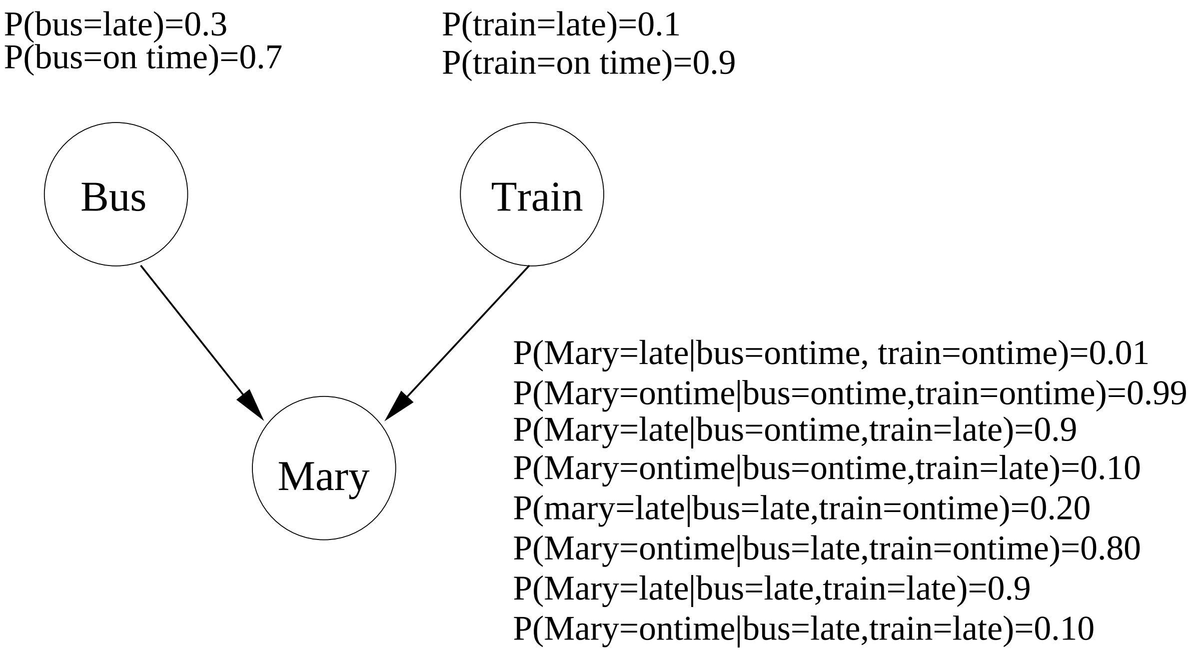
2AIT608 - Solutions to Tutorial 5 Exercises

Calculations in Bayesian Networks

1. The Bayesian network is shown in Figure 1.



2. From Bayes rule:

$$P(bus = late|mary = late) = \frac{P(mary = late|bus = late) \cdot P(bus = late)}{P(mary = late)} \tag{1}$$

Applying the theorem of total probability in the following two equations:

$$\begin{aligned} P(mary = late|bus = late) &= P(mary = late|bus = late, train = late) \cdot P(train = late) + \\ &\quad P(mary = late|train = ontime, bus = late) \cdot P(train = ontime) \\ &= 0.9 \cdot 0.1 + 0.2 \cdot 0.9 = 0.27 \end{aligned} \tag{2}$$

$$\begin{aligned} P(mary = late) &= P(mary = late|bus = late, train = late) \cdot P(bus = late, train = late) + \\ &\quad P(mary = late|bus = ontime, train = late) \cdot P(bus = ontime, train = late) + \end{aligned}$$

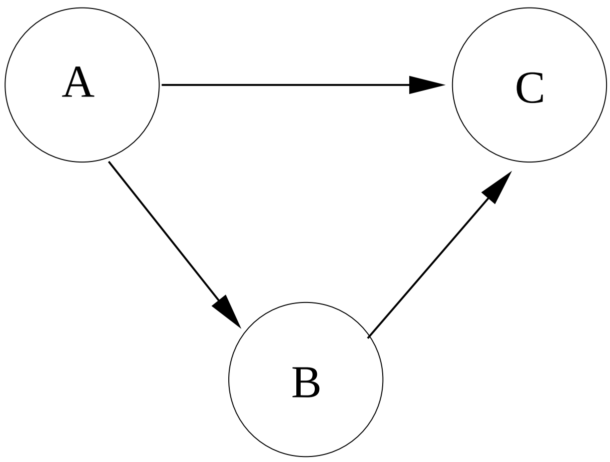
$$
\begin{aligned}
& P(mary = late|bus = late, train = ontime) \cdot P(bus = late, train = ontime) + \\
& P(mary = late|bus = ontime, train = ontime) \cdot P(bus = ontime, train = ontime) \\
= \; & P(mary = late|bus = late, train = late) \cdot P(bus = late) \cdot P(train = late) + \\
& P(mary = late|bus = ontime, train = late) \cdot P(bus = ontime) \cdot P(train = late) + \\
& P(mary = late|bus = late, train = ontime) \cdot P(bus = late) \cdot P(train = ontime) + \\
& P(mary = late|bus = ontime, train = ontime) \cdot P(bus = ontime) \cdot P(train = onti \\
= \; & 0.9 \cdot 0.3 \cdot 0.1 + 0.9 \cdot 0.7 \cdot 0.1 + \\
& 0.2 \cdot 0.3 \cdot 0.9 + 0.01 \cdot 0.7 \cdot 0.9 = 0.15
\end{aligned}
$$

Substituting (2), (3) in (1):

$$P(bus = late|mary = late) = \frac{0.27 \cdot 0.3}{0.15} = 0.54 \tag{4}$$

Bayesian Networks Used for Prediction in Simulations

1. The Bayesian network (without the calculated probabilities) is shown in Figure 1.



2. To calculate the probabilities in the table we need to keep track of the total number of the strikes thrown N. Then we can calculate probabilities like the following:

$$P(A = Punch) = N_{punch}/N \tag{5}$$

$$P(B = Punch|A = LowKick) = N_{LowKick-punch}/N \tag{6}$$

$$P(C = Punch|B = Punch, A = LowKick) = N_{LowKick-punch-punch}/N_{AB} \tag{7}$$

Then assuming that `move` is the most recent strike thrown by the player and `AB[2]` is a two dimensional array holding the two previous strikes, the program (in C - convert to the Matlab equivalent) is:

```
enum TStrikes {Punch, LowKick, HighKick};

ProcessMove(TStrikes move)
{
        int     i, j;

        N++;
        if(move == Prediction) NSuccess++;
        if(move == RandomPrediction) NRandomSuccess++;

        if((AB[0] == Punch) && (AB[1] == Punch)) i = 0;
        if((AB[0] == Punch) && (AB[1] == LowKick)) i = 1;
        if((AB[0] == Punch) && (AB[1] == HighKick)) i = 2;

        if((AB[0] == LowKick) && (AB[1] == Punch)) i = 3;
        if((AB[0] == LowKick) && (AB[1] == LowKick)) i = 4;
        if((AB[0] == LowKick) && (AB[1] == HighKick)) i = 5;

        if((AB[0] == HighKick) && (AB[1] == Punch)) i = 6;
        if((AB[0] == HighKick) && (AB[1] == LowKick)) i = 7;
        if((AB[0] == HighKick) && (AB[1] == HighKick)) i = 8;

        if(move == Punch) j = 0;
        if(move == LowKick) j = 1;
        if(move == HighKick) j = 2;

        NAB[i]++;
        NCAB[i][j]++;

        AB[0] = AB[1];
        AB[1] = move;


        if((AB[0] == Punch) && (AB[1] == Punch)) i = 0;
        if((AB[0] == Punch) && (AB[1] == LowKick)) i = 1;
        if((AB[0] == Punch) && (AB[1] == HighKick)) i = 2;

        if((AB[0] == LowKick) && (AB[1] == Punch)) i = 3;
        if((AB[0] == LowKick) && (AB[1] == LowKick)) i = 4;
        if((AB[0] == LowKick) && (AB[1] == HighKick)) i = 5;

        if((AB[0] == HighKick) && (AB[1] == Punch)) i = 6;
        if((AB[0] == HighKick) && (AB[1] == LowKick)) i = 7;
        if((AB[0] == HighKick) && (AB[1] == HighKick)) i = 8;

        ProbPunch = (double) NCAB[i][0] / (double) NAB[i];
        ProbLowKick = (double) NCAB[i][1] / (double) NAB[i];
        ProbHighKick = (double) NCAB[i][2] / (double) NAB[i];
```

```
        if((ProbPunch > ProbLowKick) && (ProbPunch > ProbHighKick)) return Punch;
        if((ProbLowKick > ProbPunch) && (ProbLowKick > ProbHighKick)) return LowKick;
        if((ProbHighKick > ProbPunch) && (ProbHighKick > ProbLowKick)) return HighKick;

        return (TStrikes) rand() % 3;

}
```

In the above program we do not choose randomly among two strikes if they have the same probability which is higher than the third, but the program could be modified accordingly to do this.