer.

Dai, J. G. (1995). On positive harris recurrence of multiclass queueing networks: a unified approach via fluid limit models. *The Annals of Applied Probability*, 49–77.

Enzenberger, M., Muller, M., Arneson, B., & Segal, R. (2010). Fuego—an open-source framework for board games and Go engine based on Monte Carlo tree search. *IEEE Transactions on Computational Intelligence and AI in Games, 2*(4), 259–270.

Eyerich, P., Keller, T., & Helmert, M. (2010). High-quality policies for the canadian traveler's problem. In *Proceedings of the third annual symposium on combinatorial search*.

Floyd, R. W. (1962). Algorithm 97: Shortest path. *Communications of the ACM, 5*(6), 345. doi:10.1145/367766.368168.

Fried, J. S., Gilless, J. K., & Spero, J. (2006). Analysing initial attack on wildland fires using stochastic simulation. *International Journal of Wildland Fire, 15*, 137–146. http://dx.doi.org/10.1071/WF05027.

Gallego, G., & van Ryzin, G. (1994). Optimal dynamic pricing of inventories with stochastic demand over finite horizons. *Management Science, 40*(8), 999–1020.

Gelly, S., & Silver, D. (2011). Monte-Carlo tree search and rapid action value estimation in computer Go. *Artificial Intelligence, 175*(11), 1856–1875. http://dx.doi.org/10.1016/j.artint.2011.03.007.

Gurobi Optimization, Inc. (2013). Gurobi optimizer reference manual. http://www.gurobi.com.

Harchol-Balter, M. (2013). *Performance modeling and design of computer systems: Queueing theory in action*. Cambridge University Press.

Harrison, J. M. (1988). Brownian models of queueing networks with heterogeneous customer populations. In *Proceedings of the Stochastic differential systems, stochastic control theory and applications* (pp. 147–186). Springer.

Hu, X., & Ntaimo, L. (2009). Integrated simulation and optimization for wildfire containment. *ACM Transactions on Modeling and Computer Simulation, 19*(4), 19:1–19:29.

Kocsis, L., & Szepesvári, C. (2006). Bandit based Monte-Carlo planning. In *Proceedings of the European conference on machine learning* (pp. 282–293). Springer.

Luo, X., & Bertsimas, D. (1998). A new algorithm for state-constrained separated continuous linear programs. *SIAM Journal on control and optimization, 37*(1), 177–210.

Mišić, B., Sporns, O., & McIntosh, A. R. (2014). Communication efficiency and congestion of signal traffic in large-scale brain networks. *PLoS Computational Biology, 10*(1), e1003427:1–10.

Ntaimo, L., Arrubla, J. A. G., Stripling, C., Young, J., & Spencer, T. (2012). A stochastic programming standard response model for wildfire initial attack planning. *Canadian Journal of Forest Research, 42*(6), 987–1001. doi:10.1139/x2012-032.

Ntaimo, L., Gallego-Arrubla, J. A., Jianbang, G., Stripling, C., Young, J., & Spencer, T. (2013). A simulation and stochastic integer programming approach to wildfire initial attack planning.. *Forest Science, 59*(1), 105–117.

Pullan, M. C. (1993). An algorithm for a class of continuous linear programs. *SIAM Journal on Control and Optimization, 31*(6), 1558–1577.

Rubin, J., & Watson, I. (2011). Computer poker: A review. *Artificial Intelligence, 175*(56), 958–987. http://dx.doi.org/10.1016/j.artint.2010.12.005.

Tymstra, C., Bryce, R., Wotton, B., Taylor, S., & Armitage, O. (2010). Development and structure of prometheus: The Canadian wildland fire growth simulation model. *Information Report NOR-X-417*. Canadian Forest Service.

Van Mieghem, J. A. (1995). Dynamic scheduling with convex delay costs: The generalized c—mu rule. *Annals of Applied Probability, 5*(3), 809–833.

Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and Computing, 4*(2), 65–85. doi:10.1007/BF00175354.