Modeling Interaction Effects in Polarization: Individual Media Influence and the Impact of Town Meetings

Eric Pulick¹, Patrick Korth², Patrick Grim³, Jiin Jung⁴

¹University of Michigan, United States
²University of Michigan, United States
³University of Michigan, United States
⁴Claremont Graduate University, United States

Correspondence should be addressed to patrick.grim@stonybrook.edu

Received: 25-05-2015   Accepted: 25-01-2016   Published: 31-03-2016

Appendix: pseudo-code

agent properties

culture ; A list feature_count values in the range of trait_count
avg_culture ; The average value of the culture list

Global variables

agent_avg ; Overall average agent culture
agent_std_dev ; Overall standard deviation of agent cultures
feature_count
trait_count
interaction_method ('jump','shift')
similarity_method ('axelrod','relative')
minimum_similarity
media_interaction ('nearest','chance')
meeting_interaction ('average','permute')
use_neighbor_interactions
neighbor_interaction_proportion
max_nearby_distance
nearby_interaction_proportion
use_meetings
meeting_interaction_interval
meeting_radius
use_media_sources
media_interaction_interval
media_interaction_proportion

JASSS, 19(2):1, 2016   http://jasss.soc.surrey.ac.uk/19/2/1.html   doi: 10.18564/jasss.3021
media_sources ; List containing the culture lists of enabled media
media_center ; Is the non-bias media source be enabled?
media_center_left ; Is the slightly left-wing media source be enabled?
media_center_right ; Is the slightly right-wing media source be enabled?
media_far_left ; Is the left-wing media source be enabled?
media_far_right ; Is the right-wing media source be enabled?

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Very first function to be called.
function setup() {
    ; Create initial cultures.
    setupAgents()
    setupMediaSources()
}

;; Fill the initial culture array of each agent.
function setupAgents() {
    for (each agent) {
        agent.culture = []
        for (i = 0 to feature_count) {
            agent.culture.append(createCultureValue())
        }
    }
}

;; Populate global media_sources list with the appropriate cultures.
function setupMediaSources() {
    media_sources = []

    ; Add news sources of varying political siding if enabled.
    if (media_far_left) {media_sources.append(createMediaSource(1))}
    if (media_center_left) {media_sources.append(createMediaSource(4))}
    if (media_center) {media_sources.append(createMediaSource(5.5))}
    if (media_center_right) {media_sources.append(createMediaSource(7))}
    if (media_far_right) {media_sources.append(createMediaSource(10))}
}

;; Helper function for setupMediaSources that creates a media source,
;; which is simply a list of culture values.
function createMediaSource(culture_value) {
    culture = []
    for (i = 0 to feature_count) {
        culture.append(culture_value)
    }
    return (culture)
}

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;; Main simulation loop

function simulate() {
    ; Run whatever interactions are enabled
    if (use_neighbor_interactions) {

simulateNeighborInteractions()
}
if ( use_nearby_interactions ) {
    simulateNearbyInteractions()
}

; Media and meeting interactions happen once every x number of ticks
if ( use_media_sources and currentTick() mod media_interaction_interval == 0 ) {
    simulateMediaInterations()
}
if ( use_meetings and currentTick() mod meeting_interaction_interval == 0 ) {
    simulateCityMeeting()
}

calculateAgentStats()
}

;;;;;;;;; ;;;;;;;;;
;;;;;;;;; Agent interactions ;;;;;;;;
;;;;;;;;; ;;;;;;;;;;

; For some number of agents, choose a random agent and
; have it interact with a random neighboring agent.
function simulateNeighborInteractions() {
    interaction_count = numberOfAgents() * neighbor_interaction_proportion
    for (i = 0 to interaction_count ) {
        agent = chooseRandomAgent()
        neighbor = chooseRandomNeighborOf(agent)
        interactWithCulture(agent, neighbor.culture)
    }
}

; For some number of agents, choose a random agent and have it
; interact with a random nearby (not necessarily touching) agent.
function simulateNearbyInteractions() {
    interaction_count = numberOfAgents() * nearby_interaction_proportion
    for (i = 0 to interaction_count ) {
        agent = chooseRandomAgent()
        neighbor = chooseRandomNearbyOf(agent, max_nearby_distance)
        interactWithCulture(agent, neighbor.culture)
    }
}

; Choose some number of agents and have them interact with a culture
; defined in the global media_cultures list. The agents will interact
; with whichever culture is nearest their own.
function simulateMediaInterations {
    interaction_count = numberOfAgents() * media_interaction_proportion
    if ( media_interaction == 'nearest' ) {
        ; Loop through all media cultures to find the one most similar
        ; to the culture of the agent, then interact with it.
        for (i = 0 to interaction_count ) {
            agent = chooseRandomAgent()
            highest_similarity = 0
            most_similar_media = []
            for (j = 0 to media_cultures.length ) {
                similarity = calculateCultureSimilarity(agent.culture, media_cultures[j])
                if ( similarity > highest_similarity ) {
                    highest_similarity = similarity
                    most_similar_media = media_cultures[j]
                }
            }
            interactWithCulture(agent, most_similar_media)
        }
    }
}
for (j = 0 to media_sources.length) {
    similarity = calculateSimilarity(agent.culture, media_sources[i])
    if (similarity >= highest_similarity) {
        highest_similarity = similarity
        most_similar_media = media_sources[i]
    }
}

interactWithCulture(agent, most_similar_media)
}
else {
    ; Loop through all media cultures and probabilistically choose
    ; one based on similarity to the agent, then interact with it.
    for (i = 0 to interaction_count) {
        agent = chooseRandomAgent()
        total_media_distance = 0
        media_distance = []
        for (i = 0 to media_sources.length) {
            distance = 1 - (calculateSimilarity(agent.culture, media_sources[i]))
            total_media_distance = total_media_distance + distance
            media_distance[i] = distance
        }
        media_prob_bound = []
        media_total_prob = 0
        for (i = 0 to media_sources.length) {
            distance = media_distance[i]
            prob = 1 - (distance / total_media_distance)
            media_total_prob = media_total_prob + prob
            media_prob_bound.append(media_total_prob)
        }
        selected_prob = uniformRandom(min = 0, max = 1)
        done = false
        for (i = 0 to media_sources.length) {
            if (selected_prob <= media_prob_bound[i] and !done) {
                interactWithCulture(agent, media_sources[i])
                done = true
            }
        }
    }
}

; Choose a random location in the world and hold a city meeting,
; in which all nearby agents interact with each other. If we are
; using 'average' mode, compute the average culture of the entire
; meeting and have each meeting participant interact with it. Otherwise,
; have each participant interact with every other participant.
function simulateCityMeeting() {
    ; Choose a location and get nearby agents
    meeting_position = randomWorldPosition()
    participants = []
    for (each agent) {
        if (distance(agent.position, meeting_position) <= meeting_radius) {
            participants.append(agent)
        }
    }
}

if (meeting_interaction == 'average') {

; Sum each cultural feature of each participant
avg_culture = []
for (i = 0 to feature_count) {
    avg_culture.append(0)
}
for (each agent in participants) {
    for (i = 0 to feature_count) {
        avg_culture[i] = avg_culture[i] + agent.culture[i]
    }
}

; Compute the average of each cultural feature
for (i = 0 to feature_count) {
    avg_culture[i] = avg_culture[i] / participants.length
}

; Have every participant interact with the average culture
for (each agent in participants) {
    interactWithCulture(agent, avg_culture)
}
} else {
; Have every participant interact with every other participant.
for (each agent agent_0 in participants) {
    for (each agent agent_1 in participants) {
        if (agent_0 != agent_1) {
            interactWithCulture(agent_0, agent_1.culture)
        }
    }
}
}

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; Utilities ;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

; Have an agent interact with some another culture if they
; are similar enough that the agent accepts the influence
; and if the event passes some degree of randomness. In
; 'jump' mode we take the exact culture value of the other
; culture, in 'shift' mode we shift closer to that culture.
function interactWithCulture(agent, other_culture) {
    similarity = calculateSimilarity(agent.culture, other_culture)
    chance = uniformRandom(min = 0, max = 1)
    if (similarity > minimum_similarity and chance < similarity) {
        index = findDifferingElement(agent.culture, other_culture)
        if (interaction_method == 'jump') {
            agent.culture[index] = other_culture[index]
        } else {
            difference = agent.culture[index] - other_culture[index]
            agent.culture[index] = agent.culture[index] - difference * shift_degree
        }
    }
}

; Calculate whichever similarity measure is enabled for the model.
; Return a value of 0 (dissimilar) - 1 (max similarity).
function calculateSimilarity(list_a, list_b) {

JASSS, 19(2):1, 2016
http://jasss.soc.surrey.ac.uk/19/2/1.html
doi: 10.18564/jasss.3021
if (similarity_method == 'axelrod') {
    return (calculateAxelrodSimilarity(list_a, list_b))
} else {
    return (calculateRelativeSimilarity(list_a, list_b))
}

// Calculate the Axelrod method of determining cultural similarity.
// Return a degree 0 to 1 based on the number of list elements that
// are equivalent divided by the total number of elements.
function calculateAxelrodSimilarity(list_a, list_b) {
    similarity_count = 0
    for (i = 0 to list_a.length) {
        if (list_a[i] == list_b[i]) {
            similarity_count = similarity_count + 1
        }
    }
    return (similarity_count / list_a.length)
}

// Return a degree of similarity 0 to 1 between two lists based on the
// sum of differences of each list element, divided by the maximum
// possible difference between two culture lists.
function calculateRelativeSimilarity(list_a, list_b) {
    differences = 0
    for (i = 0 to list_a.length) {
        differences = differences + absoluteValue(list_a[i] - list_b[i])
    }
    max_differences = list_a.length * trait_count
    return (1 - (differences / max_differences))
}

// Identify the differing elements of two lists, then randomly select
// one of those differing elements and return its 0-based index.
function findDifferingElement(list_a, list_b) {
    differing_indexes = []
    for (i = 0 to list_a.length) {
        if (list_a[i] != list_b[i]) {
            differing_indexes.append(i)
        }
    }
    index = uniformRandom(min = 0, max = differing_indexes.length)
    return differing_indexes[index]
}

// Create a culture value derived from a normal distribution,
// limited to allowable culture values (1 to 10).
function createCultureValue() {
    culture_value = normalRandom(mean = 5.5, std_dev = 1.5)
    culture_value = limit(culture_value, min = 1, max = 10)
    return (culture_value)
}
; Compute various statistics based on agent average culture.
function calculateAgentStats() {
    for (each agent) {
        agent.avg_culture = average(agent.culture)
    }

    agent_cultures = createCultureHistogram()
    agent_std_dev = standardDeviation(agent_cultures)
    agent_avg = average(agent_cultures)
    coefficient_of_variation = agent_std_dev / agent_avg
}

; Create and return an array representing a histogram of the
; average culture of each agent.
function createCultureHistogram() {
    histogram = []
    for (each agent) {
        histogram.append(agent.avg_culture)
    }

    return (histogram)
}