SMM 641
Revenue Management and Pricing

Week 2: Introduction to DP and Multi Fare Allocation

An Introduction to Dynamic Programming

Quantity Based Revenue Management Part 2:

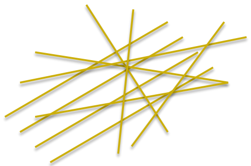
Capacity Allocation for Multiple Fare Classes

Today: Dynamic Programming

- A Dynamic Programming problem is an optimization problem in which decisions are given sequentially over several time periods.
- The periods are linked, e.g., actions taken at any period impact the available decisions and rewards in subsequent periods.
- It involves breaking a large problem down into smaller problems and then solving each small problem in turn.
- By solving the small problems we find the optimal solution to the large problem.

Introductory Example

Consider the following game*:



- Setup: A pile of 10 toothpicks
- Playing against a computer
- Game consists of rounds. The sequence of events is as follows:
  - You start first. You can pick either 1 or 2 toothpicks from the pile.
  - Computer moves next. Picks 1 with probability ½ and picks 2 with prob ½.
  - Game proceeds until all toothpicks are removed from the pile.
- If you hold the last toothpick, you win and receive £20. Otherwise the computer wins and you get nothing.

* Source: Paat Rusmevichientong

Introductory Example

Observations:

- If the game starts with 1 or 2 toothpicks, we win!
- If game starts with 0 toothpicks, we lose.
- Suppose we start round *k* with $S_k \geq 3$ toothpicks and let S_{k+1} be the number of toothpicks at the beginning of the next round.
  - If we pick 1 toothpick, then $S_{k+1} = S_k - 1 - X_k$
  - If we pick 2 toothpicks, then $S_{k+1} = S_k - 2 - X_k$

  where $X_k \sim$ Uniform{1,2}
- Next, we will see how we can figure out an optimal set of moves through Dynamic Programming.

Introductory Example

Value Function:

- Let V(x) be the maximum expected reward from the beginning of a round until the end of the game if we start the round with x toothpicks.

Define:

- V(0) = 0, also for completeness, V(-1) = 0
- V(1) = 20
- V(2) = 20

- We want to find V(10).

Introductory Example

- V(3) = Max Expected Reward if round starts with 3 toothpicks.
- If we pick 1 toothpick, computer will start with 2 toothpicks.
  - With probability ½, computer will pick 1 toothpick and hence we will start the next round with 1 toothpick.
  - With probability ½, computer will pick 2 toothpicks and hence we will start the next round with 0 toothpicks (game has ended).
  - Hence, if we pick 1 toothpick, our reward is:

    0.5 * V(1) + 0.5 * V(0) = 0.5 * 20 + 0.5 * 0 = 10.
- If we pick 2 toothpicks, computer will start with 1 toothpick.
  - With probability ½, computer will pick 1 toothpick and hence we will start the next round with 0 toothpick (game has ended).
  - With probability ½, computer will pick 2 toothpicks and hence we will start the next round with -1 toothpicks (game has ended).
  - Hence, if we pick 2 toothpicks, our reward is:

    0.5 * V(0) + 0.5 * V(-1) = 0.5 * 0 + 0.5 * 0 = 0.

Introductory Example

- V(3) = max (Reward if we pick 1, Reward if we pick 2}

  = max{10, 0}

  = 10

- V(4) = max {0.5 * V(2) + 0.5 * V(1) , 0.5 * V(1) + 0.5 * V(0) }

  = max {0.5 * 20 + 0.5 * 20 , 0.5 * 20 + 0.5 * 0 }

  = max {20, 10}

  = 20

- V(5) = max {0.5 * V(3) + 0.5 * V(2) , 0.5 * V(2) + 0.5 * V(1) }

  = max {0.5 * 10 + 0.5 * 20 , 0.5 * 20 + 0.5 * 20 }

  = max {15, 20}

  = 20

Introductory Example

- V(6) = max {0.5 * V(4) + 0.5 * V(3) , 0.5 * V(3) + 0.5 * V(2) }

  = max {0.5 * 20 + 0.5 * 10 , 0.5 * 10 + 0.5 * 20 }

  = max {15, 15}

  = 15
- V(7) = max {0.5 * V(5) + 0.5 * V(4) , 0.5 * V(4) + 0.5 * V(3) }

  = max {0.5 * 20 + 0.5 * 20 , 0.5 * 20 + 0.5 * 10 }

  = max {20, 15}

  = 20
- V(8) = max {0.5 * V(6) + 0.5 * V(5) , 0.5 * V(5) + 0.5 * V(4) }

  = max {0.5 * 15 + 0.5 * 20 , 0.5 * 20 + 0.5 * 20 }

  = max {17.5, 20}

  = 20

Introductory Example

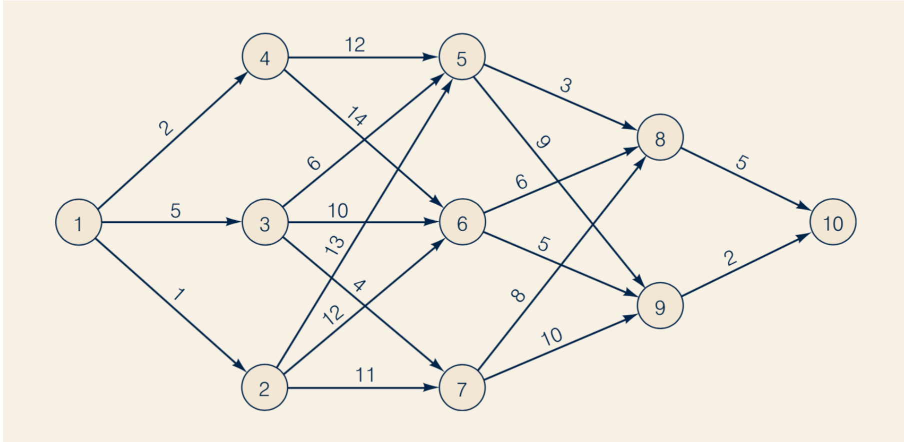
- V(9) = max {0.5 * V(7) + 0.5 * V(6) , 0.5 * V(6) + 0.5 * V(5) }
  = max {0.5 * 20 + 0.5 * 15 , 0.5 * 15 + 0.5 * 20 }
  = max {17.5, 17.5}
  = 17.5

- V(10) = max {0.5 * V(8) + 0.5 * V(7) , 0.5 * V(7) + 0.5 * V(6) }
  = max {0.5 * 20 + 0.5 * 20 , 0.5 * 20 + 0.5 * 15 }
  = max {20, 17.5}
  = 20

Optimal Policy:

- Move to nearest multiple of 3.
- If the initial number of toothpicks is not a multiple of 3, we always win!

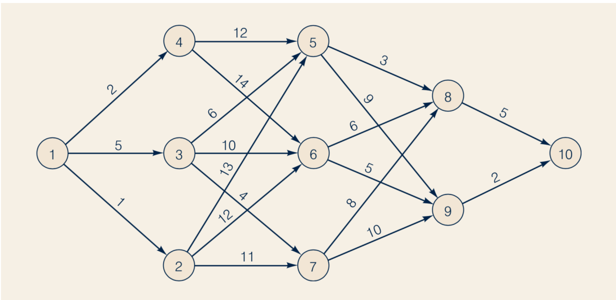
A Shortest Path Example*



- Finding the shortest path from Node 1 to Node 10

* Source: Anderson, Sweeney, Williams, Wisniewski, Pierron (2017)

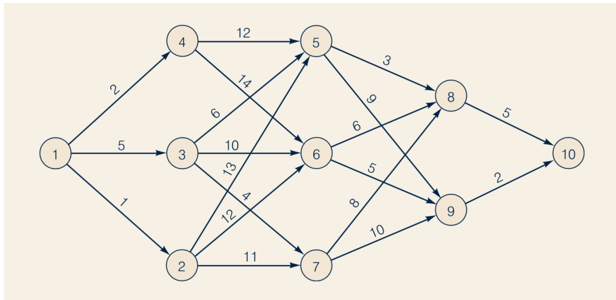
A Shortest Path Example



- Let V(x) be the minimum cost to go from Node x to Node 10.

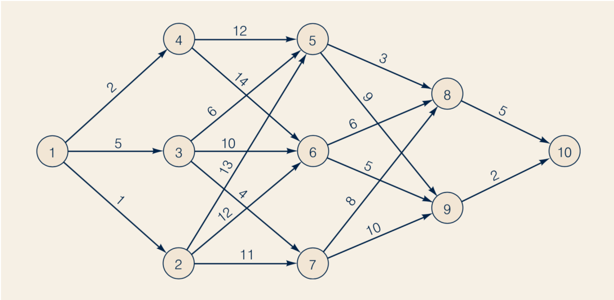
- V(9) = 2
- V(8) = 5

A Shortest Path Example



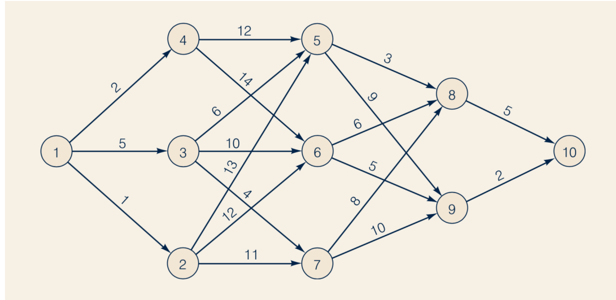
- V(7) = min { 8 + V(8), 10 + V(9) }
  = min { 8 + 5, 10 + 2 }
  = 12

A Shortest Path Example



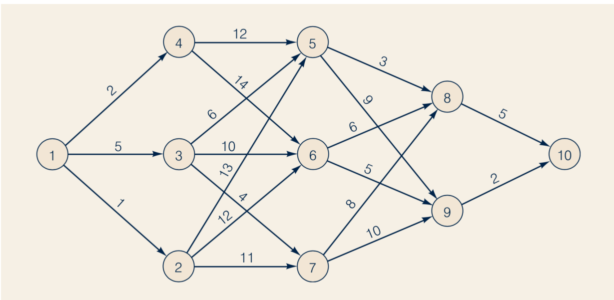
- V(6) = min { 6 + V(8), 5 + V(9) } = min { 6 + 5, 5 + 2 }
  = 7
- V(5) = min { 3 + V(8), 9 + V(9) } min { 3 + 5, 9 + 2 }
  = 8

A Shortest Path Example



- V(4) = min { 12 + V(5), 14 + V(6) }
  = min { 12 + 8, 14 + 7}
  = 20
- V(3) = min { 6 + V(5), 10 + V(6), 4 + V(7) }
  = min { 6 + 8, 10 + 7, 4 + 12 }
  = 14

A Shortest Path Example



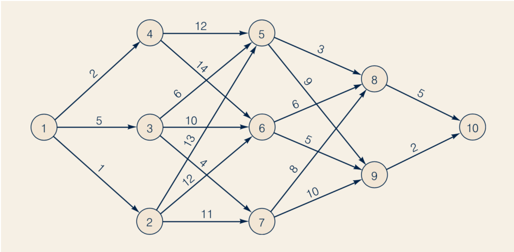
- V(2) = min { 13 + V(5), 12 + V(6), 11 + V(7) }
  = min { 13 + 8, 12 + 7, 11 + 12 } = 19
- V(1) = min { 1 + V(2), 5 + V(3), 2 + V(4) }
  = min { 1 + 19, 5 + 14, 2 + 20 } = 19
- The optimal path is 1 → 3 → 5 → 8 → 10 with a cost of 19.

A Shortest Path Example

- What if we wanted to evaluate all paths exhaustively?
- Here, we have 16 different possible paths:
  - 1,4,5,8,10: cost is 22
  - 1,4,5,9,10: cost is 23

    …
  - 1,2,7,9,10: cost is 24
- Advantage of exhaustive search?
  - It will give us the optimal solution, i.e., the shortest path.
- Drawback of exhaustive search?
  - Computationally very expensive, if at all feasible. Number of paths grow very quickly as number of stages and states per stage grow.
  - For example, if a problem has 10 intermediate stages and 10 states per stage, then the number of paths is $10*10*\ldots*10 = 10^{10}$ (not an unrealistic case, in fact many problems are much larger).

A Shortest Path Example



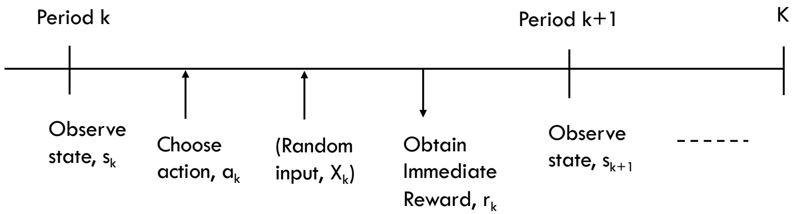
- What if we wanted to use a greedy path?
  - From 1, go to the shortest cost node, which is 2, etc.
  - 1,2,7,8,10: cost is 25
- Advantage of the greedy algorithm?
  - Faster computation.
- Drawback of the greedy algorithm?
  - Not necessarily optimal.

Introduction to Dynamic Programming

- Dynamic Programming:
  - is much faster than exhaustive search and
  - gives us the optimal solution.
- If a particular node is on the optimal route, then the shortest path from that node to the end is also on the optimal route.
- This is called the principle of optimality.

Introduction to Dynamic Programming

(Stochastic) Dynamic Programming:



- Horizon (K): K discrete decision periods (stages)
- State (s_k): The position we start a period
- Action (a_k): Allowed set of actions in each period (may depend on s_k)
- (Random Disturbance (X_k): Impacts state transition and rewards.)
- Reward (r_k): Immediate reward (may depend on s_k, a_k, X_k)
- Value Function ($V_k(s_k)$): Maximum possible total expected reward over the remaining horizon (depends on state s_k). Terminal reward $V_K(s)$ given.

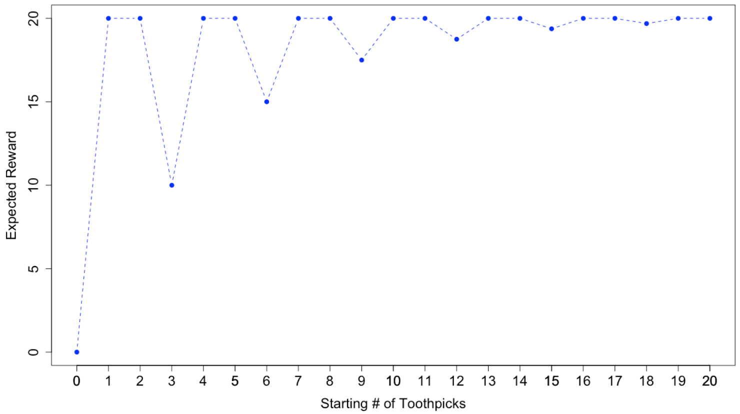
Introduction to Dynamic Programming

Solve by Backward Induction:

- *Initialization:* Terminal values $V_K(s)$ given for all s.
- *DP recursion:* $V_k(s) = \max E\left[r(s,a,X) + V_{k+1}(s_{k+1}(s,a,X))\right]$
  - This is the `Bellman Equation'
- If more appropriate, stage index can also be defined in reverse:
  - *Initialization:* Terminal values $V_0(s)$ given for all s.
  - *DP recursion:* $V_k(s) = \max E\left[r(s,a,X) + V_{k-1}(s_{k-1}(s,a,X))\right]$
- If future values are discounted with discount rate $\beta<1$,
  - *DP recursion:* $V_k(s) = \max E\left[r(s,a,X) + \beta V_{k-1}(s_{k-1}(s,a,X))\right]$

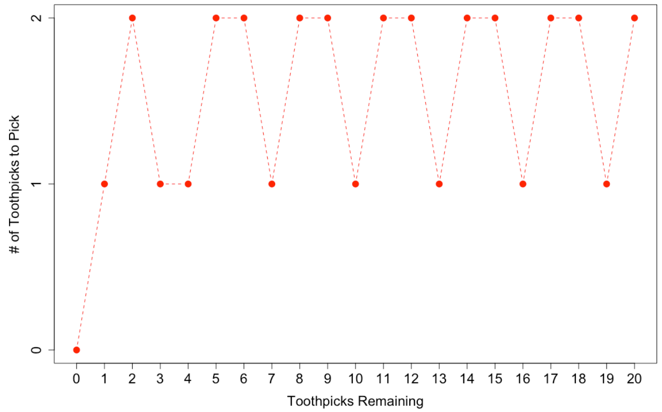
Implementing Dynamic Programming in R

- Please see the accompanying R Supplement.
- Visualizing Optimal Expected Reward for a 20 toothpick game:



Implementing Dynamic Programming in R

- Please see the accompanying R Supplement.
- Visualizing the Optimal Policy for a 20 toothpick game:



Single Resource Multi-Fare Capacity Allocation

- Capacity c
- allocated among n fare classes.
- Fares are indexed based on prices: $p_1 > p_2 > p_3 > \cdots > p_n$
- Low before high fare arrival order, e.g., fare class 1 arrives last.
  - Cheaper fares generally have time of purchase restrictions.
- D_j: random demand for fare class j, $j = \{1,2,\dots,n\}$
  - Demands are independent of each other.

Single Resource Multi-Fare Capacity Allocation

DP Formulation:

Let $V_j(x)$: optimal expected revenue from fare class $j, j-1, \dots, 1$.

Sequence of Events:

- Observe x, remaining capacity just before arrival of fare class j.
- Select protection level $y \leq x$ for remaining stages, i.e., for $j-1, j-2, \dots 1$.
  - In other words, make $x - y$ available for the current fare class j.
- Observe demand D_j for fare class j
  - Obtain revenue: $p_j \min(D_j, x - y)$
- Remaining capacity before facing fare class $j-1$:
  - Remaining capacity: $x - \min(D_j, x - y)$
  - (i.e., beginning capacity at stage j – sold in stage j)

Single Resource Multi-Fare Capacity Allocation

DP Formulation:

- *Initialization:*
  Terminal values $V_0(x) = 0$. Unsold seats are worthless.

- *DP recursion:* for $j \geq 1$

$$V_j(x) = \max_{y \leq x} \underbrace{p_j\, E\big[\min\big(D_j, x - y\big)\big]}_{\text{Immediate expected reward in current stage}} + \underbrace{E\big[V_{j-1}\big(x - \min\big(D_j, x - y\big)\big)\big]}_{\text{Optimal Expected Reward from Remaining Stages}}$$

Example*: Single Resource Multi-Fare Capacity Allocation

- Suppose that there are 5 fare classes.
  - $(p_1, p_2, p_3, p_4, p_5) = (100, 60, 40, 35, 15)$
- Demand for all fare classes is a Poisson random variable
  - Corresponding Expected Demands $= (15, 40, 50, 55, 120)$
- Initial capacity = 200

* Gallego and Topaloglu

Ex: Single Resource Multi-Fare Capacity Allocation

DP Formulation:

- *Initialization:*
  Terminal values $V_0(x) = 0$. Unsold seats are worthless.
- *DP recursion:*
  - *At stage 1, one period remaining:*

$$V_1(x) = \max_{y \leq x} p_1\, E[\min(D_1, x - y)] + \overbrace{E\big[V_0\big(x - \min\big(D_j, x - y\big)\big)\big]}^{\text{These } V_0(\cdot) \text{ values are all zero}}$$

We computed all these values during the previous recursion

  - *At stage 2, two periods remaining:*

$$V_2(x) = \max_{y \leq x} p_2\, E[\min(D_2, x - y)] + E[V_1(x - \min(D_2, x - y))]$$

  - *At stage 3, three periods remaining:*

$$V_3(x) = \max_{y \leq x} p_3\, E[\min(D_3, x - y)] + E[V_2(x - \min(D_3, x - y))]$$

  - *At stage 4, four periods remaining:*

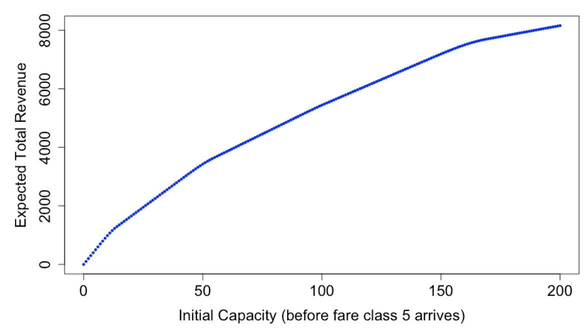
$$V_4(x) = \max_{y \leq x} p_4\, E[\min(D_4, x - y)] + E[V_3(x - \min(D_4, x - y))]$$

  - *At stage 5, five periods remaining:*

$$V_5(x) = \max_{y \leq x} p_5\, E[\min(D_5, x - y)] + E[V_4(x - \min(D_5, x - y))]$$

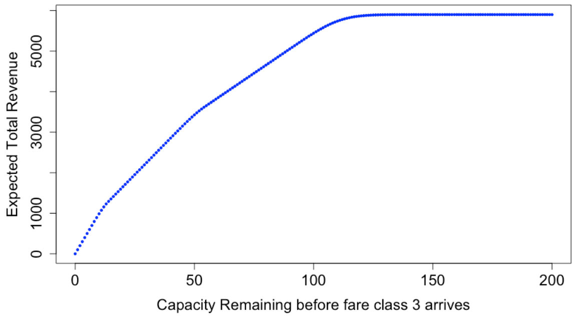
Example: Single Resource Multi-Fare Capacity Allocation

- For the code, please see the R Supplement.
- Visualizing Total Expected Revenue (5 stages to go):



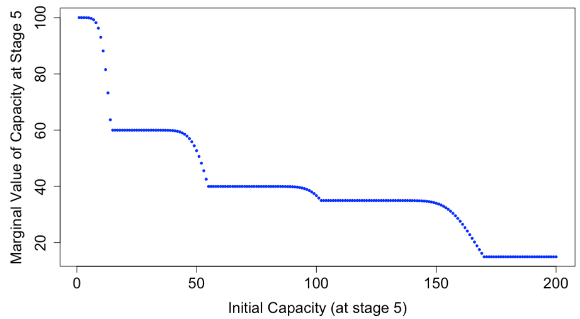
Example: Single Resource Multi-Fare Capacity Allocation

- Visualizing Total Expected Revenue (3 stages to go):



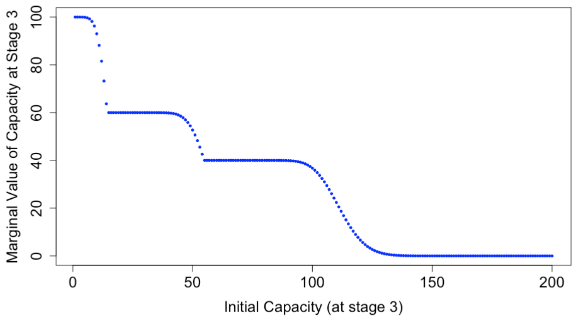
Example: Single Resource Multi-Fare Capacity Allocation

- Visualizing Marginal Value of Capacity (5 stages to go):



Example: Single Resource Multi-Fare Capacity Allocation

- Visualizing Marginal Value of Capacity (3 stages to go):



Heuristics for Multi-Fare Capacity Allocation

- A Heuristic Policy: An easy to compute decision rule that
  - is usually inspired by some structural properties of the optimal policy,
  - is computationally very efficient and
  - can achieve close to optimal reward.

A heuristic for the multi fare capacity allocation problem:

- At each stage j:
  - Compute total remaining demand from fares $j-1, \ldots, 1$ For example, at stage 3, the total remaining demand is the sum of two Poisson random variables with means 15 and 40, which itself is Poisson with mean 15+40=55.
  - Compute an effective mean demand weighted price for the total remaining demand. For example, at stage 3, the effective price for remaining demand is (100*15+60*40)/(15+40)=70.91.
  - Compute the Critical Fractile and the protection level through Littlewood's rule as in the two fare problem.

Modeling Time Dimension Explicitly: Mixed Arrivals

- Consider an entertainment event (e.g., a concert) with c=100 seats
- There are two fare classes: p_1=200, p_2=100
- Assume that the selling horizon is divided into small time segments with a total length of T=200 periods.
- At most one customer arrives at each time segment
  - With probability 0.3, a high fare demand arrives
  - With probability 0.6, a low-fare demand arrives
  - With probability 0.1, no customer arrives

Modeling Time Dimension Explicitly: Mixed Arrivals

- We will use Dynamic Programming to find the optimal capacity allocation across the time horizon.
- *Initialization:* $\mathrm{V}(x, 0)$ =0 for all x.
- *DP Recursions:*

$$\mathrm{V}(x,t) = \lambda_0 * \mathrm{V}(x,t-1) + \lambda_1 * \max\{p_1 + \mathrm{V}(x-1,t-1), \mathrm{V}(x,t-1)\} + \lambda_2 * \max\{p_2 + \mathrm{V}(x-1,t-1), \mathrm{V}(x,t-1)\}$$

- Please see R Supplement for a numerical implementation

Modeling Time Dimension Explicitly: Mixed Arrivals