NBA Scheduling

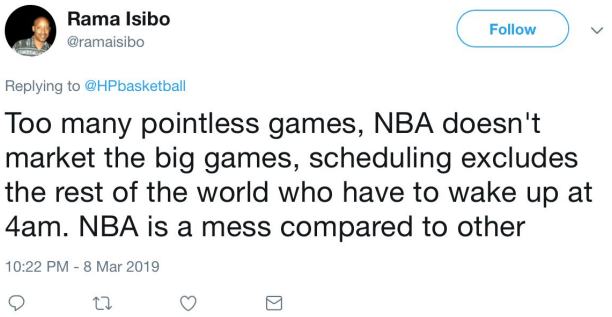
May 2, 2019

Mitch Gaiser, Alex Carvalho,
Luqman Ebrahim, Norman Paasivaara

https://freebiesupply.com/logos/nba-logo/

Introduction

- NBA schedules 1230 games in 176 days
- Revenues total $2.6 bn
- Inefficiencies create
  - Lost Revenue Opportunities
  - Player dissatisfaction
  - Fan dissatisfaction



Initial Ideas

- Create an LP with team fatigue and fan penalty
  - Maximize Revenue for NBA
  - Minimize Fatigue (back-to-back games)
  - Minimize travel distance
- 162,000 Variable matrix
  - Very hard to **optimize**
  - Massive computing power

The Model

Definitions

x_{ij}^{d} = 1 if team **i** plays team **j** on day **d**; 0 otherwise.

→ i = 1,...,30; j = 1,...,30; d = 1,...,180.

w_{ij}^{d} = objective coefficient value if team **i** plays team **j** on day **d.**

→ comes from team-viewership data and importance of day.

y_{i}^{w} represents the number of games team **i** plays in week **w**

Each team i is in Conference Set S_C ; where C = 1 or 2

Each team i is in Division Set S_D ; where D = 1, 2, 3, 4, 5, or 6

The Model

$$\max \quad \sum_{i=1}^{30}\sum_{j=1}^{30}\sum_{d=1}^{180} w_{ij}^{d}\, x_{ij}^{d} \;-\; \lambda \sum_{i=1}^{30}\sum_{j=1}^{30}\sum_{d=1}^{179} x_{ij}^{d}\, x_{ij}^{d+1}$$

subject to:

$\sum_{i=1}^{30}\sum_{j=1}^{30} x_{ij}^{d} \le 1 \;\forall\, d$ Teams can only play once per day

$\sum_{j=1}^{30}\sum_{d=1}^{180} x_{ij}^{d} = 82 \;\forall\, i$ Each team plays 82 total games

$\sum_{d=1}^{180} x_{ij}^{d} = 2 \;\forall\, i \in S_{C_i},\ j \in S_{C_j},\ C_i \neq C_j$ Teams in opposing conferences play twice

$\sum_{d=1}^{180} x_{ij}^{d} \le 4 \;\forall\, (i,j) \in S_{C_i}$ Teams in the same conference play at most 4 times

$\sum_{d=1}^{180} x_{ij}^{d} = 4 \;\forall\, (i,j) \in S_{D_i}$ Teams in the same division play exactly 4 times

$\sum_{i=1}^{30}\sum_{j=1}^{30}\sum_{d=1}^{180} x_{ij}^{d} = 1230$ Total number of scheduled games is 1,230

$\sum_{i=1}^{30}\sum_{j=1}^{30}\left(\sum_{d=123}^{128} x_{ij}^{d} + x_{ij}^{70} + x_{ij}^{38}\right) = 0$ No games during all-star break, Christmas Eve and Thanksgiving

$\sum_{i=1}^{30}\sum_{j=1}^{30} x_{ij}^{71} \le 5$ At most five games on Christmas Day

$\sum_{i=1}^{30}\sum_{j=1}^{30} x_{ij}^{1} \le 2$ At most 2 games on Opening Day

$y_i^w = \sum_{d=7w-8}^{7w-2}\sum_{j=1}^{30} x_{ij}^{d} \;\forall\, i,\ w = 1, \ldots, 26$ Weekly variable definition

$y_i^w \le 4 \;\forall\, i, w$ At most 4 games per week for all teams

$y_i^w \ge 1 \;\forall\, i, w,\ w \neq 19$ At least 1 games per week for all teams

The Greedy Algorithm


```
while (all teams haven't played 82 games) {
        for (weight-sorted days in the schedule) {
                choose the objective maximizing game on that day
                if (the game is schedulable) {
                        schedule it
                        decrease objective coefficients for d-1 and d+1
                } else {
                        set objective coefficient for team-team-day combination = 0
                }
        }
}
```

```
# NBA scheduling project

# total length of schedule
days = rep(0, 180)

# create a function to easily update day weights by day of the
day_coefficient_by_week = function(day_of_week, weight, list) {
  # day of week is a number
  # weight is a number
  # days is a vector of weights

  # for input:
  #Sunday = 1
  #Monday = 2 ... Saturday = 7

  # input is changed here for correct use in code
  if (day_of_week == 1 || day_of_week == 2) {
    day_of_week = day_of_week + 5
  } else {
    day_of_week = day_of_week - 2
  }
  for (i in seq(from = day_of_week, to = length(days), by
    list[i] = weight
  }
  return(list)
}

team_weights = rep(0, 30)
# fill in team weights

# create our objective function coefficients easily with d
objective_coefficients = function(team_weights, day_weight
  # team weights and day weights are both vectors
  num_teams = length(team_weights)
  num_days = length(day_weights)
  # create a 3D matrix of zeros to be changed
  zero_s = rep(0, (num_teams^2)*num_days)
  obj = array(zero_s, c(num_teams, num_teams, num_days))

  for (d in 1:num_days) {
    for (i in 1:num_teams) {
      for (j in 1:num_teams) {
        if (i == j) { # do nothing
        } else {
          obj[i,j,d] = team_weights[i]*team_weights[j]*day_
        }
      }
    }
  }
  return(obj)
}

# Index, Team, #Wins in 17-18 season, TV viewership statis
# EAST:
# Atlantic
# 1 - Raptors, 59, 1.44
# 2 - 76ers, 52, 1.69
# 3 - Celtics, 55, 2.33
# 4 - Nets, 28, *1.66 + 1.82 / 2 = 1.74
# 5 - Knicks, 29, 1.82
# 29 - Pelicans, 48, 1.42
# 30 - Mavericks, 24, 1.14

# Hawks, Magic and Nets had no TV data, so we took average

# from code to English:
Team_list = c("Raptors", "76ers", "Celtics", "Nets", "Knic
              "Bucks", "Pacers", "Pistons", "Bulls", "Cava
              "Magic", "Hornets", "Heat", "Wizards", "Hawk
              "Nuggets", "Trail Blazers", "Jazz", "Thunder
              "Warriors", "Clippers", "Kings", "Lakers", "
              "Rockets", "Spurs", "Grizzlies", "Pelicans",

# thoughts - are one of these more important? different we
team_weights1 = c(59*1.44, 52*1.69, 55*2.33, 28*1.74, 29*1
                  44*1.39, 48*1.53, 39*1.33, 27*1.48, 0.5*
                  25*1.31, 36*1.71, 44*1.30, 43*1.79, 24*1
                  46*1.15, 49*1.30, 48*1.06, 48*2.20, 47*1
                  58*2.42, 42*1.25, 27*1.66, 1.5*35*1.80,
                  65*2.07, 47*1.75, 22*1.17, 48*1.42, 24*1
# team_weights1
# [1]  84.96  87.88 128.15  48.72  52.78  61.16  73.44  5
# [16]  52.98  63.70  50.88 105.60  72.85 157.76  52.50  4

# Day weight creation:
#Saturdays are most weighted (Day 7):
days = day_coefficient_by_week(7,8,days)
# then Fridays (day 6):
days = day_coefficient_by_week(6,7,days)
# then Sunday (day 1):
days = day_coefficient_by_week(1,6,days)
# then Wednesdays (day 4)
days = day_coefficient_by_week(4,5,days)
# and Mondays and Thursdays are tied:
days = day_coefficient_by_week(2,4,days)
days = day_coefficient_by_week(5,4,days)
# tuesdays are lowest (day 3)
days = day_coefficient_by_week(3,3,days)
# most important day is christmas which is day 71 on the s
days[71] = 10
# second most important day is opening day
days[1] = 9
# the all star break will be added in the choosing in the a

# now that we have team weights, and day weights, we can co

obj1 = objective_coefficients(team_weights1, days)



# Greedy Algorithm Construction:


# where to go: loop through this list over and over until a

# first iteration: randomize days to spread out
set.seed(10)
day_set = 1:180
random_days = sample(day_set, replace = F)

# obj1 = objective value matrix
L = length(obj1[1,,1])

# create an empty matrix that will be edited to
schedule = rep(0, 30*30)
number_games = array(data, c(30,30))
# number of games each team plays against divis
# number of games against other teams in confer
# number of games against other teams out of co
# number of total games must be 82
obj_copy = obj1

# add names for conferences and divisions:
East = seq(1:15)
West = seq(from = 16, to = 30)

Atlantic = seq(1:5)
Central = seq(from = 6, to = 10)
Southeast = seq(from =11, to = 15)
Northwest = seq(from = 16, to = 20)
Pacific = seq(from = 21, to = 25)
Southwest = seq(from = 26, to = 30)

obj_value = 0
# create an empty 3D matrix where binaries will
Binary = rep(0, 180*30*30)
Binary_Schedule = array(data, c(30,30,180))

sorted_random_days = rep(0,180)

which(days == 10)
# [1] 71
sorted_random_days[1] = 71
which(days == 9)
# [1] 1
sorted_random_days[2] = 1
which(days == 8)
# [1] 5  12  19  26  33  40  47  54  61  68  75
w_ = sample(w, replace = F)
# > w_
# [1] 166 159 180  75 110  61 152 117  26 145
for (i in 3:(length(w) + 2)) {
  sorted_random_days[i] = w_[i-2]
}

w = which(days == 7)
w_ = sample(w,replace = F)
for (i in 29:(length(w) + 28)) {
  sorted_random_days[i] = w_[i-28]
}

w = which(days == 6)
w_ = sample(w,replace = F)
for (i in 55:(length(w) + 54)) {
  sorted_random_days[i] = w_[i-54]
}

w = which(days == 5)
w_ = sample(w,replace = F)
for (i in 80:(length(w) + 79)) {
  sorted_random_days[i] = w_[i-79]
}

w = which(days == 4)
w_ = sample(w,replace = F)
for (i in 106:(length(w) + 105)) {
  sorted_random_days[i] = w_[i-105]
}

w = which(days == 3)
w_ = sample(w,replace = F)
for (i in 157:(length(w) + 156)) {
  sorted_random_days[i] = w_[i-156]
}

#sorted_random_days

# add christmas day to maximize that
sorted_random_days = sorted_random_days[-c(1,2)

# day 122 - 129 is the all-star break

# make the games per week matrix:
week_count = 1
week_set = c()
for (i in seq(from = -1, to = 180, by = 7)) {
  print(paste0(week_count, " ", i, " ", i+1, " ",
  week_set = rbind(week_set, c(i,i+1, i+2, i+3,
  week_count = week_count + 1
}
# week set tells us which day belongs to which

weekly_games = array(rep(0,26*30),c(30,26))
# weekly_games[i,w] represents the number of ga

# back to back reducer
reducer = 0.55

total_games = rep(0,30)
total_conference_games = rep(0,30)
LOOP = 1
X = 0
# algorithm produces a schedule where each team plays 76 games, the final 6 games will b
# greedy algorithm

while(sum(total_games) != 30*76) {


  # consider adding the Christmas game constraint
  for (d in sorted_random_days) {
    w = which(week_set == d, arr.ind = T)[1]
    if ((d > 122 && d < 129) ||
        (d == 71 && LOOP > 1) ||
        (d == 1 && LOOP > 1) ||
        d == 70 || d == 38) {
      # don't schedule anything (All-star break, christmas eve, thanksgiving)
    } else {
      scheduled = 0
      # sort the weights in decreasing order
      sorted_weights = sort(obj_copy[,,d], decreasing = T)
      # determine if the game is schedulable:
      while (scheduled == 0) {
        #East_West_Game = 0
        TEAMS = which(obj_copy[,,d] == max(obj_copy[,,d]), arr.ind = T)
        if (dim(TEAMS)[1] > 1) {
          # all zeros
          scheduled = 1
        } #print(paste0(TEAMS, " TEAM SEARCH"))
        else if (total_games[TEAMS[1]] >= 82 || total_games[TEAMS[2]] >= 82 ||
                 sum(Binary_Schedule[TEAMS[1],,d], Binary_Schedule[,TEAMS[1],d]) >= 1 ||
                 sum(Binary_Schedule[TEAMS[2],,d], Binary_Schedule[,TEAMS[2],d]) >= 1 ||
                 weekly_games[TEAMS[1],w] >= 3 || weekly_games[TEAMS[2],w] >= 4) {
          # do not schedule games if team has a full schedule
          # do not schedule game if team already plays
          obj_copy[TEAMS[1],TEAMS[2],d] = 0
        }


        # if teams are in opposing conferences and they havent played 2 or more games, s
        else if ((TEAMS[1] %in% East && TEAMS[2] %in% West) || (TEAMS[2] %in% East && TE
          if (number_games[TEAMS[1], TEAMS[2]] < 2) {
            # schedule the game
            number_games[TEAMS[1], TEAMS[2]] = number_games[TEAMS[1], TEAMS[2]] + 1
            scheduled = 1
            obj_value = obj_value + X
            Binary_Schedule[TEAMS[1], TEAMS[2], d] = 1
            #print(paste0(TEAMS[1], " plays ", TEAMS[2], " E-W"))
            print(paste0("Day ", d, ": ", Team_list[TEAMS[1]], " vs. ", Team_list[TEAMS[
            total_games[TEAMS[1]] = total_games[TEAMS[1]] + 1
            total_games[TEAMS[2]] = total_games[TEAMS[2]] + 1
            weekly_games[TEAMS[1],w] = weekly_games[TEAMS[1],w] + 1
            weekly_games[TEAMS[2],w] = weekly_games[TEAMS[2],w] + 1
            # when a game is scheduled, lets reduce the objective of "nextdoor" games

            if (d == 180) {
              #dont increase to d == 181
            } else {
              obj_copy[TEAMS[1],,(d+1)] = reducer*obj_copy[TEAMS[1],,(d+1)]
              obj_copy[,TEAMS[1],(d+1)] = reducer*obj_copy[,TEAMS[1],(d+1)]
            }
            if (d == 1) {
              # dont reduce to d == 0
            } else {
              obj_copy[TEAMS[1],,(d-1)] = reducer*obj_copy[TEAMS[1],,(d-1)]
              obj_copy[,TEAMS[1],(d-1)] = reducer*obj_copy[,TEAMS[1],(d-1)]
            }

          } else {
            #East_West_Game = 1
            #max_finder = max_finder + 1
            obj_copy[TEAMS[1],TEAMS[2],d] = 0
            #print(paste0(max_finder, " FIRST"))
          }
        } # if the teams are in the same division and haven't played 4 or more games, schedule it
        else if ((((TEAMS[1] %in% Atlantic && TEAMS[2] %in% Atlantic) || (TEAMS[1] %in% Central && TEAMS
                   (TEAMS[1] %in% Southeast && TEAMS[2] %in% Southeast) || (TEAMS[1] %in% Pacific && TEA
                   (TEAMS[1] %in% Northwest && TEAMS[2] %in% Northwest) || (TEAMS[1] %in% Southwest && TEA
                 number_games[TEAMS[1], TEAMS[2]] < 4 )) {
          # && East_West_Game != 1 add above
          # schedule the game
          number_games[TEAMS[1], TEAMS[2]] = number_games[TEAMS[1], TEAMS[2]] + 1
          scheduled = 1
          obj_value = obj_value + X
          Binary_Schedule[TEAMS[1], TEAMS[2], d] = 1
          #print(paste0(TEAMS[1], " plays ", TEAMS[2], " DIVISION"))
          print(paste0("Day ", d, ": ", Team_list[TEAMS[1]], " vs. ", Team_list[TEAMS[2]],
          total_games[TEAMS[1]] = total_games[TEAMS[1]] + 1
          total_games[TEAMS[2]] = total_games[TEAMS[2]] + 1
          weekly_games[TEAMS[1],w] = weekly_games[TEAMS[1],w] + 1
          weekly_games[TEAMS[2],w] = weekly_games[TEAMS[2],w] + 1
          # when a game is scheduled, lets reduce the objective of "nextdoor" games

          if (d == 180) {
            #dont increase to d == 181
          } else {
            obj_copy[TEAMS[1],,(d+1)] = reducer*obj_copy[TEAMS[1],,(d+1)]
            obj_copy[,TEAMS[1],(d+1)] = reducer*obj_copy[,TEAMS[1],(d+1)]
          }
          if (d == 1) {
            #dont reduce to d == 0
          } else {
            obj_copy[TEAMS[1],,(d-1)] = reducer*obj_copy[TEAMS[1],,(d-1)]
            obj_copy[,TEAMS[1],(d-1)] = reducer*obj_copy[,TEAMS[1],(d-1)]
          }
        }
      }
    }
    #print(LOOP)
    if (d == 71 && LOOP == 1) {
      obj_copy[TEAMS[1],TEAMS[2],1] = 0
    }
  }
  LOOP = LOOP + 1
}
# choosing final 6 games for each team

obj_copy2 = obj1

# find schedulable games as possibilities
possibilities = array(rep(0,30*30*180), c(30,30,180))
for (i in 1:30) {
  for (j in 1:30) {
    if (i == j) {
      # do nothing
    }
    else if (((((i %in% East && j %in% East) || (i %in% West && j %in% West)) &&
        ((((i %in% Atlantic) && j %in% Atlantic) || ((i %in% Atlantic) && i %in% Atlantic)) ||
        ((((i %in% Central) && j %in% Central) || ((i %in% Central) && i %in% Central)) ||
        ((((i %in% Southeast) && j %in% Southeast) || ((i %in% Southeast) && i %in% Southeast)) ||
        ((((i %in% Pacific) && j %in% Pacific) || ((i %in% Pacific) && i %in% Pacific)) ||
        ((((i %in% Northwest) && j %in% Northwest) || ((i %in% Northwest) && i %in% Northwest)) ||
        ((((i %in% Southwest) && j %in% Southwest) || ((i %in% Southwest) && i %in% Southwest)))
      ) {
      # if the teams are in the same conference but different divisions, then its a match
    # if the teams are in the same conference but different divisions, then its a match
      for (d in 1:length(Binary_Schedule[1,1,])) {
        if (which(week_set == d, arr.ind = T)[1]
          if (sum(Binary_Schedule[i,,d], Binary_Schedule[,i,d]) >= 1 ||
          sum(Binary_Schedule[j,,d], Binary_Schedule[,j,d]) >= 1 ||
          weekly_games[i,w] >= 4 || weekly_games[j,w] >= 4) {
            # cannot schedule game if either team is already playing that day
          } else {
            possibilities[i,j,d] = obj_copy2[i,j,d]
            #print(paste0(i, " can play ", j, " on day ", d))
          }
        }
      }
    }
  }
}
possibilities_copy = possibilities

# new approach
team_weight_matrix = array(rep(0,30*30),c(30,30))
for (i in 1:30) {
  for (j in 1:30) {
    if (i == j) {
      # do nothing
    } else {
      team_weight_matrix[i,j] = team_weights1[i] * team_weights1[j]
    }
  }
}

# order the team weights:
ordered_team_weights = c()

sorted_team_weights = sort(team_weights1)
for (i in 1:30) {
  ordered_team_weights = c(ordered_team_weights, which(team_weights1 == sorted_team_weights[i]))
}

matched_number_games = rep(0, 30)
number_games3 = array(rep(0,30*30), c(30,30))
six_schedule_games = c()

while (sum(matched_number_games) < 30*6) {
  for (i in ordered_team_weights) {
    while (matched_number_games[i] < 6) {
      TEAMS = which(team_weight_matrix == max(team_weight_matrix[,i],team_weight_matrix[i,]), arr.ind = T)

      if(sum(possibilities[TEAMS[1],TEAMS[2],]) == 0 ||
         number_games3[TEAMS[1],TEAMS[2]] >= 1 ||
         matched_number_games[TEAMS[1]] >= 6 || matched_number_games[TEAMS[2]] >= 6) {
        # do nothing
      } else {
        # pair them up
        matched_number_games[TEAMS[1]] = matched_number_games[TEAMS[1]] + 1
        matched_number_games[TEAMS[2]] = matched_number_games[TEAMS[2]] + 1
        number_games3[TEAMS[1], TEAMS[2]] = number_games3[TEAMS[1], TEAMS[2]] + 1
        print(paste0(TEAMS[1], " should play ", TEAMS[2]))

        six_schedule_games = rbind(six_schedule_games, c(TEAMS[1], TEAMS[2]))
      }
      team_weight_matrix[TEAMS[1],TEAMS[2]] = 0

    }
  }
}

# now we have matchings of which teams will play, and we just have to schedule them
# lets find the number of days each team could play:
game_count = c()
for (i in 1:length(six_schedule_games[,1])) {
  team1 = six_schedule_games[i,1]
  team2 = six_schedule_games[i,2]
  count = 0
  for (d in 1:180) {
    if (possibilities[team1,team2,d] == 0) {
      # do nothing
    } else {
      count = count + 1
    }
  }
  game_count = c(game_count, count)
}

# theres so many possible days, this really should not be an issue

# we choose to remove christmas day and opening day for these remaining games
#six_random_days = sorted_random_days[-c(1:6)]
possibilities[,,71] = 0
possibilities[,,1] = 0

for (i in 1:length(six_schedule_games[,1])) {
  scheduled = 0
  team1 = six_schedule_games[i,1]
  team2 = six_schedule_games[i,2]
  # max_days = which(possibilities[team1,team2,] == max(possibilities[team1,team2,]), arr.ind = T)
  # choice_list = sample(1:TOP,TOP)

  while(scheduled == 0) {

    max_days = which(possibilities[team1,team2,] == max(possibilities[team1,team2,]), arr.ind = T)
    TOP = length(max_days)
    choice_list = sample(1:TOP,TOP)

    for (choice in choice_list) {
      d = max_days[choice]
      w = which(week_set == d, arr.ind = T)[1]
      if (weekly_games[team1,w] >= 4 || weekly_games[team2,w] >= 4) {
        # do nothing
        possibilities[team1,team2,d] = 0
      } else {
        scheduled = 1
      }
    }
  }
}

# schedule the game
Binary_Schedule[team1,team2,d] = 1
print(paste0(team1, " play ", team2, " on day ", d))
weekly_games[team1,w] = weekly_games[team1,w] + 1
weekly_games[team2,w] = weekly_games[team2,w] + 1

# remove possibilities for that team on that day in the future
possibilities[team1,,d] = 0
possibilities[,team1,d] = 0
possibilities[team2,,d] = 0
possibilities[,team2,d] = 0

# how will game not get scheduled


}

for (i in 1:30) {
  if (sum(Binary_Schedule[,i,],Binary_Schedule[i,,]) != 82) {
    print(paste0("oh no ", Team_list[i]))
  }
}

# we're all good -- lets write up the schedule
NBA_SCHEDULE = data.frame(matrix(NA,nrow = 3,ncol=1230))
colnames(NBA_SCHEDULE) = c("Day", "Team1", "Team2")

row_count = 1
for (d in 1:180) {
  for (i in 1:30) {
    for (j in 1:30) {
      if (i == j) {
        # else do nothing
      } else {
        NBA_SCHEDULE[row_count,1] = d
        NBA_SCHEDULE[row_count,2] = Team_list[i]
        NBA_SCHEDULE[row_count,3] = Team_list[j]
        print(paste0("Day ", d, ": ", Team_list[i], " vs. ", Team_list[j]))
        row_count = row_count + 1
      }
    }
  }
}
```

Interesting Finds:

Opening Day

Actual 2018-19 NBA Schedule

vs.

vs.

Our 2018-19 NBA Schedule

vs.

vs.

Interesting Finds:

Christmas Day

Actual 2018-19 NBA Schedule

vs.


vs.


vs.


Our 2018-19 NBA Schedule

vs.


vs.


vs.


Interesting Finds:

25

Exactly matched games

Evaluation

First Iteration Schedule

- 7% increased TV revenue over NBA
- $180M “improvement”
- However, on average 35 back-to-back games
- Slightly less total travel distance as real schedule

New Schedule (B2B penalty)

- 5% increased TV revenue over NBA
- $130M “improvement”
- Reduction to average of 25 back-to-back games
- Slightly less total travel distance as real schedule

Conclusion

- Tradeoff between revenue and player comfortability
- Difficulty balancing different objectives
- NBA likely not optimizing for revenue
- Optimizing for additional constraints such as rest, travel time or cost

Further exploration

Additional Inputs

- Better Revenue Model
- Broadcasting availability and constraints
- Home and Away optimization

Methodology

- Different or added objective (travel time, rest etc.)
- Improved Heuristic
- Industrial solver with IP

Q&A