Arthur's Technologiewahlmodell

Claudius Gräbner

4/14/2020

Inhaltsverzeichnis

1 Einführung

Dieses Dokument beschreibt die Grundstruktur von Brian Arthur's Technologiewahlmodell. Neben der Einführung in das Modell soll dabei auch die Implementierung in R vermittelt werden. Das in diesem Text eingeführte Modell bildet auch die Grundlage für die Shiny-App, welche die Vorlesung ergänzt.

Die Beschreibung hier ist recht ausführlich. Wem es genügt: in der App selbst ist eine Beschreibung der wichtigsten Parameter und ganz am Ende dieses Dokuments werden die Leitfragen noch einmal ausführlicher erläutert. Dies mag genügen, wenn Sie sich nicht zu sehr im Detail mit den zugrundeliegenden Mechanismen oder Fragen der Implementierung beschäftigen wollen. Für alle anderen bietet die hier folgende Beschreibung eine genauere Einführung in das Modell und die Implementierung in R.

Die folgenden R-Pakete und Farben werden dabei im Text verwendet:

```
library(tidyverse)
library(latex2exp)
library(ggpubr)
library(igraph)
col_1 <- "#006600"
col_2 <- "#800000"
col_3 <- "#004c93"
col_4 <- "1a171b"
```

2 Motivation

Wie der Name schon sagt geht es im Modell um das Entscheidungsverhalten von Personen, die sich zwischen verschiedenen Technologien entscheiden müssen. Die Motivation des Modells war die Beobachtung, dass a) bereits etablierte Technologien häufig nicht mehr von neuen - und möglicherweise besseren - Technologien verdrängt werden und b) Menschen häufig Technologien verwenden, die sie eigentlich anderen Technologien gegenüber für inferior halten. In solchen Fällen sprechen wir von einem *technologischen lock-in*: eine bestimmte Technologie ist am Markt derart stark vertreten, dass sie von anderen Technologien nicht mehr verdrängt werden kann.

Das Phänomen existiert nicht erst seit der Relevanz digitaler Technologien, allerdings bietet dieser Bereich eine besondere Fülle für entsprechende Beispiele: Google, Facebook, WhatsApp - all das sind Technologien, die - teils trotz enormer Kritik - eine enorm stabile Position im Markt haben.

Arthur's Kernargument ist, dass es besonders bei solchen Technologien leicht zu einem lock-in kommt, bei denen der Gesamtnutzen der Technologie nicht nur von der *intrinsischen Qualität* der Technologie, sondern auch von der *Nutzeranzahl* abhängt. Dann wird eine Technologie unter Umständen selbst dann nicht aus dem Markt gedrängt, wenn es eigentlich bessere Alternativen gäbe.

Ein mögliches Beispiel wäre der Messanger WhatsApp: ich persönlich finde Signal besser als WhatsApp. Der Schutz persönlicher Daten ist mir wichtig und da ich ansonsten den Messanger nur zum Schreiben und Telefonieren verwende sind die Sonderfunktionen von WhatsApp nicht relevant für mich. Für mich hat Signal also den *höheren intrinsischen Nutzen*. Bedeutet das, dass ich mich auf jeden Fall für Signal entscheiden werde? Nein, denn wenn alle meine Freunde WhatsApp verwenden bringt mir Signal nichts. Der *Netzwerknutzen*, also die Nutzenkomponente, die sich aus der Anzahl der Nutzer ergibt, ist bei WhatsApp so viel größer als bei Signal, dass ich mich am Ende vielleicht dennoch für WhatsApp entscheide - ein Fall für technologischen lock-in.

Natürlich enthält diese Beschreibung zahlreiche Annahmen über das Entscheidungsverhalten. Das Schöne an Arthur's Modell ist die einfache Möglichkeit der Erweiterung: es startet von einem relativ simplen (fast 'simplistischen') Ausgangspunkt, kann aber leicht um immer neue Features erweitert werden, z.B. um die Relevanz sozialer Netzwerke, die Anzahl der Technologien und vieles mehr. Im Folgenden soll zunächst das einfache Grundmodell eingeführt werden und dann eine Erweiterung beispielhaft diskutiert werden.

3 Der einfache Ausgangsfall

3.1 Theorie

Betrachten wir eine Situation, in der sich N Personen nacheinander zwischen zwei Technologien, A und B, entscheiden müssen. Jede Person muss sich dabei für eine der beiden Technologien entscheiden.

Sei der Nutzen der Technologie $i \in \{A, B\}$, u_i gegeben durch:

$$u_i = r_i + v(n_i)$$

Hier ist r_i der *intrinsische Nutzen* von Technologie i - also der Wert, den ich der Technologie unabhängig der Nutzer*innenzahl beimessen würde. Die Funktion $v(n_i)$ misst dabei den *Netzwerknutzen* und ist daher eine Funktion von der Anzahl der Nutzer*innen von i, bzw. dem Anteil der Personen in der Gesamtpopulation N, die Technologie i nutzen. Im folgenden nehmen wir an, dass $v(n_i)$ eine linear wachsende Funktion ist.

Definieren wir nun $\delta = r_A - r_B$. Dann können wir schreiben:

$$u_A - u_B = \delta + v(n_A - n_B)$$

Eine einzelne Person würde sich für Technologie A entscheiden wenn $u_A - u_B > 0$. Fälle, in denen alle Personen die gleichen Präferenzen haben sind nicht sonderlich interessant. Relevanter wäre ein Fall in dem unsere Population aus zwei Gruppen besteht: Personen in der ersten Gruppe finden Technologie A im Bezug auf ihren intrinsischen Nutzen besser, es gilt also $r_A^1 > r_B^1$. Bei den Mitgliedern der zweiten Gruppe ist es genau anders herum: für sie hat Technologie B einen höheren intrinsischen Nutzen, für sie gilt $r_A^2 < r_B^2$. Welche Technologie wird sich also durchsetzen? Das hängt natürlich von der relativen Größe der relevanten Parametern r_A^1, r_B^1, r_A^2 und r_B^2, der genauen funktionalen Form von $v(\cdot)$ und der Reihenfolge, in der sich die Nutzer*innen entscheiden müssen, ab.

Für die vorliegende App habe ich mich entschieden als Netzwerknutzen einfach den Anteil der relevanten Personen zu nehmen, welche die jeweilige Technologie verwenden. Im einfachsen Falle sind alle Personen in der Population gleich relevant, es gilt also:

$$v(n_i) = \frac{n_i}{n_A + n_B}$$

Der intrinsische Nutzen wird als exogener Parameter gesetzt. In der Shiny-App können Sie die Implikationen verschiedener Werte untersuchen. Gehen wir von einem Wert von 0.5 für die jeweilige Gruppe aus, also $r_A^1 = 0.5$ und $r_B^1 = 0$, sowie $r_A^2 = 0$ und $r_B^2 = 0.5$. Das bedeutet, dass sich ein Agent aus Gruppe 1 gegen seine intrinsische Präferenz entscheiden würde, wenn mehr als die Hälfte der für ihn relevanten Personen bereits Technologie B benutzen würden. Illustrieren wir das für die Situation in der sich 8 von 10 Personen bereits für Technologie B entschieden haben. Dann gilt für unseren Agenten:

$$u_A - u_B = (r_A - r_B) + v(n_A - n_B) \tag{1}$$

$$u_A - u_B = (0.5 - 0.0) + \frac{2}{2+8} - \frac{8}{2+8} \tag{2}$$

$$u_A - u_B = 0.5 + 0.2 - 0.8 = -1 \tag{3}$$

Entsprechend wird sich die Person - trotz intrinsischer Präferenz für Technologie A für Technologie B entscheiden.

3.2 Implementierung in R

3.2.1 Setup

Zunächst legen wir die Anzahl der potenziellen Nutzer*innen und die Anzahl der Technologien fest. Für das Design betrachten wir den einfachen Fall mit 6 Nutzer*innen und zwei Technologien, später betrachten wir 50 Agenten. In der App können Sie auch mehrere Technologien auswählen.[1] Die Technologien speichern wir als Faktor. Zudem legen wir den jeweiligen intrinsischen Nutzen r_i mit `0.5` fest:

```
n_techs <- 2
all_techs <- factor(1:n_techs)
intrinsic_utilities <- rep(0.5, n_techs)
n_agents <- 4
agents_id <- 1:n_agents
```

[1] Wichtig ist hierbei, dass die Anzahl der Agenten ohne Rest durch die Anzahl der Technologien teilbar ist. In der App wurde eine Funktion geschrieben, die im Zweifel die Anzahl der Agenten erhöht. Solche Spitzfindigkeiten können Sie sich im Quellcode der App bei Interesse genauer ansehen.

Die einzelnen Agenten speichern wir als Liste, da wir ihnen ja auch eine Präferenz für eine der beiden Technologien mitgeben müssen. Die Hälfte der Agenten präferiert Technologie A, die andere Hälfte Technologie B:

```
all_agents <- list()
for (i in agents_id[1:(n_agents/2)]){
  all_agents[[i]] <- list()
  all_agents[[i]][["id"]] <- i
  all_agents[[i]][["preferred_tech"]] <- all_techs[1]
  all_agents[[i]][["tech_chosen"]] <- NA
}

for (i in agents_id[(n_agents/2+1):n_agents]){
  all_agents[[i]] <- list()
  all_agents[[i]][["id"]] <- i
  all_agents[[i]][["preferred_tech"]] <- all_techs[2]
  all_agents[[i]][["tech_chosen"]] <- NA
}
```

Zudem legen wir eine Liste an, in der wir später speichern ob ein bestimmter Agent diese Technologie nutzt:

```
shares_list <- list()
for (t in all_techs){
  shares_list[[t]] <- rep(NA, n_agents)
}
```

3.2.2 Die Technologiewahlfunktion

Als nächstes schreiben wir noch eine Funktion für die eigentliche Technologiewahl. Diese entwickeln wir Schritt für Schritt. Als erstes benötigen wir einen Vektor mit allen bislang gewählten Technologien. Wir müssen also aus der Liste `all_agents` für jeden Agenten den Wert des Eintrags `tech_chosen` auswählen. Das geht mit der Funktion `sapply`:

```
chosen_techs <- sapply(all_agents, "[[", "tech_chosen")
chosen_techs <- factor(chosen_techs, levels = all_techs)
chosen_techs
```

```
#> [1] <NA> <NA> <NA> <NA>
#> Levels: 1 2
```

Da bisher noch kein Agent seine Technologie gewählt hat, ist das richterweise ein Vektor der Länge `n_agents`, der ausschließlich fehlende Werte enthält. Die Umwandlung des Ergebnis in einen Faktor ist wichtig, damit wir später beim Zählen auch die Technologien berücksichtigen, die noch gar nicht gewählt wurden. Diese sollen nicht übersprungen werden, sondern sollen explizit den Wert `0`erhalten, da sie zwar existieren, aber von keinem Agenten gewählt worden sind.

Im vorliegenden Fall sollte der Agent also einfach seine präferierte Technologie wählen, denn es gibt ja noch keinen relevanten Netzwerknutzen:

```
current_agent <- 1
if (length(chosen_techs[!is.na(chosen_techs)])==0){
  all_agents[[current_agent]][["tech_chosen"]] <-
    all_agents[[current_agent]][["preferred_tech"]]
}
```

Der entsprechende Eintrag von Agent `1` wurde nun entsprechend angepasst:

```
all_agents[[current_agent]]
```

```
#> $id
#> [1] 1
#>
#> $preferred_tech
#> [1] 1
#> Levels: 1 2
#>
#> $tech_chosen
#> [1] 1
#> Levels: 1 2
```

Und eine erneute Berechnung der bislang gewählten Technologien gibt das zu erwartende Ergebnis:

```
chosen_techs <- sapply(all_agents, "[[", "tech_chosen")
chosen_techs <- factor(chosen_techs, levels = all_techs)
chosen_techs
```

```
#> [1] 1    <NA> <NA> <NA>
#> Levels: 1 2
```

Wenn bereits andere Agenten eine Technologie gewählt haben müssen wir noch den Netzwerknutzen berechnen. Im vorliegenden Fall wollen wir als Netzwerknutzen den relativen Anteil der Technologie wählen. Den können wir mit den Funktionen `table` und `prop.table` berechnen:[2]

```
abs_freqs <- table(chosen_techs)
rel_freqs <- prop.table(abs_freqs)
rel_freqs
```

```
#> chosen_techs
#> 1 2
#> 1 0
```

Jetzt müssen wir noch für jede Technologie den Gesamtnutzen ausrechnen. Wie oben beschrieben setzt sich dieser aus dem intrinsischen Nutzen r_i und dem Netzwerknutzen zusammen. Den Wert für den intrinsischen Nutzen haben wir ja als Parameter `intrinsic_utilities` gesetzt, für den Netzwerknutzen nehmen wir einfach den relativen Nutzer*innenanteil:

```
utilities <- unname(rel_freqs) + intrinsic_utilities
utilities
```

```
#> [1] 1.5 0.5
```

Nun soll der Agent einfach die Technologie mit dem höchsten Gesamtnutzen wählen. Dazu verwenden wir zunächst die Funktion `which`, die uns den Index der Technologie mit dem größten Nutzen gibt, und wählen dann die Technologie mit diesem Index aus der Liste aller Technologien aus:

```
index_max <- which(utilities==max(utilities))
tech_chosen <- all_techs[index_max]
tech_chosen
```

```
#> [1] 1
#> Levels: 1 2
```

[2] Eine wichtige Designentscheidung hier ist ob Agenten, die bisher noch keine Technologie gewählt haben, in die Berechnung des relativen Anteils mit einfließen sollten. Falls nicht könnte man jeden Agenten mit einer Technologie `0` beginnen lassen. Sie können die Implikation dieser Entscheidung in der App untersuchen (siehe Leitfrage 4), in der Modellbeschreibung wird sie im Folgenden nicht explizit diskutiert.

Falls zwei Technologien den gleichen Nutzenwert haben sollten, soll sich der Agent zufällt entscheiden:

```
index_max <- which(utilities==max(utilities))
if (length(index_max)>1){
  index_max <- sample(index_max, 1)
  }
tech_chosen <- all_techs[index_max]
```

Den bisher geschriebenen Code können wir nun zusammenfassen:

```
current_agent <- 2
if (length(chosen_techs[!is.na(chosen_techs)])>0){
  abs_freqs <- table(chosen_techs)
  rel_freqs <- prop.table(abs_freqs)
  utilities <- unname(rel_freqs) + intrinsic_utilities
  index_max <- which(utilities==max(utilities))
  if (length(index_max)>1){
    index_max <- sample(index_max, 1)
    }
  tech_chosen <- all_techs[index_max]

  all_agents[[current_agent]][["tech_chosen"]] <- tech_chosen
}
```

Wie zu erwarten wählt Agent 2 die erste Technologie:

```
all_agents[[current_agent]][["tech_chosen"]]
```

```
#> [1] 1
#> Levels: 1 2
```

Der Einfachheit halber fassen wir die Technologiewahl jetzt noch in einer Funktion zusammen, die wir dann für jeden Agenten aufrufen können. Daher ändern wir den Code insofern leicht ab, dass die Liste aller Agenten noch nicht direkt modifiziert wird, sondern nur die gewählte Technologie ausgegeben wird:

```
#' Ein Agent wählt eine Technologie
#'
#' @param current_agent_id Die id des wählenden Agenten
#' @param agent_list Die Liste aller Agenten und deren Technologiewahl
#' @param all_techs Die Liste aller Techologien
#' @param intrinsic_utilities Die intrinsischen Nutzenwerte für alle Techs
#' @return Die gewählte Technologie
choose_tech <- function(current_agent_id, agent_list, all_techs){
  chosen_techs <- sapply(agent_list, "[[", "tech_chosen")
  chosen_techs <- factor(chosen_techs, levels = all_techs)

  if (length(chosen_techs[!is.na(chosen_techs)])==0){
    tech_chosen <- agent_list[[current_agent_id]][["preferred_tech"]]
  } else if (length(chosen_techs[!is.na(chosen_techs)])>0){
    abs_freqs <- table(chosen_techs)
    rel_freqs <- prop.table(abs_freqs)
    utilities <- unname(rel_freqs) + intrinsic_utilities
    index_max <- which(utilities==max(utilities))
    if (length(index_max)>1){
      index_max <- sample(index_max, 1)
      }
```

```
    tech_chosen <- all_techs[index_max]
  }
  return(tech_chosen)
}
```

Nachdem wir die im Beispiel oben durchgeführten Aktionen zurückgesetzt haben funtioniert die Funktion wie erwartet:

```
choose_tech(current_agent_id = 1,
            agent_list = all_agents, all_techs = all_techs)
```

```
#> [1] 1
#> Levels: 1 2
```

3.2.3 Technologiewahl für alle Agenten

Nun können wir die Reihenfolge in der die Agenten ihre Technologie wählen festlegen. Da es das Ergebnis verzerren würde, wenn zuerst die Mitglieder der einen und dann die Mitglieder der anderen Gruppe ihre Technologie wählen würden (warum?) legen wir die Reihenfolge zufällt fest. Dafür verwenden wir die Funktion `sample()`:

```
agents_sequence <- sample(agents_id)
```

Das bedeutet auch, dass wir für jeden Durchlauf des Modells ein anderes Ergebnis bekommen. Wir verwenden daher Monte-Carlo Simulationen: Wir simulieren das Modell häufig und analysieren dann die durchschnittlichen Ergebnisse. Doch dazu später mehr. Zunächst führen wir einen einzelnen Simulationsdurchlauf durch:

```
for (v in 1:length(agents_sequence)){
  current_agent <- agents_sequence[v]
  chosen_tech <- choose_tech(
    current_agent_id = current_agent,
    agent_list = all_agents,
    all_techs = all_techs)

  all_agents[[current_agent]][["tech_chosen"]] <- chosen_tech

  chosen_techs <- factor(sapply(all_agents, "[[", "tech_chosen"),
                         levels = all_techs)
  current_rel_freqs <- prop.table(table(chosen_techs))

  for (t in all_techs){
    shares_list[[t]][v] <- current_rel_freqs[t]
  }
}
```

Die Liste `shares_list` gibt uns nun die relative Verteilung nach jeder Technologiewahl an:

```
shares_list
```

```
#> $`1`
#> [1] NA NA NA NA
#>
#> $`2`
#> [1] NA NA NA NA
```

Das können wir noch etwas schöner als Tabelle formatieren:

```
simulation_result <- tibble(
  "t"=1:n_agents,
  "Anteil Tech A"=shares_list[[1]],
  "Anteil Tech B"=shares_list[[2]])
simulation_result
```

```
#> # A tibble: 4 x 3
#>       t `Anteil Tech A` `Anteil Tech B`
#>   <int> <lgl>           <lgl>
#> 1     1 NA              NA
#> 2     2 NA              NA
#> 3     3 NA              NA
#> 4     4 NA              NA
```

Diesmal haben also alle Agenten Technologie 1 gewählt. Da die Reihenfolge der Agenten zufällig ist, müssen wir das Modell mit Hilfe einer Monte Carlo Simulation untersuchen, also sehr häufig durchführen und deskriptive Statistiken sammeln.

Um das zu erleichtern fassen wir noch die Durchführung einer einzelnen Simulation in einer Funktion zusammen:

```
run_simulation <- function(n_techs, int_utils, n_agents){

  all_techs <- factor(1:n_techs)
  intrinsic_utilities <- rep(int_utils, n_techs)
  agents_id <- 1:n_agents
  agents_sequence <- sample(agents_id)

  shares_list <- list()
  for (t in all_techs){
    shares_list[[t]] <- rep(NA, n_agents)
  }

  all_agents <- list()
  for (i in agents_id[1:(n_agents/2)]){
    all_agents[[i]] <- list()
    all_agents[[i]][["id"]] <- i
    all_agents[[i]][["preferred_tech"]] <- all_techs[1]
    all_agents[[i]][["tech_chosen"]] <- NA
  }

  for (i in agents_id[(n_agents/2+1):n_agents]){
    all_agents[[i]] <- list()
    all_agents[[i]][["id"]] <- i
    all_agents[[i]][["preferred_tech"]] <- all_techs[2]
    all_agents[[i]][["tech_chosen"]] <- NA
  }

  for (v in 1:length(agents_sequence)){
    current_agent <- agents_sequence[v]
    chosen_tech <- choose_tech(
      current_agent_id = current_agent,
      agent_list = all_agents,
      all_techs = all_techs)
```

```
    all_agents[[current_agent]][["tech_chosen"]] <- chosen_tech

    chosen_techs <- factor(sapply(all_agents, "[[", "tech_chosen"),
                           levels = all_techs)
    current_rel_freqs <- prop.table(table(chosen_techs))

    for (t in all_techs){
      shares_list[[t]][v] <- current_rel_freqs[t]
    }
  }

  simulation_result <- tibble(
  "t"=1:n_agents,
  "Anteil Tech A"=shares_list[[1]],
  "Anteil Tech B"=shares_list[[2]])

  return(simulation_result)
}
```

Diese Funktion fasst alle oben durchgeführten Schritte zusammen. Jetzt ist es übrigens auch ganz einfach die Anzahl der Agenten einfach auf 50 oder mehr zu erhöhen:

```
sim_results <- run_simulation(n_techs = 2, int_utils = 0.5, n_agents = 50)
head(sim_results, 2)
```

```
#> # A tibble: 2 x 3
#>       t `Anteil Tech A` `Anteil Tech B`
#>   <int>           <dbl>           <dbl>
#> 1     1               1               0
#> 2     2               1               0
```

3.2.4 Einbettung in eine Monte-Carlo Simulation

Da unsere Simulation Zufallsprozesse enthält wollen wir die Simulation sehr häufig durchführen und dann die Ergebnisse über deskriptive Statistiken zusammenfassen.

Dazu schreiben wir einfach eine 'Meta-Funktion', die unsere Simulation häufig durchführt und dann die Ergebnisse zusammen fasst:

```
run_n_simulations <- function(n_iterations, n_agents, n_techs,
                              intrinsic_preference){
  result_frames <- list()
  result_networks <- list()
  for (i in 1:n_iterations){
    print(paste0("Run simulation ", i, "/", n_iterations))
     result_list <- run_simulation(n_techs = n_techs,
                                   int_utils = intrinsic_preference,
                                   n_agents = n_agents)
     result_frames[[as.character(i)]] <- result_list
  }
  final_frame <- dplyr::bind_rows(result_frames)
  return(final_frame)
}
```

Jetzt ist es einfach die Simulation mehrere Male durchzuführen und die Ergebnisse in einer Tabelle

auszugeben:

```
mcs_ergebnis <- run_n_simulations(
  n_iterations=5, n_agents=20, n_techs=2,
  intrinsic_preference=0.5)
```

```
#> [1] "Run simulation 1/5"
#> [1] "Run simulation 2/5"
#> [1] "Run simulation 3/5"
#> [1] "Run simulation 4/5"
#> [1] "Run simulation 5/5"
```

Diese Ergebnisse können dann visualisiert werden. Hierzu bieten sich z.B. Boxplots an. Den der Visualisieurng der App zugrundeliegenden Code finden Sie im Quellcode der App.

4 Ergänzung Soziale Netzwerke

4.1 Motivation

Der in der vorangegangen Section beschriebene Ausgangsfall operiert natürlich in vielerlei Hinsicht mit sehr unintuitiven Annahmen. So ist für die Berechnung des Netzwerknutzens immer die gesamte Population relevant. Häufig ist für eine Person aber nur die Wahl ihres unmittelbaren Umfelds von Bedeutung: wenn ich mich für einen Messanger entscheide ist mir eigentlich recht egal wie viele Personen in Deutschland oder auf der Welt diesen Messanger verwenden. Relevant ist für mich, wie viele von meinen Bekannten, also den Menschen mit denen ich tatsächlich kommunizieren möchte, den gleichen Messanger verwenden. Eine Möglichkeit das Modell dahingehen zu erweitern ist die Berücksichtigung sozialer Netzwerke. Die kann auch erklären, warum wir trotz der Relevanz von Netzwerknutzen häufig keine komplette Monopolisierung von Technologien beobachten, sondern vielmehr die Vormachtstellung bestimmter Technologien in bestimmten Bevölkerungsgruppen.

In dieser Erweiterung verwenden wir das R-Paket igraph um zumindest einfache Netzwerke im Modell zu berücksichtigen. Dazu werden wir den oben beschriebenen Code leicht adaptieren müssen. Zuerst sollen die in der App verwendeten Netzwerke aber kurz beschrieben werden.

4.2 Vier Typen von Netzwerken

In der App können Sie vier verschiedene Netzwerke oder ('Netzwerktopologien') auswählen. Diese sind in Abbildung 1 dargestellt.

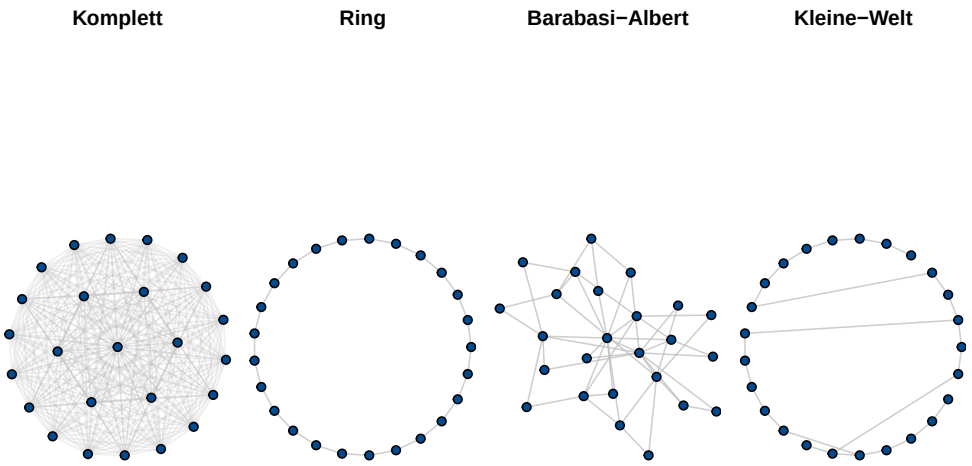

Abbildung 1: Die vier in der App implementierten Netzwerktopologien.

1. Das **komplette Netzwerk**. Hier ist jeder Agent mit jedem anderen Agenten verbunden. Das bedeutet: bei der Berechnung des Netzwerknutzens werden die Entscheidungen aller anderen Agenten berücksichtigt.
2. Beim **Ring** hat jeder Agent zwei Nachbarn. Für die Berechnung des Netzwerknutzens sind nur die Entscheidungen dieser beiden Nachbarn relevant. Die Entscheidungen der anderen Agenten werden nicht berücksichtigt.
3. Bei **skalenfreien Netzwerk** (auch: **Barabasi-Albert-Netzwerk**) folgt die Anzahl der Nachbarn der Agenten einer Potenzverteilung. Es gibt also wenige Agenten, die mit sehr vielen anderen Agenten verbunden sind, und viele Agenten, die nur recht wenige Nachbarn haben. Es gibt also einige wenige zentrale Agenten, deren Wahl viele andere Agenten beeinflusst (aber auch von vielen anderen Agenten beeinflusst wird), und viele Agenten, die sich nur an der der Entscheidung von wenigen anderen Agenten orientieren. Die meisten empirischen Freundschaftsnetzwerke folgen dieser Struktur.
4. Das **Kleine-Welt-Netzwerk** ist ähnlich dem Ring, allerdings gibt es noch einige zusätzliche zufällige Verbindungen. Diese Netzwerke weisen eine für soziale Netzwerke sehr typische Eigentschaft auf: wir interagieren mit recht wenigen Leuten in unserer Nachbarschaft (unseren 'Cliquen'), aber haben darüber hinaus noch einzelne Verbindungen zu anderen Cliquen, sodass die durchschnittliche Entfernung zu anderen Agenten recht klein sind. Diese Netzwerke gehen auf ein Experiment zurück, in dem Menschen einer ihnen unbekannten Zielperson einen Brief schicken sollten, diesen Brief aber nur an ihnen bekannte Personen schicken durften. Diese sollten ihn dann weiterleiten. Diese Studie trug den Titel Six Degrees of Separation und gab dem Kleinen-Welt-Netzwerk seinen Namen.

Netzwerke spielen im Modell also insofern eine Rolle, als dass für die Berechnung des Netzwerknutzens einer Technologie nur die Verteilung der Technologie in der Nachbarschaft des Agenten berücksichtig wird. Der in der vorherigen Section besprochene Spezialfall, in dem die Entscheidungen *aller* anderen Agenten relevant sind, korrespondiert also zu der Situation einer *kompletten Netzwerks*.

4.3 Implementierung in R

Wir werden nun die Agentenpopulation von Anfang an als Netzwerk begreifen. Dabei korrespondiert jeder Agent zu einer Kante ('node' oder 'vertex') im Netzwerk. Um ein Netzwerk zu erstellen verwenden wir die entsprechende Funktion aus dem Paket `igraph`. Für das komplette Netzwerk geht das folgendermaßen:

```
n_agents <- 6
full_network <- make_full_graph(
  n_agents, directed = FALSE, loops = FALSE)
```

Die Argumente `directed=FALSE` und `loops=FALSE` stellen sicher, dass die Verbindungen in unserem Netzwerk keine Richtung haben (wenn Agent i mit Agent j verbunden ist, ist automatisch Agent j mit Agent i verbunden) und kein Agent mit sich selbst verbunden ist. Für die anderen oben beschribenen Netzwerke gibt es eigene Funktionen, auf deren Funktion hier im genaueren nicht eingegangen werden soll:

```
ring_netzwerk <- make_ring(
  n_agents, directed = FALSE, mutual = FALSE,
  circular = TRUE)

ba_netzwerk <- sample_pa(
      n_agents, power = 1.2, m = 2,
      directed = FALSE, algorithm = "psumtree")
```

```
kw_netzwerk <- simplify(sample_smallworld(
  dim = 1, size = n_agents, nei = 1, p = 0.1))
```

Die Agenten sind nun als Kanten des Netzwerks definiert und können mit `V(netzwerk)` angezeigt werden:

```
V(full_network)
```

```
#> + 6/6 vertices, from 8e81e68:
#> [1] 1 2 3 4 5 6
```

Um die Attribute der Agenten festzulegen müssen wir nun besondere Netzwerkfunktionen verwenden. Wo wir oben die Agenten als Liste von Listen erstellt haben spezifizieren wir hier die Attribute der Agenten folgendermaßen:

```
full_network <- set_vertex_attr(
  full_network, "technology",
  index = V(full_network), NA)
```

Wir können uns dieses Attribut nun folgendermaßen anzeigen lassen:

```
V(full_network)$technology
```

```
#> [1] NA NA NA NA NA NA
```

Entsprechend gehen wir auch bei der Spezifizierung der präferierten Technologie vor:

```
full_network <- set_vertex_attr(
  full_network, "preferred_tech",
  index = V(full_network)[1:(n_agents/2)],
  value = all_techs[1])

full_network <- set_vertex_attr(
  full_network, "preferred_tech",
  index = V(full_network)[(n_agents/2+1):n_agents],
  value = all_techs[2])
```

```
V(full_network)$preferred_tech
```

```
#> [1] 1 1 1 2 2 2
```

Auch bei unserer Technolgiewahlfunktion müssen wir noch einige Änderungen vornehmen. So sollen nun bei der Berechnung des Netzwerknutzens nur noch die benachbarten Agenten berücksichtigt werden. Die Nachbarn eines Knotens im Netzwerk können wir durch die Funktion `neighbors` ausgeben lassen. Im kompletten Netzwerk ist jeder Agent mit jedem anderen Agenten benachbart:

```
V(full_network)
```

```
#> + 6/6 vertices, from 8e81e68:
#> [1] 1 2 3 4 5 6
```

```
neighbors(full_network, 1)
```

```
#> + 5/6 vertices, from 8e81e68:
#> [1] 2 3 4 5 6
```

Beim Ring ist das entsprechend anders:

```
V(ring_netzwerk)
```

```
#> + 6/6 vertices, from a0365d3:
```

```
#> [1] 1 2 3 4 5 6
```

```
neighbors(ring_netzwerk, 1)
```

```
#> + 2/6 vertices, from a0365d3:
#> [1] 2 6
```

Wir adaptieren die Technologiewahlfunktion nun so, dass sie bei der Berechnung des Netzwerknutzens nur diese Nachbarschaft berücksichtigt:

```
choose_tech <- function(graph_used, vertex_used, # Angepasst an Netzwerke
                        all_techs, intrinsic_preference){
  neighborhood <- neighbors(graph_used, vertex_used) # Das ist neu
  techs_neighborhood <- factor(
    neighborhood$technology[!is.na(neighborhood$technology)],
    levels = c(levels(all_techs)))
  neighborhood_len <- length(techs_neighborhood)
  if (neighborhood_len == 0){
    tech_chosen <- V(graph_used)[vertex_used]$preferred_tech
  } else{
    abs_freqs <- table(techs_neighborhood)
    rel_freqs <- prop.table(abs_freqs)

    utilities <- unname(rel_freqs)
    utilities <- utilities + intrinsic_utilities
    utilities[V(graph_used)[vertex_used]$preferred_tech] <-
      utilities[V(graph_used)[vertex_used]$preferred_tech] +
      intrinsic_preference

    index_max <- which(utilities==max(utilities))
    if (length(index_max)>1){
      index_max <- sample(index_max, 1)
    }
    tech_chosen <- all_techs[index_max]
  }

  return(tech_chosen)
}
```

Der Rest der Simulation funktioniert genauso wie vorher.

5 Leitfragen an das Modell

Das der App zugrundeliegende Modell ist an der einen oder anderen Stelle noch etwas komplexer als die hier beschriebene Variante. Auf Basis der Beschreibung oben sollten Sie aber in der Lage sein, den Quellcode auf Wunsch selbst zu verstehen und ggf. auch zu adaptieren.

Für die Analyse des Modells bieten sich folgende Leitfragen an:

1. Welchen Einfluss hat die Netzwerktopologie auf die Tendenz zu technologischen Monopolisierung? Warum?

In anderen Worten: macht es für die Technologiewahldynamiken einen Unterschied ob sich die Agenten immer an den Entscheidungen aller anderen orientieren, oder nur an ihrer Nachbarschaft? In welchem Fall kommt es eher zu einer Monopolisierung?

2. Was ist der Einfluss des speziellen Technologienutzens und der gruppenbezogenen Technologiepräferenz?

Der spezielle Technologienutzen ist ein allgemeiner Nutzenbonus, den eine Technologie erhält. Es misst also eine generelle technische Überlegenheit. Die Frage ist: welche Rolle spielt das dafür, ob sich die Technologie tatsächlich auf dem Markt durchsetzt? Und welche Rolle spielt die Stärke der gruppenbezogenen Präferenz?

3. Was ist der Effekt der Anzahl der Technologien und Agenten auf die Modell-Dynamik? Ist dieser Effekt Ihrer Ansicht nach intuitiv?

Das hier besprochene Beispiel war auf zwei Technologien beschränkt. Ändert sich die Kernaussage des Modells wenn mehr als zwei Technologien verwendet werden?

4. Welchen Unterschied macht es ob für die Berechnung des Netzwerknutzens der Anteil der Technologien an den bisher gewählten Technologien, oder der Anteil der Agenten, die diese Technologie verwenden, herangezogen wird? Welche Annahme ist für welche Situation plausibler?

Wie oben beschrieben kann der Netzwerknutzen auf zwei Arten berechnet werden: wir berechnen den Anteil der Agenten, die bei ihrer bereits getroffenen Technologiewahl die Technologie gewählt haben, also:

$$v(n_i) = \frac{n_i}{\sum_{j=1}^{T} n_j}$$

wobei T die Anzahl der Technologien bezeichnet.

Oder aber wir berechnen den Anteil der Agenten, die die Technologie verwenden, berücksichtigen im Nenner also auch Agenten, die bislang noch keine Technologie gewählt haben. Sei N die Anzahl der Agenten in der Nachbarschaft des Agenten. Dann hätten wir für den Netzwerknutzen:

$$v(n_i) = \frac{n_i}{N}$$

5. Unter welchen Umständen beobachten wir trotz unterschiedlicher Qualität der Technologien keine Monopolisierung? Warum?

Intuitiv würde man sagen: die beste Technologie setze sich durch. Arthur's Modell zeigt, dass das häufig aber nicht passiert. Unter werlchen Umständen ist es besonders unwahrscheinlich?