AMSI
AUSTRALIAN MATHEMATICAL SCIENCES INSTITUTE

National Collaboration in the Mathematical Sciences

Applying Approximate Dynamic Programming to the Elevator Dispatching Problem

Luran Li
Supervisor: Dr Michael Bulmer
University of Queensland

February 2014

1 Introduction

This project investigates the application of the optimisation technique approximate dynamic programming to the elevator dispatching problem, which asks for a policy that schedules a group of elevators in a building to serve passenger requests. The complexity of the problem increases drastically when the number of floors is of the magnitude of a skyscraper and a considerable amount of passengers are being transported. Although this problem is commonly formulated using stochastic dynamic programming, researchers have approached it via evolutionary optimisation techniques, such as genetic algorithm [1, 3, 4]. In this project, I have attempted to solve the problem using approximate dynamic programming [2] that adopts the stochastic dynamic programming formulation.

A solution to the elevator dispatching problem is comprised of the action of each elevator at every decision point, which we call time step, given the state of the system. A set of fixed operational rules are typically used in the conventional elevator systems, whilst more dynamic strategy may be deployed in modern systems due to the advent of the computer. I have attempted this approach by programming a solver that produces schedules for the elevators.

The rest of the paper consists of the formulation of the problem, the method of approximate dynamic programming, and the result from experiments.

2 Formulation

A standard dynamic programming formulation comprises (1) parameters that define a specific instance of the problem, (2) system states that describe the system at a given time step, (3) decisions that need to be made at each state, and (4) objective that is the goal of the optimisation.

For the elevator dispatching problem discussed in this project, each part is formulated as following.

(1) Parameters

N floors ($N \leq 200$)

M elevators ($M \leq 8$)

Elevator capacity: P passengers ($P = 20$)

T time steps

The constant T determines the temporal horizon to be considered in the solution.

(2) State

$$S_t = (R_{ta}, D_{tb})$$

where

$t = 0, 1, \ldots, T$

$a = (\text{next floor}, \text{arrival time}, \text{carrying } k \text{ passengers to floor } j)$

$b = (i, j)$ request from floor i to floor j

$R_{ta} =$ number of elevators with attribute a at time t

$D_{tb} =$ number of passengers with attribute b at time t

The use of the number of elevators/passengers with certain attributes provides an easy transition function between states.

(3) Decision

$$x_t = (x_{tad})$$

where

$t = 0, 1, \ldots, T$

$a =$ attribute for elevator

$d = (i, j)$ action to serve a request from floor i to floor j

(4) Objective

$$V_t(S_t) = \min_{x_t}(C(S_t, x_t) + \gamma \mathbb{E}\{V_{t+1}(S_{t+1}))|S_t\})$$

where

$C(S_t, x_t)$ is the cost function of time taking x_t in S_t

$S_{t+1} = S(S_t, x_t, W_{t+1})$

$\mathbb{E}\{V_{t+1}(S_{t+1})|S_t\}$ is the expectation taking x_t in S_t

γ is a discount factor $(0 < \gamma < 1)$

The transition function $S(\cdot)$ takes the state and the decision at time t, as well as the new requests W_{t+1} arriving at $t + 1$, and returns the next state S_{t+1}.

The discount factor γ is introduced to compensate for the inaccuracy of our prediction in the future states.

3 Method

In dynamic programming, we would solve

$$V_t(S_t) = \min_{x_t}(C(S_t, x_t) + \gamma \mathbb{E}\{V_{t+1}(S_{t+1}))|S_t\})$$

recursively, where the expectation of the value in the future states is evaluated as

$$\mathbb{E}\{V_{t+1}(S_{t+1}))|S_t\} = \sum_{W_{t+1}} \mathbb{P}(S_{t+1}|S_t, x_t)V_{t+1}(S_{t+1})$$

which uses a transition matrix $\mathbb{P}$ that consists of the probabilities of a transition between any two states.

For instance, to solve $V_0(S_0) = \min_{x_0}(C(S_0, x_0) + \gamma \sum_{W_1} \mathbb{P}(S_1|S_0, x_0)V_1(S_1))$, we have to not only consider all possible decisions x_0 and new requests W_1, but also have the values of V_1 for all possible states S_1. This requires the values of V_{t+1} are obtained before evaluating V_t, and therefore we would program by starting at V_T and proceeding backwards to V_0.

With the prevalent computing power, a problem instance of small size, such as 10 floors, can be solved in a negligible amount of time. For large problems, such as 100 floors, this standard approach becomes insufficient due to the huge space of new requests, states, and decisions.

Approximate dynamic programming is an iterative method that tackles this computational problem by reducing the space we examine, and improving the solution over iterations. Instead of considering multiple new requests per time step, we generate a sample of future requests W_t for $t = 0, 1, \ldots, T-1$ at the beginning of each iteration. Similarly, we choose a small set of decisions to consider, which can be generated randomly or according to certain policies. Consequently, the number of states to visit is significantly reduced due to the decrease of new requests and decisions to examine. Listing 1 shows the structure of the algorithm.

Listing 1: Algorithm

$$\overline{V}^n = \text{estimation in iteration } n$$

$$\mathscr{X}^n = \text{set of decisions in iteration } n$$

Step 0 Initialise

Step 0a Initialise $\overline{V}^0$, e.g. $\overline{V}^0 = \inf$

Step 0b Choose an initial state S^1

Step 0c Set iteration counter $n = 1$

Step 1 Generate a sample path W^n

Step 2a Solve

$$\hat{v}^n = \min_{x \in \mathscr{X}^n} (C(S^n, x) + \gamma \mathbb{P}(s'|S^n, x)\overline{V}^{n-1}(s'))$$

Step 2b Update value function estimation

$$\overline{V}^n(S) = \begin{cases} \hat{v}^n, & S = S^n \\ \overline{V}^{n-1}, & \text{otherwise.} \end{cases}$$

Step 2c Update state $S^n = S(S^n, x^n, W^n)$ where x^n is the solution to $\hat{v}^n$

Step 2d Go to step 2a

Step 3 Update $n = n + 1$. Go to step 1 if $n <$ MAX_ITER

4 Experimental Result

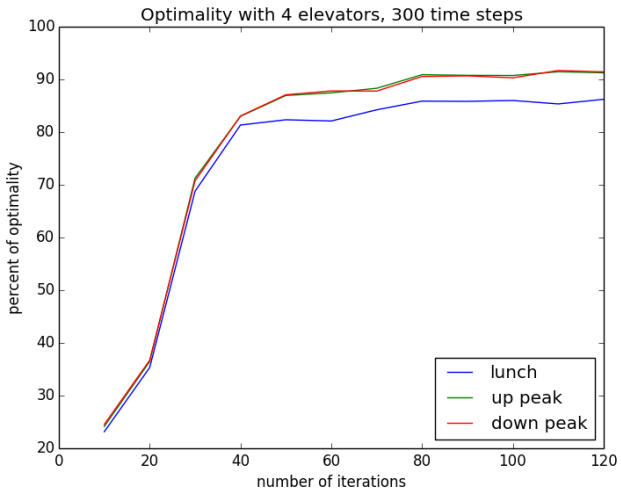
Effectiveness and efficiency are the two common factors that characterise an optimisation algorithm, that is, how good the solution is and how fast a program solves the problem.

I wrote two computer programs: one uses dynamic programming that always returns the optimal solution as it explores all possible states; the other uses approximate dynamic programming, of which the result is compared with the optimal solution.

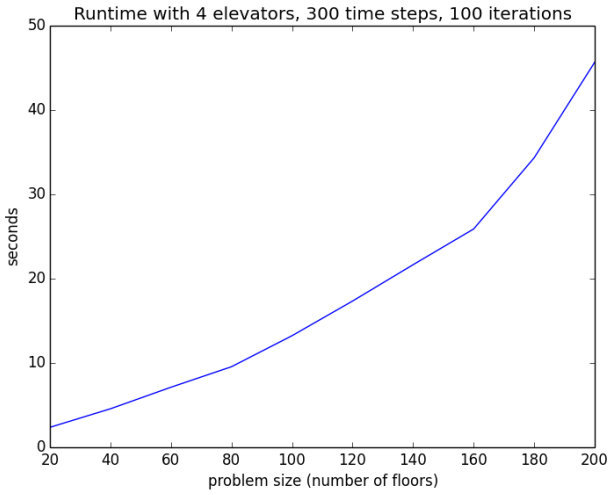
The traffic data was randomly generated by a Poisson process, which comprises the scenarios of up peak, down peak, and lunch.

To test effectiveness, I experimented with a problem instance of 10 floors, 4 elevators, 300 time steps. The instance was solved to optimality by the first program, and then solved using various iterations by the second program. The plot below shows the

approximate dynamic programming solver converges after about 80 iterations, up to 91% optimality for up/down peak traffic, and 85% for lunch.



To test efficiency, the approximate dynamic programming solver was run against problem instances of 4 elevators and 300 time steps with the number of floors up to 200. All instances are solved under 1 minute on my laptop with moderate computing power. This can be interpreted as, if we take 1 time step as 1 second in real time, then we are able to spend less than 1 minute planning a schedule for the next 5 minutes.



Postal Address: 111 Barry Street
c/- The University of Melbourne
Victoria 3010 Australia

Email: enquiries@amsi.org.au
Phone: +61 3 8344 1777
Fax: +61 3 9349 4106
Web: www.amsi.org.au

5 Conclusion

The elevator dispatching problem can be formulated using stochastic dynamic programming. The time to solve the problem, required by the standard approach, grows exponentially due to the huge space of uncertainties, states, and decisions. To tackle this computational difficulty, approximate dynamic programming considers a small subset of the problem space, and improves its solution iteratively. The effectiveness and efficiency of this technique are justified by experimental results, which meets the expectation in a real-world situation.

I would like to thank AMSI for providing me this scholarship and my supervisor Dr Michael Bulmer for his patience and invaluable advice on the project. The Big Day In event was the highlight of the Vacation Research Scholarship for me, where I was introduced to various topics in different mathematical areas, and had good conversations with the students from other universities.

References

[1] P. Cortés, J. Larrañeta, and L. Onieva. Genetic algorithm for controllers in elevator groups: analysis and simulation during lunchpeak traffic. *Applied Soft Computing*, 4(2):159–174, 2004.

[2] W. B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality, 2nd Edition (Wiley Series in Probability and Statistics)*. Wiley, 2011.

[3] J. Sorsa, M.-L. Siikonen, and H. Ehtamo. Optimal control of double-deck elevator group using genetic algorithm. *Int. Trans. Oper. Res.*, 10(2):103–114, 2003.

[4] T. Tyni and J. Ylinen. Evolutionary bi-objective optimisation in the elevator car routing problem. *European J. Oper. Res.*, 169(3):960–977, 2006.