Online Stochastic Optimization in the Large: Application to Kidney Exchange

Pranjal Awasthi
Machine Learning Department
Carnegie Mellon University
pawasthi@cs.cmu.edu

Tuomas Sandholm
Computer Science Department
Carnegie Mellon University
sandholm@cs.cmu.edu

Abstract

Kidneys are the most prevalent organ transplants, but demand dwarfs supply. *Kidney exchanges* enable willing but incompatible donor-patient pairs to swap donors. These swaps can include cycles longer than two pairs as well, and chains triggered by altruistic donors. Current kidney exchanges address *clearing* (deciding who gets kidneys from whom) as an offline problem: they optimize the current batch. In reality, clearing is an online problem where patient-donor pairs and altruistic donors appear and expire over time. In this paper, we study trajectory-based online stochastic optimization algorithms (which use a recent scalable optimal offline solver as a subroutine) for this. We identify tradeoffs in these algorithms between different parameters. We also uncover the need to set the batch size that the algorithms consider an atomic unit. We develop an experimental methodology for setting these parameters, and conduct experiments on real and generated data. We adapt the REGRETS algorithm of Bent and van Hentenryck for the setting. We then develop a better algorithm. We also show that the AMSAA algorithm of Mercier and van Hentenryck does not scale to the nationwide level. Our best online algorithm saves significantly more lives than the current practice of solving each batch separately.

1 Introduction

The role of kidneys is to filter waste from blood. Kidney disease is common around the world and often lethal. Death can be postponed by dialysis treatment, but the quality of life on dialysis is extremely low, and only 12% of dialysis patients survive 10 years [13]. Furthermore, dialysis treatment is expensive, and it is nearly impossible for dialysis patients to be productive in society.

The only permanent cure for kidney disease is a transplant. Kidney transplants outnumber all other organ transplants combined. Unfortunately, demand for deceased-donor kidneys dwarfs supply. In the United States alone, over 79,000 patients await a kidney. In 2008, 4,268 died waiting.[1]

People have two kidneys, but a person can live fine with just one. This has beget the practice of voluntary live-donor kidney transplants. However, usually willing donors (even from within the same family) are incompatible—due to blood type, tissue type, or other reasons—with the patient. Furthermore, buying and selling of organs is illegal in most countries, including the US.

Kidney exchanges have recently emerged as a way of mitigating this problem. They enable willing but incompatible donor-patient pairs to swap donors. These swaps can include cycles longer than two pairs as well, and chains triggered by altruistic donors. In 2007, Abraham, Blum, and Sandholm developed an algorithm that can optimally solve the NP-complete kidney exchange *clearing* problem (deciding who gets kidneys from whom) on a nationwide scale [1]. However, that algorithm, and, to our knowledge, all current kidney exchanges, address the clearing problem as an offline problem: whenever they optimize, they optimize the current batch as if it were the last. In reality, clearing is an online problem where patient-donor pairs and altruistic donors appear and expire over time.

We prove that no prior-free algorithm can do sufficiently well on this online problem. Therefore, we leverage the excellent probabilistic information that is available (about blood and tissue type distributions, etc.). Conceptually this information could be incorporated in the classical way into a multistage integer stochastic program which would then be solved to come up with the optimal plan. Unfortunately, such true stochastic optimization techniques are not capable of efficiently solving problems of the size required in this domain.[2] In contrast, online stochastic algorithms are suboptimal but scalable ways of solving stochastic integer programs [2, 5, 8, 9, 12]. The idea is to sample a subset of the future scenarios (trajectories), solve the offline problem on each of them, assign a score to each possible action, and select the action that is the best overall.

In this paper, we study such algorithms in the large, with application to kidney exchange. Our problem differs from

[1] *United Network for Organ Sharing (UNOS)*, www.unos.org.

[2] In fact, Algorithm 3 in this paper approximates such true stochastic optimization, and as we will show, even that approximate variation does not scale well.

most prior work in that the action space is enormous, and the sequences of events and actions are very long. We present approaches for dealing with that. We identify tradeoffs in these algorithms between different parameters, especially the lookahead depth and the number of sample trajectories. We also uncover the need to set the batch size that the algorithms consider an atomic unit. We develop an experimental methodology for setting these parameters, and conduct experiments on real and generated data. We adapt the REGRETS algorithm [2, 8] for the setting. We then develop a better algorithm. We also show that the AMSAA algorithm [9] does not scale.

Our best online algorithm saves significantly more lives (both in the sense of statistical significance and practical importance) than the current practice of solving each batch separately. One reason for this is that the current approach is myopic rather than anticipatory. Furthermore, the current myopic approach keeps the pool in a depleted state where the remaining patients tend to be difficult to match (for instance, they have difficult-to-match blood and/or tissue types). Part of the benefit of viewing the problem as an online problem is that such depletion should occur only to an effective extent.

1.1 Additional related research

Several authors have studied the problem of solving stochastic versions of various combinatorial problems. For a class of combinatorial optimization problems, Gupta et al. give a framework for converting an approximation algorithm for the offline problem into an approximation algorithm for the stochastic version of the problem [6, 7]. The main idea is to form a problem instance by sampling from the distribution of future scenarios, solving the deterministic problem on each scenario and customizing the solution once the actual scenario has been observed. That paper gives theoretical guarantees on the quality of the solution, provided that the approximation algorithm for the offline problem satisfies certain properties (which may not hold for kidney exchange). Their approach is computationally prohibitive for kidney exchange because it involves running the offline algorithm on the graph obtained by taking the union of a number of scenarios.

There has also been a bit of recent work on online kidney exchange. Ünver studies the problem under the objective of minimizing average waiting cost [11]. The paper ignores tissue type incompatibility and assumes patients do not expire. Assuming Poisson arrivals, the paper interestingly proves that certain dispatch rules—which do not use optimization—constitute an optimal policy. Zenios studies kidney exchange restricted to 2-cycles [13]. He models the exchange as a birth and death process, where no patients expire but long wait is penalized by a fixed cost. His objective is to maximize average quality-adjusted life years. The optimal policy is analytically derived: it uses no optimization algorithms. It is a simple policy that limits the number of patients that can take part in pairwise exchange. Patients not admitted to that exchange queue for altruistic donors (the wait time here is assumed to be zero).

2 Problem formulation

A kidney exchange can be represented as a directed graph. Figure 1 shows an example. Each patient-donor pair forms a vertex in the graph. Each edge $(u \rightarrow v)$ is assigned a weight that denotes the goodness of the donor kidney in vertex v for the patient in vertex u. Altruistic donors can be included as

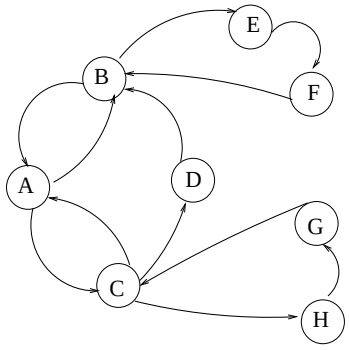


Figure 1: *Example kidney exchange graph.*

vertices with one edge of the kind described above; in addition, a dummy edge of weight zero is included from each such vertex to each donor-patient vertex. The weight of a cycle, w_c, is the sum of the weights of the edges participating in it. In the offline problem, the goal is to find a collection of vertex-disjoint cycles that has maximum collective weight, under the constraint that no cycle is of length greater than some cap L. A cap is used because longer cycles are logistically more difficult or infeasible (because all operations in the cycle have to be conducted simultaneously so no donor has the opportunity to back out) and because longer cycles are more likely to fail due to last-minute incompatibility detection. Most current kidney exchanges, and the proposed nationwide kidney exchange by the *United Network for Organ Sharing (UNOS)*, usually use $L = 3$. The problem is NP-complete for $L \geq 3$, but can be optimally solved in practice at a nationwide scale using a recent algorithm [1].

In this paper we present a solution to the stochastic version of this problem. At each time step, new vertices can appear and existing ones can expire; the distribution of this process is known (not in closed form but in a form that supports sampling). The objective is to choose a collection of cycles with maximum weight in expectation. Let G_t denote the graph at time t, and let P be the distribution according to which the graph changes over time. Let $C(L, G)$ be the set of all cycles in G no longer than L. Introduce a $0/1$ variable x_c for each cycle $c \in C(L, G)$. Let $C(L) = \cup_{t=0}^{T} C(L, G_t)$ and $V = \cup_{t=0}^{T} V_t$. The online problem is

$$\max_{c \in C(L,G_0)} E_P[\max_{c \in C(L,G_1)} E_P[\ldots \max_{c \in C(L,G_T)} \sum w_c x_c]]$$

$$subject\ to \sum_{c: v_i \in c} x_c \leq 1,\ \forall v_i \in V$$

$$with\ c \in \{0, 1\},\ \forall c \in C(L)$$

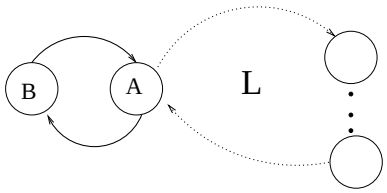
3 No prior-free online algorithm performs well

The usual approach to dealing with online problems in theoretical computer science is to take a prior-free approach. The algorithm tries to do as well as possible, and a hypothetical adversary picks the sequence of events. Performance is compared against a solution that is optimal in hindsight: the *competitive ratio* is the solution quality of the hindsight optimum (OPT) divided by the solution quality of the algorithm.

It is easy to show that in the kidney exchange problem no prior-free algorithm can do well:

Proposition 1 *No deterministic prior-free algorithm can achieve a competitive ratio better than* $L/2$.

Proof: We construct an instance on which no deterministic algorithm can achieve ratio better than $L/2$, see figure. Vertex



A enters up front and remains for the entire duration. Vertex B enters up front as well, but is only available for a limited time. After that the adversary may or may not bring in a cycle of length L that includes A. The algorithm has to decide whether to take the short cycle (i.e., 2-cycle A-B) for sure for a payoff of 2, or wait for the unsure long cycle with potential payoff L. If the algorithm decides to wait, the adversary does not bring in the long cycle, so the ratio would be $\frac{2}{0} = \infty$. If the algorithm decides to take the short cycle, the adversary brings in the long cycle, so the ratio would be $\frac{L}{2}$. The latter ratio is better. Thus the algorithm always takes the short cycle. ■

Proposition 2 *No randomized prior-free algorithm can achieve a competitive ratio better than* $\frac{2L-2}{L}$.

Proof: Consider the proof above. Now the algorithm can randomize between taking the short cycle (with some probability p) and waiting for the unsure long cycle (with probability $1-p$). If the adversary does not bring in the long cycle, the ratio is $\frac{2}{p \cdot 2}$; if he does, the ratio is $\frac{L}{p \cdot 2 + (1-p)L}$. The algorithm's best strategy is to make the adversary indifferent between these choices: $\frac{2}{p \cdot 2} = \frac{L}{p \cdot 2 + (1-p)L}$, i.e., $p = \frac{2L-2}{L}$. Thus the competitive ratio, $\frac{2}{p \cdot 2}$, is $\frac{2L-2}{L}$. ■

One familiar with competitive ratios might argue that the ratios in the above propositions are not too bad. However, losses that might be tolerable when measuring, say, computational resource waste, are not acceptable here since we deal in human lives. Even with $L = 3$, Proposition 2 shows that any prior-free algorithm might only save (on average over the algorithms randomization) 75% of the lives that can be saved.

4 Online stochastic algorithms

Because no prior-free algorithm can do sufficiently well in this domain, we want to leverage the probabilistic information that we have. Indeed there is excellent information available about the blood and tissue type distributions (and expiration rates, etc.). We will discuss these in detail in the experimental section. Conceptually these could be incorporated into a stochastic program which would then be solved. Unfortunately stochastic optimization is very far from scaling to problems of the size addressed here. Recently, sample trajectory-based online stochastic algorithms have attracted interest for solving stochastic integer programs [2, 5, 8, 12]. Unlike stochastic optimization, they are suboptimal and scalable. They are relatively easy to implement, fast, and can leverage any—in particular the best—available algorithms for solving the offline problem.

The main idea behind these algorithms is to sample a subset of the future scenarios (trajectories), solve the offline problem on each of them, assign a score to each possible action, and select the action which is the best overall. The following subsections present specific algorithms in this family. These algorithms are targeted to scale to the large.

4.1 Algorithm 1 (adaptation of REGRETS)

Let G_t be the graph representing the kidney exchange at time step t and P denote the distribution according to which the graph changes over time.

The first algorithm we discuss is a slight adaptation of the REGRETS algorithm [2, 8]. A straightforward implementation of REGRETS is not scalable here because in our problem an action would correspond to a *collection* of cycles—and the number of such collections is exponential in the number of cycles (which itself is $\Theta(|V|^L)$). This would make the action space prohibitively large: exponential in the size of the input. To overcome this, we instead assign a score to each cycle, and then finally select a set of cycles that has the maximum score.

Algorithm 1 (adaptation of REGRETS)
Input: Graph G_t, distribution P over future graphs until time T, and a non-negative constant δ
Output: A set of vertex-disjoint cycles $\{c_1, c_2, \ldots, c_n\}$ in G_t
1. For each cycle $c \in G_t$, set $score(c) \leftarrow 0$
2. Using P, generate m scenarios $\{\epsilon_1, \epsilon_2, \ldots, \epsilon_m\}$
3. For each scenario ϵ_i do
   4. $S \leftarrow$ solution of the offline problem on $\{G_t, \epsilon_i\}$
   5. For $c \in G_t \cap S$, set $score(c) \leftarrow score(c) + value(S)$
   6. For $c \in G_t - (G_t \cap S)$, set $score(c) \leftarrow score(c) - \delta$
7. Using another integer program, determine a set of vertex-disjoint cycles with maximum score, and return it

In Step 4, any offline solver can be used. In particular, we use the offline solver that was recently developed specifically for kidney exchanges [1]. It can optimally solve problems at the projected size of the US nationwide kidney exchange.

Step 6 is to ensure that a cycle that is optimal in a very small fraction of scenarios does not get selected. We experimented with different values of δ using the generated data set and methodology described later.[3] The results are shown

[3]For these experiments, the parameters (explained in detail later on in the paper in the experimental section) were set as follows. The lookahead was 10, the number of samples was 50, and batch size was 10. The death rate was set so that without a transplant, 12% of the patients survive 10 years.

in the following table. In the rest of the experiments in this

δ	Average number of lives saved	Standard deviation
0	244.3	3.36
8	**248.0**	**3.32**
15	245.7	3.45
20	242.3	4.87
50	243.6	3.12
100	242.8	4.12
500	208.3	3.79

paper, we used $\delta = 8$.

Algorithm 1 is not optimal. We provide a counter-example on the graph of Figure 1. Let the graph at time 0 contain the vertices $\{A, B, C, D\}$. After one time step, A expires and either vertices $\{E, F\}$ or vertices $\{G, H\}$ enter (with equal probability). In the former scenario the optimal action is to choose cycle $A - C$ and cycle $B - E - F$. In the latter scenario the optimal action is to choose cycle $A - B$ and cycle $C - G - H$. However, the optimal action at time 0 is to choose cycle $A - B - C - D$ since it has expected weight 4. The other two cycles at time 0 have expected weight $2 + \frac{3}{2}$ each. Hence, Algorithm 1 will not select the optimal action—because it is not optimal in any one of the trajectories. (This same example serves to prove that the original REGRETS algorithm, which would treat collections of cycles as actions, would also be suboptimal.)

4.2 Algorithm 2

The above counter-example motivated us to try to develop a better algorithm. The idea is to optimize the scenarios for each action (i.e., cycle) separately rather than optimizing each individual scenario separately:

Algorithm 2
Input: Graph G_t and distribution P over future graphs until time T
Output: A set of vertex-disjoint cycles $\{c_1, c_2, \ldots, c_n\}$ in G_t
1. Using P, generate m scenarios $\{\epsilon_1, \epsilon_2, \ldots, \epsilon_m\}$
2. For each cycle $c \in G_t$ do
    2. Set $score(c) \leftarrow 0$
    3. For each scenario ϵ_i do
        4. $S \leftarrow$ solution of the offline problem on $\{G_t - c, \epsilon_i\}$
        5. set $score(c) \leftarrow score(c) + value(S) + value(c)$
6. Using another integer program, determine a set of vertex-disjoint cycles with maximum score, and return it

We also include a dummy action in the set of possible actions at each step. It is for the case where choosing no cycle is the most valuable action. A dummy action has proven useful in prior research on online problems in other contexts [3]. We will show its usefulness in our domain in the experimental section of this paper.

For Step 5 of Algorithm 2, we also experimented with a variant where we do not include the term "$+value(c)$". It performed very similarly to the main version of Algorithm 2 shown in the pseudocode above.

4.3 Algorithm 3 (adaptation of AMSAA)

Recent theoretical results relate the solution quality of REGRETS-based algorithms to a quantity called the *Global Anticipatory Gap (GAG)* [8]. The analysis applies to both Algorithm 1 and 2. The GAG is a property of the problem and of the probability distribution, P, that describes how the problem changes over time:

$$GAG = E_P[max_c \sum_t \Delta_g(G_t)]$$

$\Delta_g(G_t)$ is the *anticipatory gap* of a state G_t. It is defined as

$$\Delta_g(G_t) = min_c \, \Delta(G_t, c)$$

Here $\Delta(G_t, c)$ is the expected local loss of state action pair (G_t, c):

$$\Delta(G_t, c) = E_P[O(G_t, \epsilon) - O(G_t, c, \epsilon)|G_t]$$

This is the difference in solution quality produced by choosing the best action at state G_t versus action c. Intuitively, $\Delta_g(G_t)$ is small if there exists an action in state G_t that is close to optimal in most of the scenarios. Due to the huge action space in kidney exchange, this will likely not be the case for a reasonable probability distribution P. Thus the $\Delta_g(s_t)$ are likely large, yielding a high GAG.

Therefore, we implemented the *AMSAA* algorithm [9], which specifically addresses a high GAG. It tries to the solve the multi-step problem directly by approximating it using sample trajectories and then solving it as a Markov Decision Process. Again, in our adaptation of it, individual cycles (rather than collections of cycles) are actions.

Algorithm 3 (adaptation of AMSAA)
Input: Graph G_t and distribution P over future graphs until time T
Output: A set of vertex-disjoint cycles $\{c_1, c_2, \ldots, c_n\}$ in G_t
1. Using P, generate m scenarios $\epsilon = \{\epsilon_1, \epsilon_2, \ldots, \epsilon_m\}$
2. For each state s do // Construct an approximate MDP
    3. if s is a final state, then $v(s) \leftarrow$ offline solution in state s
        else $v(s) \leftarrow$ average value of the offline solution over all scenarios in ϵ assuming that no vertex dies
4. Solve that MDP using tree search starting at vertex G_t
5. For each cycle $c \in G_t$, set $score(c) \leftarrow Q(G_t, c)$
6. Using another integer program, determine a set of vertex-disjoint cycles with maximum score, and return it

In Step 4, various search strategies can be used. As in the prior paper on AMSAA [9], in our experiments we use learning depth first search (LDFS) [4]

By default we again include a dummy action.

5 Experiments

We conducted experiments on data of two kinds. The first kind is a real data set that we obtained from the largest current kidney exchange in the US by helping them conduct exchange clearing. The data set corresponds to almost two years of real time. Because even the largest current real kidney exchange is relatively small—compared to the projected size of the US nationwide kidney exchange—that data set only has 158 pairs and 11 altruistic donors (and 4,086 edges).

We also benchmarked on larger problems. This second kind of data was generated by the most commonly used generator for the problem, which was designed by others to closely mimic the real-world population [10]. Briefly, patients are generated with a random blood type, gender, and probability of being tissue-type incompatible with a randomly chosen donor. Each patient is then assigned a donor with a random blood type. If the patient and donor are incompatible, they join the exchange. We generated a training set of 510 pairs, 25 altruistic donors, and 15,400 edges. We also generated a test set of the same size.

The real and generated data sets differ not only in size, but also the real data has much lower average degree in the exchange graph. This is because current greedy approaches to matching pairs in kidney exchanges leave a depleted pool where the remaining patients tend to be difficult to match (they have difficult-to-match blood and/or tissue types). Part of the benefit of viewing the problem as an online problem—as we do in this paper—is that such depletion should occur only to an effective extent.

We considered all transplants equally worthy, i.e., the weight of each edge in the exchange graph was set to one.

In our experiments, the exchange graph changes over time as follows. Initially the exchange starts with no vertices. At each time step a fixed number of vertices enter the graph. Each vertex has a probability, p, of dying at each time step. In each experiment, the value of p was set to match the fact that in reality, on average 12% of patients survive 10 years [13].

After some time, the exchange reaches steady state, i.e., the number of donors (and patients) entering roughly equals the number of donors (and patients) leaving. When there are not too many time steps left in the experiment, the algorithm's lookahead reaches all the way to the end of the time horizon. This might cause some special effects, so we exclude that ramp-down phase from any steady-state analysis.

In what follows, we first tune the parameters of the algorithms using the training set. We then compare the algorithms on the separate test set. In contrast, the real data set is relatively small; hence for experiments on real data we use only one set. Each reported number in each table is an average over ten runs, and the second number is the standard deviation.

5.1 Tuning the lookahead depth and the number of sample trajectories

All of the online algorithms work by sampling a number of scenarios into the future. This number is called the *sample size*. Also of interest is the parameter *lookahead*, which defines how many steps into the future the algorithm sees, i.e., how long each trajectory is. If an algorithm uses a large lookahead, it can make a more informed decisions. However, due to computational limitations one cannot sample all trajectories. There is a tradeoff between number of trajectories and lookahead. One straightforward reason for this tradeoff is that it takes more time to handle a deep trajectory than a shallow one. A more subtle reason—demonstrated by our experiments, e.g., Table 1—is that increasing lookahead increases the number of possible scenarios (exponentially), and therefore for a given number of sample trajectories, the coverage of possible trajectories by samples decreases. Therefore, increasing the lookahead too much actually decreases solution quality even if one keeps the number of sample trajectories constant!

We ran experiments to choose a good value for these two parameters for each of the algorithms separately. For these experiments we fixed the batch size to be 10 for the generated data set, and 5 for the real data set. (As explained above, the death rate was set so that 12% survive in 10 years.) Algorithm 3 did not scale to the generated data set, so we only ran it on the real data. The results are shown in Tables 1 and 2. They show that there are diminishing returns to the number of samples, as expected. They also show that increasing the lookahead too much (for a given number of samples) actually decreases solution quality.

Lookahead	Samples	Lives saved by Algorithm 1	Lives saved by Algorithm 2
5	10	223.8 ± 2.54	234.5 ± 3.25
5	20	224.6 ± 3.15	234.2 ± 3.27
5	50	228.3 ± 2.37	235.9 ± 2.97
10	10	235.4 ± 2.86	233.8 ± 4.12
10	20	240.7 ± 3.21	237.4 ± 3.74
10	50	$\mathbf{244.2 \pm 2.74}$	243.2 ± 3.29
20	10	237.3 ± 2.18	$\mathbf{250.1 \pm 3.17}$
20	20	238.6 ± 2.94	249.4 ± 3.25
20	50	240.1 ± 3.32	249.8 ± 3.64

Table 1: *Algorithms* 1 *and* 2 *on generated data.*

Lookahead	Samples	Lives saved by Algorithm 1	Lives saved by Algorithm 2	Lives saved by Algorithm 3
2	5	25.2 ± 1.23	25.45 ± 1.51	28.4 ± 4.33
2	10	27.1 ± 2.16	28.3 ± 1.83	29.6 ± 2.19
2	15	29.8 ± 1.74	30.4 ± 1.62	30.4 ± 2.19
4	5	30.5 ± 1.97	30.3 ± 1.36	28.8 ± 1.09
4	10	32.2 ± 2.21	33.1 ± 1.78	28 ± 1.16
4	15	$\mathbf{32.7 \pm 2.08}$	$\mathbf{34.1 \pm 1.72}$	$\mathbf{31.6 \pm 1.67}$
7	5	32.2 ± 1.43	32.3 ± 2.12	30.8 ± 2.28
7	10	31.9 ± 1.24	32.8 ± 1.41	29.4 ± 1.54
7	15	32.4 ± 1.58	32.2 ± 2.07	30.2 ± 1.78

Table 2: *Algorithms* 1, 2, *and* 3 *on real data.*

Based on these results, we picked reasonable values for the two parameters for each algorithm for each of the two kinds of data (these parameter values were chosen based on the best performance among the parameter combinations tested, as shown in bold in the tables). These parameter settings were used in the rest of the experiments.

5.2 Tuning the batch size

Another issue that we need to deal with in scaling online stochastic optimization into the large is that the horizon of possible events is too long to conduct meaningful lookahead on. To address this, we introduce the notion of batching, and the associated algorithm parameter which we call *batch size*. It defines how many arrivals the algorithm considers as one atomic event. In principle, for an ideal algorithm the best batch size is one, but because the algorithms are computationally limited and therefore cannot look ahead arbitrarily deeply (while ensuring reasonable coverage of the possible trajectories with sample trajectories), it turns out to be better to use a batch size significantly greater than one.

In practice, batch size determines how often the algorithm is run. For example, assuming that 10 vertices enter the system every month and that the batch size is 20, the algorithm will be run every two months. Therefore, one step of the algorithm will correspond to a time period of two months, and the death rate is adjusted accordingly to maintain the real-world fact that 12% survive 10 years.

We ran a set of experiments to determine the appropriate batch size for each of the algorithms—including the *offline-based algorithm* which runs the offline algorithm for each batch as if it were the last—for each of the two kinds of data. Tables 3 and 4 show that the number of lives saved increases with batch size up to some point and then decreases. For each algorithm and each kind of data, we ran experiments to sandwich this best batch size, and we will use that batch size in the remaining experiments. As one might expect, the best batch size is significantly larger for the offline-based algorithm since the algorithm itself is myopic.

Batch size	Lives saved by Algorithm 1	Lives saved by Algorithm 2
5	109.3 ± 2.31	108.6 ± 3.45
10	111.2 ± 2.75	110.4 ± 3.52
12	$\mathbf{116.4 \pm 3.12}$	115.8 ± 3.25
15	115.8 ± 3.22	118.2 ± 3.73
18	110.2 ± 2.92	$\mathbf{119.1 \pm 3.21}$
20	110.5 ± 3.1	116.3 ± 3.14

Batch size	Lives saved by offline-based algorithm
20	112.6 ± 2.94
40	$\mathbf{115.4 \pm 3.89}$
45	111.8 ± 4.36
50	115.2 ± 3.04
60	105 ± 4.02

Table 3: *Algorithms 1 and 2 (Left), and the offline-based algorithm (Right) on generated data.*

Batch size	Lives saved by Algorithm 1	Lives saved by Algorithm 2	Lives saved by Algorithm 3	Lives saved by offline-based algorithm
3	16.2 ± 2.21	16.4 ± 2.65	16.4 ± 2.06	9.2 ± 4.13
5	20.4 ± 2.34	21.3 ± 2.76	21 ± 1.69	18 ± 3.12
10	$\mathbf{25.8 \pm 2.4}$	24.1 ± 2.31	22.2 ± 2.57	20.8 ± 3.01
15	25.2 ± 2.66	$\mathbf{26 \pm 3.1}$	$\mathbf{25.2 \pm 3.55}$	$\mathbf{23.4 \pm 2.77}$
17	24.3 ± 2.98	25.8 ± 2.63	22.8 ± 2.46	22.8 ± 2.56
20	22.6 ± 2.57	25.2 ± 2.7	21.8 ± 2.74	22.4 ± 2.62

Table 4: *Algorithms 1, 2, and 3, and the offline-based algorithm on real data.*

5.3 Studying the impact of the dummy action

As discussed, we supplemented the action set in Algorithms 2 and 3 with a dummy action so the algorithms can decide to make no matches and wait. On the downside, this increases the size of the space of possible plans and thus (to an extent) decreases the coverage of possible trajectories by actual sample trajectories (for the fixed lookahead and number of samples). (Algorithm 1 and the offline-based algorithm do not need a dummy because they can choose to wait anyway.)

To study the impact of the dummy action, we ran experiments with it included versus removed. In each experiment we conducted 10 runs and report average performance. We ran the generated test exchange for 51 virtual months, with 10 vertices joining the graph per month. Solution quality of Algorithm 2 was 2.2% worse when the dummy action was removed (standard deviation 1.2%). We simulated the real data for 31 virtual months with 5 vertices joining per month. Solution quality of Algorithm 3 was 1.8% worse when the dummy action was removed (standard deviation 1.0%). So, the dummy action indeed helps save more lives. This is consistent with prior experiments with dummy actions in other domains [3]. While in our experiments the dummy action helps only a small amount, that amount is statistically highly significant: in none of the 20 total runs did including the dummy action perform worse than not including it. For all these reasons, for the rest of the experiments we leave the dummy actions in.

5.4 Comparing the algorithms

After tuning each algorithm as described in the three subsections above, we compared the algorithms on the real data and on a separate test set of the generated data. We ran the generated exchange for 51 virtual months, with 10 vertices joining the graph per month. The real data was simulated for 31 virtual months with 5 vertices joining per month. Each experiment was repeated 10 times, and averages are reported.

The results are shown in Figures 2 and 3. The online algorithms outperform the offline-based algorithm. Algorithm 2 yields better results than Algorithm 1.

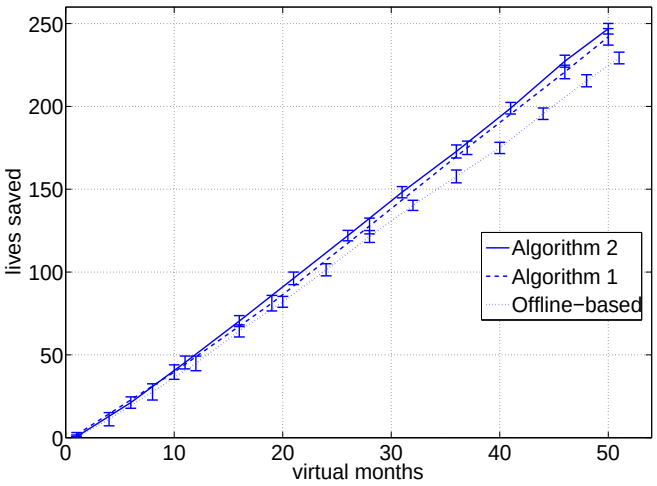


Figure 2: *Algorithms 1 and 2, and the offline-based algorithm on generated data. Standard deviations are also shown for each data point. Steady state is from month 26 to month 35.*

In steady state on the generated data, Algorithm 2 outperformed the offline-based algorithm by 13.0% (standard deviation 2.2%). Algorithm 1 outperformed the offline-based algorithm by 10.9% (standard deviation 2.0%).

Algorithm 3 did not scale to the generated data, so again it was tested only on the real data. Overall, the results on the real (significantly smaller) data set are less conclusive. For one, it is too short to establish a steady state of meaningful length. Overall, Algorithm 2 outperformed the offline-based algorithm by 6.5% (standard deviation 1.7%). Algorithm 1 outperformed the offline-based algorithm by 3.2% (standard deviation 1.7%). Algorithm 3 outperformed the offline-based algorithm by 0.7% (standard deviation 0.1%).

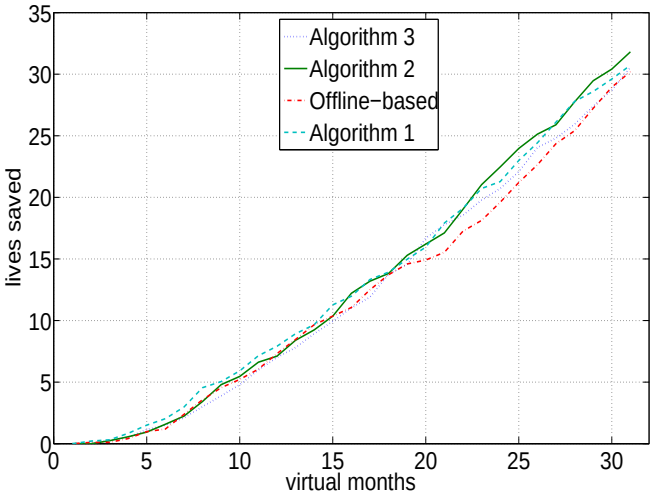


Figure 3: *Algorithms 1, 2, and 3, and the offline-based algorithm on real data.*

While these improvements may seem small as percentages, these savings represent human lives and are thus very significant from a practical perspective (and also statistically significant). Furthermore, the improvement over current practice is even greater because in current practice the batch size for the offline-based approach is ad hoc unlike the tuned batch size for the offline-based approach here.

6 Conclusions and future research

We showed how online anticipatory algorithms can save a significant number of lives (both in the sense of statistical significance and practical importance) via better online kidney exchange clearing. Real-world nationwide kidney exchanges have already adopted the scalable optimal offline algorithm that our group developed for kidney exchange in 2007 [1]. The online anticipatory algorithms presented in this paper thus have the real potential to serve as the next-generation clearing algorithms for the world's largest kidney exchanges.

From the perspective of online anticipatory algorithms, our problem differs from most prior work in that the action space is enormous. We presented approaches for dealing with that. We identified tradeoffs in these algorithms between different parameters such as lookahead and the number of samples. We also uncovered the need to set the batch size that the algorithms consider an atomic unit. We developed an experimental methodology for setting these parameters, and conducted experiments on real and generated data. We adapted the REGRETS algorithm for the setting. We then developed a better algorithm. We also showed that the AMSAA algorithm does not scale.

Future research includes extending the algorithms to even larger graphs. Since the current algorithms are still somewhat computationally intensive, one approach would be to partition the exchange graph into approximately independent components (based on low weight graph cuts) and run the algorithm on each of them separately. This is promising for kidney exchange since the graph is very sparse in practice.

Also, there is structure in the kidney exchange problem. (For example, the feasibility of an action depends on the feasibility of other actions.) It would be interesting to use the structure to design better and/or faster algorithms; the current algorithms are not specific to kidney exchange.

Acknowledgments

This work was supported by National Science Foundation grant IIS-0427858 and by the Carnegie Mellon University Center for Computational Thinking, which is funded by Microsoft Research. We thank Intel Corporation and IBM for machine gifts. We thank Luc Mercier and the anonymous reviewers for comments on earlier versions of this paper.

References

[1] David Abraham, Avrim Blum, and Tuomas Sandholm. Clearing algorithms for barter exchange markets: enabling nationwide kidney exchanges. *ACM EC-07*.

[2] Russell Bent and Pascal van Hentenryck. Regrets only! Online stochastic optimization under time constraints. *AAAI-04*.

[3] Russell Bent and Pascal van Hentenryck. Waiting and relocation strategies in online stochastic vehicle routing. *IJCAI-07*.

[4] Blai Bonet and Hector Geffner. Learning depth-first search: A unified approach to heuristic search in deterministic and non-deterministic settings, and its application to MDPs. *ICAPS-06*.

[5] Craig Boutilier, David Parkes, Tuomas Sandholm, and William Walsh. Expressive banner ad auctions and model-based online optimization for clearing. *AAAI-08*.

[6] Anupam Gupta, Martin Pál, R. Ravi, and Amitabh Sinha. Boosted sampling: approximation algorithms for stochastic optimization. *STOC-04*.

[7] Anupam Gupta, Martin Pál, R. Ravi, and Amitabh Sinha. What about Wednesday? Approximation algorithms for multistage stochastic optimization. *APPROX-RANDOM-05*.

[8] Luc Mercier and Pascal van Hentenryck. Performance analysis of online anticipatory algorithms for large multistage stochastic integer programs. *IJCAI-07*.

[9] Luc Mercier and Pascal van Hentenryck. Amsaa: A multistep anticipatory algorithm for online stochastic combinatorial optimization. *CPAIOR-08*.

[10] S Saidman, A Roth, T Sönmez, U Unver, and F Delmonico. Increasing the opportunity of live kidney donation by matching for two and three way exchanges. *Transplantation*, 81, 2006.

[11] Utku Unver. Dynamic kidney exchange. Mimeo, 2007.

[12] Pascal van Hentenryck, Russell Bent, and Yannis Vergados. Online stochastic reservation systems. *CPAIOR-06*.

[13] Stefanos A. Zenios. Optimal control of a paired-kidney exchange program. *Management Science*, 48(3), 2002.