Results

If winners are deterministic, things work out as Dodgson claimed

Of course, Dodgson was a highly accomplished mathematician so no surprise that he is correct that his method correctly allocates prizes in a 32 player competition with deterministic match outcomes. I *was* curious to see if it would work in a larger competition. I found that his rules (with minimal modifications discussed later in this post) returned the top three players with the top three prizes, in correct order, 100 times out of 100 different runs.

A correctly seeded single-elimination tournament with deterministic match outcomes will also return the top 4 players accurately 100% of the time. I think that Dodgson assumed there was no reliable prior knowledge of players' skill levels, ruling out the possibility of a seeded tournament; certainly he only compares his method to an unseeded draw.

Note that a Dodgson tournament will take around twice as many matches (varying according to the efficiency of the draw, but around 240 matches was normal in my experiment) as a single-elimination tournament (which needs 127 matches for 128 players).

Results are not so good when match wins are realistically probabilistic

Dodgson's method is not as good as he would hope when match outcomes are not deterministic, but happen by chance in accordance with the skill differential indicated by players different Elo ratings. Of course, this is the realistic model; even at the height of her powers, Steffi Graf (the top rated player in our 1990 collection) had non zero chance of losing to her top opponents in any given match, as we saw in the graphic from last week:

The figure below shows the result of 1,000 simulated tournaments played out according to Dodgson's rules, with realistic win and loss probabilities (ie not just 1 and 0). Some example conclusions from this chart include:

- The top player wins the tournament 57% of the time
- The top two players are the top two prize recipients 36% of the time, and in the right 1-2 sequence 23% of the time.
- The top three players are awarded the top three prizes in the right sequence only 7% of the time

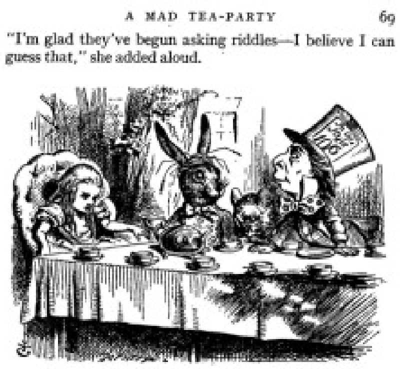
For comparison, all those figures are 100% in the deterministic model Dodgson planned on.

Under the probabilistic model, the outcomes of Dodgson's tournament rules are similar to those from a seeded single elimination tournament as in my blog post last week. For example, in that seeded competition, top player Steffi Graf won 60% of tournaments; and the top two players were in the grand final 42% of the time. This is marginally better than the result under Dodgson's rules with around half the number of matches, reflecting the effective use that the extra prior information on player ability is put to in the seeding.

Implementation

Approach

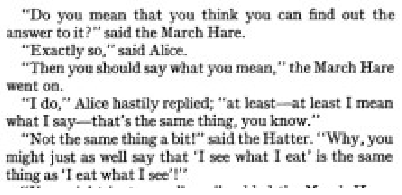
It was interesting and non-trivial to implement Dodgson's rules in a way that would be robust to larger tournament sizes, random match outcomes, and inconsistent and non-transitive results. A few notable choices I had to make:


A MAD TEA-PARTY 69

"I'm glad they've begun asking riddles—I believe I can guess that," she added aloud.

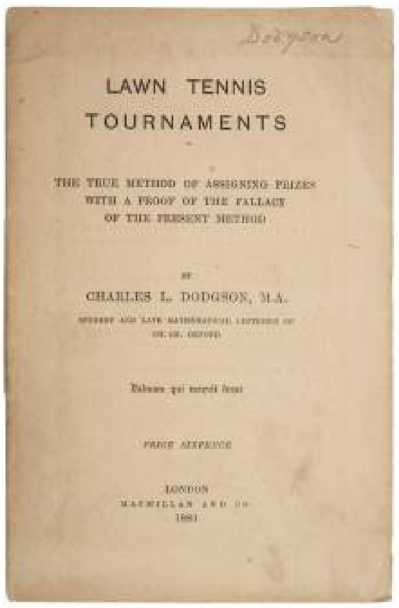
- I allocated the initial pairs of players (and subsequent matchups) at random rather than alphabetically based on their names
- I drop the idea of a "round" and instead run a loop looking for the next individual match to play, matching where possible

players with the same number of games played and losses as eachother. I made up a concept of the "awkward player" at any moment – a player who has played as few games as anyone, and has the least number of legitimate opponents available with the same number of losses, avoiding rematches, etc. Finding a match for that awkward player becomes the priority in each iteration of my loop.


"Do you mean that you think you can find out the answer to it?" said the March Hare.
"Exactly so," said Alice.
"Then you should say what you mean," the March Hare went on.
"I do," Alice hastily replied; "at least—at least I mean what I say—that's the same thing, you know."
"Not the same thing a bit!" said the Hatter. "Why, you might just as well say that 'I see what I eat' is the same thing as 'I eat what I see'!"

- I had to allow, in some circumstances, matches between players who had played a different number of games to that point. There's probably a solution that doesn't require this but I couldn't find it in my time budget.
- I also couldn't think of a practical way to implement the requirement "avoiding, as far as possible, pairing two Players who have a common superior". I did filter out re-matches except in unusual situations to avoid the algorithm getting stuck.
- With non-deterministic results and rematches allowed when unavoidable, a number of contradictions become possible and need to be carefully handled. For example, a player can become a superior of themselves (in the situation A is beaten by B; then B is beaten by A in a rematch – now future A is a 'superior' of past A, a situation that I ruled out by refusing to count).
- It's possible for the four remaining players all to graduate to having 3 superiors each as a result of a single match late in the tournament, leaving no clear placings. In this situation I made them joint first place, but more realistic would of course be a semi- and grand final series.
- Similarly, it's possible for players 2 and 3 (of three remaining) to be knocked out in one step. This means no grand final, but a play-off is needed for second place.

All up, this was a fun exercise. I'm glad this approach to running a tournament works fairly well, even at the cost of roughly twice as many matches needed as in a seeded single-elimination. And it stands up not too badly even with realistically uncertain and inconsistent match outcomes. But of course, it's not as perfect as the ideal deterministic world described in Dodgson's original monograph. Things can get pretty complicated, as the above list of decisions and challenges begins to show. There are some quite awkward edge cases that I didn't stop to sort in detail.


Dodgson

LAWN TENNIS
TOURNAMENTS

THE TRUE METHOD OF ASSIGNING PRIZES
WITH A PROOF OF THE FALLACY
OF THE PRESENT METHOD

BY
CHARLES L. DODGSON, M.A.
STUDENT AND LATE MATHEMATICAL LECTURER OF
CH. CH. OXFORD

Palmam qui meruit ferat

PRICE SIXPENCE

LONDON
MACMILLAN AND CO.
1883

I'm not aware Dodgson's method has ever been used for a real life tournament schedule, although there have been a few simulation experiments similar to my own. I'm not sure how seriously he meant it to be taken; the same monograph proposing this method also suggests in passing tennis should get rid of sets altogether and replace them with a simpler method such as "he who first wins 14 games, or who gets 9 ahead, wins the match". I doubt he seriously expected that proposal to be taken up. But it's worth noting this monograph was published under his professional persona as recreational mathematics enthusiast Charles Dodgson, rather than as children's author Lewis Carroll.

Let's just say that, like most of his other work, it's good he published this! But I would hesitate to recommend adopting it in toto.

Code

Here's all the R code for these simulations in a single chunk. A `carroll_tournament()` function is the work horse, with a couple of helpers for very specific tasks such as `one_match()` which plays out a single match using either a deterministic or probabilistic method of allocating a winner.

```
library(foreach)
library(doParallel)
library(tidyverse)
library(scales)
library(kableExtra)
library(clipr)
library(deuce)



#' Play out a single match between two players
#'
#' @param p1_elo Elo rating of player 1
#' @param p2_elo Elo rating of player 2
#' @param method whether the winner is chosen at random (but with
higher Elo rating meaning a higher
#' chance of winning) or deterministically (in which case the player
with the higher Elo rating always wins)
#' @details draws are not supported. If method is deterministic and
the Elo ratings are equal, player 1 wins.
#' @return a 2 if player 2 wins, and 1 if player 1 wins
one_match <- function(p1_elo, p2_elo, method = c("probabilistic",
"deterministic")){
  method <- match.arg(method)

  if(method == "probabilistic"){
    res <- rbinom(1, 1, prob = deuce::elo_prediction(p2_elo, p1_elo))
+ 1
  } else {
    res <- ifelse(p2_elo > p1_elo, 2, 1)
  }
  return(res)

}



#' Play out a tournament along the lines of Lewis Carroll's preferred
methold
#'
#' @param players a data frame or tibble with columns for player_id
and Elo
#' @param verbose Whether to print out the number of remaining players
in each round
#' @param method Whether individual match results are probabilistic
(realistic) or deterministic (player
#' with higher rating always wins)
#' @details complete works of Lewis Carroll available at
http://www.gasl.org/refbib/Carroll__Works.pdf
#' Description of a better lawn tennis tournament begins on page 1,082
carroll_tournament <- function(players, verbose = FALSE,
                               method = c("probabilistic",
"deterministic")){
  method <- match.arg(method)

  match_wins <- tibble(
    winner_id = character(),
    loser_id = character(),
    combo = character()
  )

  we_have_finalists <- FALSE

  while(!we_have_finalists){
    match_counts <- match_wins %>%
      gather(role, player_id, -combo) %>%
      group_by(player_id) %>%
      summarise(matches = n())

    direct_losses <- match_wins %>%
      group_by(loser_id) %>%
      summarise(losses = length(unique(winner_id))) %>%
      rename(player_id = loser_id)

    indirect_losses <- players %>%
      select(player_id) %>%
      inner_join(match_wins, by = c("player_id" = "loser_id")) %>%
      rename(first_round_winner = winner_id) %>%
      select(-combo) %>%
      inner_join(match_wins, by = c("first_round_winner" =
"loser_id")) %>%
      rename(indirect_superior = winner_id) %>%
      # remove people who are direct superiors so they don't get
counted twice:
      anti_join(match_wins, by = c("player_id" = "loser_id",
"indirect_superior" = "winner_id")) %>%
      # you can't be an indirect superior to yourself (this happens if
you beat someone who beat you before):
      filter(indirect_superior != player_id) %>%
      # you can't have someone you have beaten directly as an indirect
superior:
      anti_join(match_wins, by = c("player_id" = "winner_id",
"indirect_superior" = "loser_id")) %>%
      group_by(player_id) %>%
      summarise(indirect_superiors = length(unique(indirect_
superior)))

    superiors <- direct_losses %>%
      left_join(indirect_losses, by = "player_id") %>%
      mutate(indirect_superiors = replace_na(indirect_superiors, 0),
             superiors = losses + indirect_superiors)

    all_players <- players %>%
      left_join(superiors, by = "player_id") %>%
      left_join(match_counts, by = "player_id") %>%
      mutate(superiors = replace_na(superiors, 0),
             losses = replace_na(losses, 0),
             matches = replace_na(matches, 0))

    remaining_players <- all_players %>%
      filter(superiors < 3) %>%
      select(player_id, matches, elo)

    left <- nrow(remaining_players)

    if(left == 3){
      # It's possible to get stuck with only two superiors at this
point
      remaining_players <- all_players %>%
        filter(superiors < 2) %>%
        select(player_id, matches, elo)
      left <- nrow(remaining_players)

    }

    if(verbose){print(left)}

    if(left <= 2){
      we_have_finalists <- TRUE
    } else {

      candidate_players <- remaining_players %>%
        filter(matches <= min(matches))

      if(nrow(candidate_players) == 1 & left > 1){
        low_match_player <- pull(candidate_players, player_id)
        candidate_players <- remaining_players %>%
          filter(matches <= (min(matches)) + 1)
      } else {
        low_match_player <- NULL
      }

      # player ids of remaining players who have played minimum
matches and are a candidate for the next match:
      rpi <- candidate_players$player_id

      # possible matches (ie excluding rematches) of the candidate
players:
      possible_matches <- expand_grid(p1 = rpi, p2 = rpi) %>%
        # limit to those with id 1 less than id 2 - just for sorting
purposes
        filter(p1 < p2) %>%
        mutate(combo = paste(p1, p2)) %>%
        # restrict to those who have lost the same number of games as
eachother
        left_join(all_players[ , c("player_id", "losses")], by =
c("p1" = "player_id")) %>%
        rename(p1_losses = losses) %>%
        left_join(all_players[ , c("player_id", "losses")], by =
c("p2" = "player_id")) %>%
        rename(p2_losses = losses) %>%
        filter(p1_losses == p2_losses) %>%
        # restrict to people who haven't played eachother:
        anti_join(match_wins, by = "combo")

      get_awkward_player <- function(possible_matches,
low_match_player){

        if(is.null(low_match_player)){
          # find the player who has the least options for playing
others
          awkward_player <- possible_matches %>%
            gather(order, player, -combo) %>%
            group_by(player) %>%
            summarise(count = n()) %>%
            arrange(count, runif(n())) %>%
            slice(1) %>%
            pull(player)
        } else {
          awkward_player <- low_match_player
        }
        return(awkward_player)
      }

      awkward_player <- get_awkward_player(possible_matches,
low_match_player)

      # find a match with that player
      the_match <- possible_matches %>%
        filter(p1 == awkward_player | p2 == awkward_player) %>%
        sample_n(1)


      if(nrow(the_match) == 0){
        # need to let a rematch, or a matchhappen if there is no
alternative:
        warning("Rematch or match of players with unequal number of
losses")
        possible_matches <- expand_grid(p1 = rpi, p2 = rpi) %>%
          # limit to those with id 1 less than id 2 - just for sorting
purposes
          filter(p1 < p2) %>%
          mutate(combo = paste(p1, p2))

        awkward_player <- get_awkward_player(possible_matches,
low_match_player)

        the_match <- possible_matches %>%
          filter(p1 == awkward_player | p2 == awkward_player) %>%
          sample_n(1)
      }

      the_match <- the_match  %>%
        left_join(players[ , c("player_id", "elo")], by = c("p1" =
"player_id")) %>%
        rename(p1_elo = elo) %>%
        left_join(players[ , c("player_id", "elo")], by = c("p2" =
"player_id")) %>%
        rename(p2_elo = elo)

      winner <- one_match(p1_elo = the_match$p1_elo, p2_elo =
the_match$p2_elo, method = method)

      if(winner == 1){
        new_result <- select(the_match, p1, p2, combo)
      } else {
        new_result <- select(the_match, p2, p1, combo)
      }
      names(new_result) <- c("winner_id", "loser_id", "combo")

      match_wins <- rbind(match_wins, new_result)
    }
  }
  if(left == 2){
    grand_final <- one_match(p1_elo = remaining_players[1, ]$elo,
                             p2_elo = remaining_players[2, ]$elo,
                             method = method)

    first_place <- remaining_players[grand_final, ]$player_id
    second_place <- remaining_players[-grand_final, ]$player_id

    final_match <- tibble(
      winner_id = first_place,
      loser_id = second_place,
      combo = paste(sort(c(first_place, second_place)), collapse = "
")
    )

    match_wins <- rbind(match_wins, final_match)

    }
  if(left == 0){
    # it's possible for the last 4 players to be knocked out
simultaneously. left == 0 in this case
    warning("There was no grand final, the last four players were
knocked out at once")
    first_place <- superiors %>%
      filter(superiors == min(superiors)) %>%
      filter(losses == min(losses)) %>%
      pull(player_id) %>%
      paste(., collapse = " | ")

    second_place <- "Not awarded"
    third_place <- "Not awarded"

  }



  if(left == 1){
      # it's possible for players 2 and 3 to be knocked out
simultaneously, in which case
      # there is one remaining player and no grand final. left == 1 in
this case
      warning("There was no grand final, a player won by default")
      first_place <- remaining_players$player_id

      second_place_possibles <- superiors %>%
        filter(!player_id %in% first_place) %>%
        filter(superiors == min(superiors)) %>%
        # if there are people with the same number of superiors at
this point, the place
        # goes to the person with the less direct losses
        filter(losses == min(losses)) %>%
        pull(player_id)

      if(length(second_place_possibles) == 2){
        warning("Needed a semi final play-off for second position")
        semi_players <- tibble(player_id = second_place_possibles) %>%
          inner_join(players, by ="player_id")
        semi_final <- one_match(p1_elo = semi_players[1, ]$elo,
                                p2_elo = semi_players[2, ]$elo,
                                method = method)
        second_place <- semi_players[semi_final, ]$player_id
        third_place <- semi_players[3 - semi_final, ]$player_id
      } else {
      second_place <- paste(second_place_possibles, collapse = " | ")
      }

      if(grepl(" | ", second_place, fixed = TRUE)){
        # if there were 3 second place possibles we give up at this
point..
        third_place <- "Not awarded"
      }

    }

  if(!exists("third_place")){

     third_place <- superiors %>%
      filter(!player_id %in% c(first_place, second_place)) %>%
      arrange(superiors, losses) %>%
      filter(superiors == min(superiors)) %>%
      filter(losses == min(losses)) %>%
      pull(player_id) %>%
      paste(., collapse = " | ")
  }

  match_wins <- match_wins %>%
    mutate(match = 1:n()) %>%
    arrange(desc(match))

  return(list(
    match_wins = match_wins,
    top3 = c(first_place, second_place, third_place))
  )
}

#-------------Data prep---------------

# Identify the top 128 women playing in 1990 and their last Elo rating
that year
data(wta_elo)

women <- wta_elo %>%
  filter(year(tourney_start_date) == 1990) %>%
  # pick just the latest Elo rating
  group_by(player_name) %>%
  arrange(desc(tourney_start_date)) %>%
  slice(1) %>%
  ungroup() %>%
  arrange(desc(overall_elo)) %>%
  dplyr::select(player_id = player_name, elo = overall_elo) %>%
  slice(1:128)

# example usage:
set.seed(142)
```

```
carroll_tournament(women)
carroll_tournament(women, method = "deterministic")

# note that as currently written, rematches don't do anything unless
they come out with the other result
# this would not be popular!


#-----------set up parallel processing-----------

cluster <- makeCluster(7) # only any good if you have at least 7
processors :)
registerDoParallel(cluster)

clusterEvalQ(cluster, {
  library(foreach)
  library(tidyverse)
  library(deuce)
})

clusterExport(cluster, c("women", "carroll_tournament"))


#-------------------simulation of tournament-------------------
n_sims <- 1000

r1 <- foreach(this_sim = 1:n_sims, .combine = rbind) %dopar% {
  set.seed(this_sim)
  results <- carroll_tournament(women, method = "prob")
  tmp <- as.data.frame(t(results$top3))

  if(ncol(tmp) == 1){
    # very rarely, there's only one person emerging
    tmp$second <- NA
  }

  if(ncol(tmp) == 2){
    # sometimes there's no third person due to complications with
reversing results
    tmp$third <- NA
  }

  names(tmp) <- c("first", "second", "third")
  tmp$sim <- this_sim
  return(tmp)
}

# what order *should* have been returned?
correct <- women[1:3, ] %>%
  select(correct_player = player_id) %>%
  mutate(place = c("first", "second", "third"))

# Can it get the top three right as promised:
top3 <- r1 %>%
  gather(place, player, -sim) %>%
  left_join(correct, by = "place") %>%
  group_by(sim) %>%
  summarise(correct_sequence = sum(player == correct_player),
            correct_approx = sum(player %in% correct_player)) %>%
  group_by(correct_sequence, correct_approx) %>%
  summarise(freq = n()) %>%
  arrange(desc(correct_approx), desc(correct_sequence)) %>%
  mutate(looking_at = 3)%>%
  ungroup()

# Can it at least get the top two right:
top2 <- r1 %>%
  gather(place, player, -sim) %>%
  filter(place %in% c("first", "second")) %>%
  left_join(correct, by = "place") %>%
  group_by(sim) %>%
  summarise(correct_sequence = sum(player == correct_player),
            correct_approx = sum(player %in% correct_player)) %>%
  group_by(correct_sequence, correct_approx) %>%
  summarise(freq = n()) %>%
  arrange(desc(correct_approx), desc(correct_sequence)) %>%
  mutate(looking_at = 2)%>%
  ungroup()

# overall winner
top1 <- r1 %>%
  gather(place, player, -sim) %>%
  filter(place %in% c("first")) %>%
  left_join(correct, by = "place") %>%
  group_by(sim) %>%
  summarise(correct_approx = sum(player == correct_player)) %>%
  group_by(correct_approx) %>%
  summarise(freq = n()) %>%
  arrange(desc(correct_approx)) %>%
  mutate(looking_at = 1) %>%
  ungroup()

rbind(top3, top2) %>%
  select(-correct_sequence) %>%
  rbind(top1) %>%
  group_by(looking_at) %>%
  summarise(prop_correct = sum(freq[correct_approx == looking_at]) /
sum(freq))

# chart of results
top1 %>%
  mutate(correct_sequence = correct_approx) %>%
  rbind(top3) %>%
  rbind(top2) %>%
  group_by(looking_at) %>%
  summarise(correct_approx = sum(freq[correct_approx == looking_at]) /
sum(freq),
            correct_exact = sum(freq[correct_sequence == looking_at])
/ sum(freq)) %>%
  gather(variable, value, -looking_at) %>%
  ungroup() %>%
  mutate(variable = ifelse(variable == "correct_approx", "In the
correct prize group but not necessarily sequence", "Correct
sequence")) %>%
  ggplot(aes(x = looking_at, y = value, colour = variable )) +
  geom_line() +
  geom_point() +
  scale_y_continuous(label = percent_format(accuracy = 1)) +
  scale_x_continuous(breaks = 1:3) +
  theme(panel.grid.minor.x = element_blank()) +
  expand_limits(y = 0) +
  labs(x = "Number of top players of interest",
       y = "Probability those players get prizes at correct levels",
       colour= "",
       subtitle = "Chance of the top one, two or three players ending
in correct place in the tournament",
       title = "Performance of Lewis Carroll's tennis tournament
rules",
       caption = "Analysis by http://freerangestats.info of simulated
tournaments of the top 128 women players in 1990, with probabilistic
match outcomes")


#-------------check that deterministic approach always works for top
3------------

n_sims <- 100

r2 <- foreach(this_sim = 1:n_sims, .combine = rbind) %dopar% {
  set.seed(this_sim)
  results <- carroll_tournament(women, method = "det")
  tmp <- as.data.frame(t(results$top3))

  if(ncol(tmp) == 1){
    # very rarely, there's only one person emerging
    tmp$second <- NA
  }

  if(ncol(tmp) == 2){
    # sometimes there's no third person due to complications with
reversing results
    tmp$third <- NA
  }

  names(tmp) <- c("first", "second", "third")
  tmp$sim <- this_sim
  return(tmp)
}

stopCluster(cluster)

r2 %>%
  gather(place, player, -sim) %>%
  left_join(correct, by = "place") %>%
  group_by(sim) %>%
  summarise(correct_sequence = sum(player == correct_player),
            correct_approx = sum(player %in% correct_player)) %>%
  group_by(correct_sequence, correct_approx) %>%
  summarise(freq = n()) %>%
  arrange(desc(correct_approx), desc(correct_sequence))
```