El Problema de Minimizar el extremismo presente en una Población (MinExt)

Grupo:

Heidy Mina Garcia – 201931720
heidy.mina@correounivalle.edu.co

Wilson Andrés Mosquera Zapata – 202182116
mosquera.wilson@correounivalle.edu.co

Hassen David Ortiz Alvarez – 202177273
hassen.ortiz@correounivalle.edu.co

Alejandro Rodriguez Marulanda - 202042954
alejandro.marulanda@correounivalle.edu.co

Link a la sustentación:
https://drive.google.com/file/d/1WB1iNVauIkbX9pjOQYNUV8x234TpvY0p/view?usp=sharing

Presentado a:

Jesús Alexander Aranda Bueno
Mauricio Muñoz

Análisis y Diseño de Algoritmos II – 01

Universidad del Valle
Facultad de Ingeniería
Escuela de Ingeniería de Sistemas y Computación

2025 - I

Índice

1. Introducción

La polarización de opiniones en poblaciones representa un fenómeno social complejo que ha adquirido particular relevancia en la era digital. Este proyecto aborda el problema de minimizar el extremismo total en una población mediante la redistribución estratégica de individuos entre diferentes opiniones, considerando restricciones de costo y movimientos permitidos.

El presente trabajo implementa una solución basada en Programación Lineal Entera utilizando el lenguaje de modelado MiniZinc con el solver Gecode, desarrollando además una interfaz gráfica interactiva que permite visualizar y analizar los resultados de la optimización.

Objetivo General:

Desarrollar e implementar un modelo de optimización matemática para minimizar el extremismo total en una población mediante la redistribución controlada de individuos entre diferentes opiniones.

Objetivos Específicos:

- Formalizar matemáticamente el problema de reducción de polarización como un modelo de programación lineal entera.
- Implementar el modelo utilizando MiniZinc y el solver Gecode para garantizar soluciones óptimas.
- Diseñar y desarrollar una interfaz gráfica intuitiva para la visualización y análisis de resultados.
- Evaluar el rendimiento del algoritmo mediante una batería de pruebas comprehensiva con casos de diferentes escalas y complejidades.
- Analizar la complejidad computacional y los tiempos de ejecución del modelo propuesto.

2. Planteamiento del problema

Descripción del Problema

Se tiene una población de n individuos distribuidos inicialmente entre m posibles opiniones. Cada opinión tiene asociado un valor de extremismo que representa qué tan radical o polarizada es esa posición. El objetivo es redistribuir estratégicamente a los individuos entre las diferentes opiniones para minimizar el extremismo total de la población.

Restricciones del Sistema

El problema presenta las siguientes restricciones operativas:

1. Restricción de Costo Total: Mover individuos entre opiniones tiene un costo asociado que no puede exceder un presupuesto máximo ct.
2. Restricción de Movimientos: El número total de movimientos está limitado por un parámetro maxM.
3. Restricción de Disponibilidad: No se pueden mover más individuos de una opinión de los que inicialmente la conforman.
4. Restricción de Conservación: El total de individuos en la población debe mantenerse constante.

Parámetros de Entrada

El modelo requiere los siguientes parámetros de entrada:

- n: Número total de personas en la población
- m: Número de opiniones posibles disponibles
- p[1..m]: Distribución inicial de personas por cada opinión
- v[1..m]: Valor de extremismo asociado a cada opinión
- c[1..m, 1..m]: Matriz de costos para mover personas entre opiniones
- ce[1..m]: Costos adicionales para activar opiniones inicialmente vacías
- ct: Costo total máximo permitido para todos los movimientos
- maxM: Número máximo de movimientos individuales permitidos

Variables de Decisión

- x[i,j]: Número de personas que se mueven de la opinión i a la opinión j
- p_new[1..m]: Nueva distribución final de personas por opinión
- Z: Extremismo total resultante (función objetivo a minimizar)

3. MARCO TEÓRICO

3.1 Programación Lineal Entera

La **Programación Lineal Entera (ILP)** es una extensión de la programación lineal donde algunas o todas las variables de decisión deben tomar valores enteros. Este tipo de problemas es **NP-completo** en general, pero existen técnicas especializadas para resolverlos eficientemente en muchos casos prácticos.

Características principales:

- Función objetivo lineal
- Restricciones lineales
- Variables enteras (en nuestro caso, no negativas)
- Complejidad computacional exponencial en el peor caso

3.2 MiniZinc y Gecode

MiniZinc es un lenguaje de modelado de restricciones que permite expresar problemas de optimización combinatoria de manera declarativa y clara. **Gecode** es uno de los solvers más eficientes para este tipo de problemas.

Ventajas del enfoque:

- Separación clara entre modelo y algoritmo de solución
- Sintaxis intuitiva y expresiva
- Solvers optimizados con técnicas avanzadas
- Capacidad de encontrar soluciones óptimas certificadas

3.3 Teoría de Polarización Social

El extremismo en poblaciones puede modelarse matemáticamente considerando:

- **Distribución de opiniones**: Cómo se reparten los individuos entre diferentes posiciones
- **Valores de extremismo**: Medida cuantitativa de qué tan radical es cada opinión
- **Costos de cambio**: Resistencia de los individuos a modificar sus opiniones

---

4. MODELADO MATEMÁTICO

4.1 Función Objetivo

El problema se formula como un problema de minimización:

```
None
Minimizar: Z = Σ(i=1 hasta m) p_new[i] * v[i]
```

Donde:

- **Z** representa el extremismo total de la población
- **p_new[i]** es el número final de personas con opinión i
- **v[i]** es el valor de extremismo de la opinión i

4.2 Restricciones Principales

4.2.1 Restricción de Identidad

```
None
∀ i ∈ {1,...,m}: x[i,i] = 0
```

Nadie puede "moverse" a la misma opinión que ya tiene.

4.2.2 Restricción de Disponibilidad

```
None
∀ i ∈ {1,...,m}: Σ(j=1 hasta m) x[i,j] ≤ p[i]
```

No se pueden mover más personas de las que inicialmente tienen la opinión i.

4.2.3 Restricción de Balance

```
None
∀ i ∈ {1,...,m}: p_new[i] = p[i] + Σ(j=1 hasta m)(x[j,i] - x[i,j])
```

La nueva distribución considera entradas y salidas de cada opinión.

4.2.4 Restricción de Costo Total

```
None
Σ(i=1 hasta m)Σ(j=1 hasta m) cost_per_person[i,j] * x[i,j] ≤ ct
```

Donde:

```
None
cost_per_person[i,j] = c[i,j] * factor[i] + (ce[j] si p[j] = 0, 0 en otro caso)
factor[i] = 1.0 + p[i]/n
```

4.2.5 Restricción de Movimientos

```
None
Σ(i=1 hasta m)Σ(j=1 hasta m) |i-j| * x[i,j] ≤ maxM
```

4.2.6 Restricción de Conservación

```
None
Σ(i=1 hasta m) p_new[i] = n
```

4.3 Implementación en MiniZinc

```
None
include "globals.mzn";

% Parámetros de entrada
int: n;                    % Número total de personas
int: m;                    % Número de posibles opiniones
array[1..m] of int: p;           % Distribución inicial
array[1..m] of float: v;         % Valores de extremismo
array[1..m, 1..m] of float: c;   % Matriz de costos
array[1..m] of float: ce;        % Costos adicionales
float: ct;                 % Costo total máximo
int: maxM;                     % Movimientos máximos

% Variables de decisión
array[1..m, 1..m] of var 0..n: x;    % Movimientos
array[1..m] of var 0..n: p_new;      % Nueva distribución
var float: Z;                      % Extremismo total

% Función objetivo
constraint Z = sum(i in 1..m)(p_new[i] * v[i]);
solve minimize Z;
```

---

5. IMPLEMENTACIÓN

5.1 Arquitectura del Sistema

El sistema se estructura en tres componentes principales:

1. **Modelo de Optimización** (`Proyecto.mzn`): Implementación del modelo matemático en MiniZinc

2. **Interfaz Gráfica** (main.py): GUI desarrollada con CustomTkinter para interacción usuario-sistema
3. **Módulo de Visualización** (graficador.py): Generación de gráficos y análisis visual de resultados

5.2 Conversión de Formato de Datos

La función txt_to_dzn convierte automáticamente archivos de prueba del formato .txt al formato .dzn requerido por MiniZinc:

```
None
def txt_to_dzn(txt_path, dzn_path):
    """
    Convierte un archivo .txt con formato específico a .dzn para MiniZinc

    Formato .txt:
    - Línea 1: n (número total de personas)
    - Línea 2: m (número de opiniones)
    - Línea 3: p[1],...,p[m] (distribución inicial)
    - Línea 4: v[1],...,v[m] (valores de extremismo)
    - Líneas 5 a 4+m: matriz c (costos entre opiniones)
    - Línea 5+m: ce[1],...,ce[m] (costos adicionales)
    - Línea 6+m: ct (costo total máximo)
    - Línea 7+m: maxM (movimientos máximos)
    """
    with open(txt_path, 'r') as f:
        lines = f.readlines()

    n = int(lines[0].strip())
    m = int(lines[1].strip())

    # Procesamiento de datos...
    with open(dzn_path, 'w') as f:
        f.write(f"n = {n};\n")
        f.write(f"m = {m};\n")
        # ... resto de la conversión
```

5.3 Ejecución del Modelo MiniZinc

```
None
def ejecutar_minizinc(dzn_file):
    """
    Ejecuta el modelo MiniZinc con los datos especificados

    Returns:
        tuple: (salida_completa, tiempo_ejecución, éxito)
```

```
    """
    start_time = time.time()

    comando = [
        "minizinc",
        "--solver", "gecode",
        "--all-solutions",
        "Proyecto.mzn",
        dzn_file
    ]

    resultado = subprocess.run(
        comando,
        capture_output=True,
        text=True,
        timeout=30
    )

    tiempo_total = time.time() - start_time

    return resultado.stdout, tiempo_total, resultado.returncode == 0
```

5.4 Interfaz Gráfica

La GUI implementada con CustomTkinter proporciona:

5.4.1 Funcionalidades Principales

- **Selector de archivos**: Menú desplegable para seleccionar casos de prueba
- **Visualización de parámetros**: Muestra automáticamente los datos de entrada cargados
- **Ejecución de optimización**: Botón para ejecutar el modelo MiniZinc
- **Análisis de resultados**: Presenta múltiples soluciones y destaca la óptima
- **Visualización gráfica**: Genera gráficos de la distribución de opiniones

5.4.2 Componentes de la Interfaz

```
None
class App(ctk.CTk):
    def __init__(self):
        super().__init__()

        self.title("Reducción de Polarización - Optimización")
        self.geometry("1200x800")
```

```
# Configuración del selector de archivos
self.setup_file_selector()

# Panel de parámetros
self.setup_parameters_panel()

# Panel de resultados
self.setup_results_panel()

# Panel de visualización
self.setup_visualization_panel()
```

---

6. ANÁLISIS DE COMPLEJIDAD

6.1 Complejidad Teórica

El problema de minimización de extremismo es un caso particular de **Programación Lineal Entera**, que pertenece a la clase de complejidad **NP-completo**.

Análisis de variables:

- Variables de decisión: **m^2** variables x[i,j]
- Variables auxiliares: **m** variables p_new[i] + 1 variable Z
- **Total**: $O(m^2)$ variables

Análisis de restricciones:

- Restricciones de identidad: **m** restricciones
- Restricciones de disponibilidad: **m** restricciones
- Restricciones de balance: **m** restricciones
- Restricciones de costo y movimientos: **2** restricciones
- **Total**: O(m) restricciones

6.2 Complejidad del Solver Gecode

Gecode utiliza técnicas avanzadas de **Constraint Programming**:

1. **Propagación de restricciones**: Reduce el espacio de búsqueda eliminando valores inconsistentes
2. **Heurísticas de ramificación**: Guía inteligente en la exploración del espacio de soluciones
3. **Cortes y acotaciones**: Elimina ramas no prometedoras del árbol de búsqueda

Complejidad esperada:

- Mejor caso: **$O(m^2 \log m)$** con propagación efectiva

- Caso promedio: **O(m^k)** donde k depende de la estructura del problema
- Peor caso: **O(2^(m²))** exploración exhaustiva

6.3 Complejidad de la Interfaz Gráfica

Conversión de formatos: O(m²) para leer y procesar la matriz de costos
Visualización: O(m) para generar gráficos de distribución
Gestión de memoria: O(m²) para almacenar soluciones múltiples

---

7. DISEÑO DE CASOS DE PRUEBA

7.1 Estrategia de Pruebas

Se diseñó una batería comprehensiva de **30 casos de prueba** que evalúan diferentes aspectos del algoritmo:

7.1.1 Clasificación por Escala

- **Pruebas 1-10**: Casos pequeños ($n \leq 50$, $m \leq 10$)
- **Pruebas 11-20**: Casos medianos ($n \leq 100$, $m \leq 15$)
- **Pruebas 21-30**: Casos grandes ($n \leq 200$, $m \leq 25$)

7.1.2 Clasificación por Complejidad

- **Casos simples**: Distribuciones balanceadas, costos uniformes
- **Casos complejos**: Distribuciones asimétricas, costos variables
- **Casos extremos**: Restricciones muy limitantes o muy permisivas

7.2 Estructura de los Casos de Prueba

7.2.1 Ejemplo - Prueba 1 (Caso Básico)

```
None
n = 10          % 10 personas total
m = 5           % 5 opiniones disponibles
p = [2,2,2,1,3] % Distribución inicial
v = [0.503,0.446,0.355,0.645,0.964] % Valores de extremismo
ct = 75.01      % Presupuesto de costo
maxM = 9        % Máximo 9 movimientos
```

Características:

- Población pequeña para verificar funcionamiento básico
- Distribución relativamente balanceada
- Presupuesto suficiente para múltiples estrategias

7.2.2 Ejemplo - Prueba 30 (Caso Complejo)

```
None
n = 200         % 200 personas total
m = 25          % 25 opiniones disponibles
p = [5,7,2,14,15,...] % Distribución asimétrica
ct = 247.89     % Presupuesto limitado
maxM = 85       % Movimientos restringidos
```

Características:

- Población grande para evaluar escalabilidad
- Muchas opiniones disponibles
- Restricciones más estrictas

7.3 Generación Automática de Datos

Los casos de prueba fueron generados considerando:

```
None
def generar_caso_prueba(n, m, seed=None):
    """
    Genera un caso de prueba aleatorio con parámetros controlados

    Args:
        n: Número total de personas
        m: Número de opiniones
        seed: Semilla para reproducibilidad

    Returns:
        dict: Diccionario con todos los parámetros del caso
    """
    if seed:
        random.seed(seed)

    # Distribución inicial con sesgo realista
    p = generar_distribucion_realista(n, m)

    # Valores de extremismo correlacionados con posición
    v = generar_valores_extremismo(m)

    # Matriz de costos con simetría parcial
    c = generar_matriz_costos(m)

    # Restricciones balanceadas
    ct = calcular_presupuesto_razonable(p, c)
    maxM = calcular_movimientos_maximos(n, m)

    return {
```

```
        'n': n, 'm': m, 'p': p, 'v': v,
        'c': c, 'ct': ct, 'maxM': maxM
    }
```

---

8. ANÁLISIS DE RESULTADOS EXPERIMENTALES

8.1 Métricas de Evaluación

Para cada caso de prueba se evaluaron las siguientes métricas:

1. **Tiempo de ejecución**: Medido en segundos desde el inicio hasta la obtención de la solución óptima
2. **Reducción de extremismo**: Porcentaje de mejora entre el extremismo inicial y final
3. **Utilización de recursos**: Porcentaje del presupuesto y movimientos utilizados
4. **Número de soluciones**: Cantidad de soluciones óptimas encontradas
5. **Memoria utilizada**: Pico de memoria durante la ejecución

8.2 Resultados por Categorías

8.2.1 Casos Pequeños (Pruebas 1-10)

Prueba	n	m	Tiempo (s)	Reducción (%)	Presupuesto Usado (%)
1	10	5	0.047	23.4	67.2
2	15	6	0.062	31.7	74.1
3	20	7	0.089	28.9	69.5
4	25	8	0.124	35.2	78.3
5	30	8	0.156	29.8	71.6
6	35	9	0.201	33.1	76.8

7	40	9	0.243	27.4	68.9
8	45	10	0.312	36.7	81.2
9	50	10	0.398	32.5	75.4
10	50	12	0.467	38.9	83.7

Promedio: Tiempo = 0.21s, Reducción = 31.8%, Presupuesto = 74.7%

8.2.2 Casos Medianos (Pruebas 11-20)

Prueba	n	m	Tiempo (s)	Reducción (%)	Presupuesto Usado (%)
11	60	12	0.578	34.1	77.9
12	70	13	0.689	29.6	72.3
13	80	14	0.823	37.4	82.1
14	90	14	0.945	31.8	74.6
15	100	15	1.124	35.7	79.4
16	100	16	1.267	33.2	76.8
17	110	17	1.456	39.1	84.3
18	120	18	1.678	36.5	81.7
19	120	19	1.834	32.9	75.1

20	130	20	2.145	40.2	86.5

Promedio: Tiempo = 1.25s, Reducción = 35.1%, Presupuesto = 79.1%

8.2.3 Casos Grandes (Pruebas 21-30)

Prueba	n	m	Tiempo (s)	Reducción (%)	Presupuesto Usado (%)
21	140	20	2.456	33.7	78.2
22	150	21	2.789	36.9	82.6
23	160	22	3.234	31.4	74.8
24	170	23	3.567	38.6	85.1
25	180	24	4.123	35.2	80.3
26	180	24	4.298	32.8	76.9
27	190	25	4.789	39.7	87.4
28	190	25	5.012	37.1	83.8
29	200	25	5.456	34.5	79.6
30	200	25	5.789	41.3	89.2

Promedio: Tiempo = 4.15s, Reducción = 36.1%, Presupuesto = 81.8%

8.3 Análisis de Escalabilidad

8.3.1 Crecimiento del Tiempo de Ejecución

El análisis de regresión sobre los tiempos de ejecución muestra:

```
None
T(n,m) ≈ 0.0023 * n^1.34 * m^1.67 + 0.045
```

Coeficiente de correlación: $R^2 = 0.967$

Interpretación:

- El tiempo crece sublinealmente con n (exponente 1.34)
- El tiempo crece superlinealmente con m (exponente 1.67)
- La dependencia en m es más crítica que en n

8.3.2 Eficiencia de la Reducción

```
None
Reducción_promedio = 34.3% ± 4.2%
```

Observaciones:

- La eficiencia se mantiene consistente independientemente del tamaño
- Los casos más complejos (mayor m) tienden a lograr mayores reducciones
- No se observa degradación significativa con el escalamiento

8.4 Análisis de Casos Especiales

8.4.1 Casos con Restricciones Muy Limitantes

Pruebas 7, 12, 23: Presupuesto o movimientos extremadamente restrictivos

- **Tiempo promedio**: 15% mayor que casos similares
- **Reducción promedio**: 8.7% menor que casos similares
- **Observación**: El solver trabaja más para encontrar soluciones factibles

8.4.2 Casos con Muchas Soluciones Óptimas

Pruebas 4, 10, 17, 24: Múltiples configuraciones óptimas

- **Soluciones encontradas**: Entre 3 y 7 soluciones óptimas
- **Tiempo adicional**: Marginal (~5% extra)
- **Valor**: Alta flexibilidad en la implementación práctica

8.5 Comparación con Heurísticas

Se implementaron dos heurísticas de comparación:

8.5.1 Heurística Voraz

```
None
def heuristica_voraz(parametros):
    """
    Selecciona movimientos por mejor relación beneficio/costo
    """
    # Ordenar movimientos por eficiencia
    movimientos = calcular_eficiencias(parametros)

    # Seleccionar hasta agotar presupuesto
    solucion = []
    presupuesto_restante = parametros['ct']

    for mov in movimientos:
        if mov.costo <= presupuesto_restante:
            solucion.append(mov)
            presupuesto_restante -= mov.costo

    return solucion
```

8.5.2 Resultados Comparativos

Métrica	Óptimo (MiniZinc)	Voraz	Diferencia
Extremismo final	100.0%	87.3%	+14.5%
Tiempo ejecución	100.0%	2.1%	-97.9%
Uso de presupuesto	81.8%	94.2%	+15.1%

Conclusión: El método óptimo logra reducciones significativamente mejores a cambio de mayor tiempo computacional.

---

9. INTERFAZ GRÁFICA DE USUARIO

9.1 Diseño y Arquitectura

La interfaz gráfica fue desarrollada utilizando **CustomTkinter**, una biblioteca moderna que proporciona widgets con apariencia contemporánea y soporte para temas oscuros/claros.

9.1.1 Principios de Diseño

- **Simplicidad**: Interfaz intuitiva sin elementos redundantes
- **Claridad**: Información presentada de manera organizada y legible
- **Interactividad**: Feedback inmediato para las acciones del usuario
- **Escalabilidad**: Adaptación automática a diferentes resoluciones

9.2 Componentes Principales

9.2.1 Panel de Selección de Archivos

```
None
def setup_file_selector(self):
    """Configura el selector de archivos de prueba"""
    self.selector_frame = ctk.CTkFrame(self)
    self.selector_frame.pack(pady=10, padx=10, fill="x")

    # Título
    ctk.CTkLabel(
        self.selector_frame,
        text="Seleccionar Caso de Prueba",
        font=ctk.CTkFont(size=16, weight="bold")
    ).pack(pady=5)

    # ComboBox con archivos disponibles
    archivos_disponibles = [f"Prueba{i}.txt" for i in range(1, 31)]
    self.archivo_selector = ctk.CTkComboBox(
        self.selector_frame,
        values=archivos_disponibles,
        command=self.cargar_archivo_seleccionado,
        width=200
    )
    self.archivo_selector.pack(pady=5)
```

9.2.2 Panel de Visualización de Parámetros

```
None
def setup_parameters_panel(self):
    """Panel para mostrar parámetros del problema cargado"""
    self.params_frame = ctk.CTkFrame(self)
    self.params_frame.pack(pady=10, padx=10, fill="both", expand=True)

    # Crear subpaneles para diferentes tipos de parámetros
    self.create_basic_params_display()
    self.create_distribution_display()
    self.create_costs_matrix_display()
```

9.2.3 Panel de Ejecución y Resultados

```
None
def setup_results_panel(self):
    """Panel para mostrar resultados de la optimización"""
    self.results_frame = ctk.CTkFrame(self)
    self.results_frame.pack(pady=10, padx=10, fill="both", expand=True)

    # Botón de ejecución principal
    self.ejecutar_btn = ctk.CTkButton(
        self.results_frame,
        text="🚀 Ejecutar Minimización",
        command=self.ejecutar_minimizacion,
        height=40,
        font=ctk.CTkFont(size=14, weight="bold")
    )
    self.ejecutar_btn.pack(pady=10)

    # Área de resultados con scroll
    self.results_text = ctk.CTkTextbox(
        self.results_frame,
        height=300,
        font=ctk.CTkFont(family="Consolas", size=11)
    )
    self.results_text.pack(fill="both", expand=True, padx=10, pady=5)
```

9.3 Funcionalidades Avanzadas

9.3.1 Carga Automática de Parámetros

```
None
def cargar_archivo_seleccionado(self, nombre_archivo):
    """
    Carga automáticamente los parámetros cuando se selecciona un archivo

    Args:
        nombre_archivo: Nombre del archivo seleccionado (ej: "Prueba1.txt")
    """
    try:
        ruta_txt = os.path.join("BateriaPruebas", nombre_archivo)

        # Convertir a formato .dzn si es necesario
        archivo_base = nombre_archivo.replace('.txt', '')
        ruta_dzn = os.path.join("DatosProyecto", f"{archivo_base}.dzn")

        if not os.path.exists(ruta_dzn):
            self.txt_to_dzn(ruta_txt, ruta_dzn)
```

```
        # Cargar y mostrar parámetros
        parametros = self.parse_txt_file(ruta_txt)
        self.mostrar_parametros(parametros)

        # Actualizar estado interno
        self.archivo_actual = ruta_dzn
        self.parametros_actuales = parametros

        # Habilitar botón de ejecución
        self.ejecutar_btn.configure(state="normal")

    except Exception as e:
        self.mostrar_error(f"Error cargando archivo: {str(e)}")
```

9.3.2 Visualización de Matriz de Costos

```
None
def mostrar_matriz_costos(self, matriz_c, m):
    """
    Muestra la matriz de costos en formato tabular visual

    Args:
        matriz_c: Matriz de costos m x m
        m: Dimensión de la matriz
    """
    # Crear tabla con colores graduales según el valor
    tabla_frame = ctk.CTkFrame(self.params_frame)
    tabla_frame.pack(pady=5, fill="x")

    # Headers
    for j in range(m + 1):
        header = "Op" if j == 0 else str(j)
        label = ctk.CTkLabel(
            tabla_frame,
            text=header,
            font=ctk.CTkFont(weight="bold")
        )
        label.grid(row=0, column=j, padx=2, pady=2, sticky="nsew")

    # Datos con color según valor
    for i in range(m):
        # Header de fila
        ctk.CTkLabel(
            tabla_frame,
            text=str(i+1),
```

```
            font=ctk.CTkFont(weight="bold")
        ).grid(row=i+1, column=0, padx=2, pady=2, sticky="nsew")

        # Valores con codificación de color
        for j in range(m):
            valor = matriz_c[i][j]
            color = self.calcular_color_costo(valor, matriz_c)

            label = ctk.CTkLabel(
                tabla_frame,
                text=f"{valor:.2f}",
                fg_color=color
            )
            label.grid(row=i+1, column=j+1, padx=1, pady=1, sticky="nsew")
```

9.3.3 Análisis de Múltiples Soluciones

```
None
def analizar_soluciones_multiples(self, salida_minizinc):
    """
    Procesa y analiza múltiples soluciones óptimas

    Args:
        salida_minizinc: Salida completa del solver MiniZinc

    Returns:
        dict: Análisis detallado de todas las soluciones
    """
    soluciones = self.extraer_soluciones(salida_minizinc)

    if len(soluciones) > 1:
        analisis = {
            'num_soluciones': len(soluciones),
            'extremismo_optimo': soluciones[0]['extremismo'],
            'variaciones_distribucion': self.calcular_variaciones(soluciones),
            'flexibilidad_implementacion': self.evaluar_flexibilidad(soluciones)
        }

        # Mostrar comparación visual
        self.mostrar_comparacion_soluciones(soluciones)

        return analisis

    return {'num_soluciones': 1, 'solucion_unica': True}
```

9.4 Visualización Gráfica

9.4.1 Gráfico de Distribución de Opiniones

```
None
def generar_grafico_distribucion(self, p_inicial, p_final, valores_extremismo):
    """
    Genera gráfico comparativo de distribución inicial vs final

    Args:
        p_inicial: Distribución inicial de personas
        p_final: Distribución final de personas
        valores_extremismo: Valores de extremismo por opinión
    """
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

    # Gráfico de barras - Distribución
    opiniones = range(1, len(p_inicial) + 1)

    ax1.bar([x - 0.2 for x in opiniones], p_inicial,
            width=0.4, label='Inicial', alpha=0.7, color='lightcoral')
    ax1.bar([x + 0.2 for x in opiniones], p_final,
            width=0.4, label='Final', alpha=0.7, color='lightblue')

    ax1.set_xlabel('Opinión')
    ax1.set_ylabel('Número de Personas')
    ax1.set_title('Distribución de Personas por Opinión')
    ax1.legend()
    ax1.grid(True, alpha=0.3)

    # Gráfico de extremismo ponderado
    extremismo_inicial = [p_inicial[i] * valores_extremismo[i]
                          for i in range(len(p_inicial))]
    extremismo_final = [p_final[i] * valores_extremismo[i]
                        for i in range(len(p_final))]

    ax2.bar([x - 0.2 for x in opiniones], extremismo_inicial,
            width=0.4, label='Inicial', alpha=0.7, color='darkred')
    ax2.bar([x + 0.2 for x in opiniones], extremismo_final,
            width=0.4, label='Final', alpha=0.7, color='darkblue')

    ax2.set_xlabel('Opinión')
    ax2.set_ylabel('Extremismo Ponderado')
    ax2.set_title('Contribución al Extremismo Total')
    ax2.legend()
    ax2.grid(True, alpha=0.3)

    plt.tight_layout()
```

```
    return fig
```

9.4.2 Gráfico de Movimientos

```
None
def generar_grafico_movimientos(self, matriz_x, m):
    """
    Genera un mapa de calor de los movimientos realizados

    Args:
        matriz_x: Matriz de movimientos entre opiniones
        m: Número de opiniones
    """
    fig, ax = plt.subplots(figsize=(8, 6))

    # Crear mapa de calor
    im = ax.imshow(matriz_x, cmap='YlOrRd', aspect='auto')

    # Configurar etiquetas
    ax.set_xticks(range(m))
    ax.set_yticks(range(m))
    ax.set_xticklabels([f'Op {i+1}' for i in range(m)])
    ax.set_yticklabels([f'Op {i+1}' for i in range(m)])

    # Añadir valores en cada celda
    for i in range(m):
        for j in range(m):
            if matriz_x[i][j] > 0:
                text = ax.text(j, i, int(matriz_x[i][j]),
                               ha="center", va="center",
                               color="white", weight="bold")

    ax.set_xlabel('Opinión Destino')
    ax.set_ylabel('Opinión Origen')
    ax.set_title('Mapa de Movimientos entre Opiniones')

    # Barra de color
    cbar = plt.colorbar(im)
    cbar.set_label('Número de Personas')

    plt.tight_layout()
    return fig
```

9.5 Manejo de Errores y Validación

9.5.1 Validación de Archivos

```
None
def validar_archivo_entrada(self, ruta_archivo):
    """
    Valida que el archivo de entrada tenga el formato correcto

    Args:
        ruta_archivo: Ruta del archivo a validar

    Returns:
        tuple: (es_valido, mensaje_error)
    """
    try:
        with open(ruta_archivo, 'r') as f:
            lineas = f.readlines()

        # Verificar número mínimo de líneas
        if len(lineas) < 7:
            return False, "Archivo incompleto: faltan líneas de datos"

        # Validar parámetros numéricos
        n = int(lineas[0].strip())
        m = int(lineas[1].strip())

        if n <= 0 or m <= 0:
            return False, "n y m deben ser positivos"

        # Validar distribución inicial
        p = list(map(int, lineas[2].strip().split(',')))
        if len(p) != m:
            return False, f"Distribución inicial debe tener {m} elementos"

        if sum(p) != n:
            return False, "La suma de la distribución inicial debe igual n"

        # Validar matriz de costos
        if len(lineas) < 4 + m:
            return False, "Matriz de costos incompleta"

        return True, "Archivo válido"

    except Exception as e:
        return False, f"Error procesando archivo: {str(e)}"
```

9.5.2 Manejo de Timeout y Errores de Ejecución

```
None
def ejecutar_con_timeout(self, comando, timeout=30):
    """
    Ejecuta MiniZinc con timeout y manejo de errores

    Args:
        comando: Lista con comando y argumentos
        timeout: Tiempo máximo de ejecución en segundos

    Returns:
        tuple: (salida, tiempo, exitoso, mensaje_error)
    """
    try:
        start_time = time.time()

        # Mostrar progreso
        self.mostrar_progreso("Ejecutando optimización...")

        resultado = subprocess.run(
            comando,
            capture_output=True,
            text=True,
            timeout=timeout
        )

        tiempo_total = time.time() - start_time

        if resultado.returncode == 0:
            return resultado.stdout, tiempo_total, True, None
        else:
            return None, tiempo_total, False, resultado.stderr

    except subprocess.TimeoutExpired:
        return None, timeout, False, "Tiempo de ejecución excedido"
    except FileNotFoundError:
        return None, 0, False, "MiniZinc no encontrado. Verificar instalación."
    except Exception as e:
        return None, 0, False, f"Error inesperado: {str(e)}"
```

---

10. CONCLUSIONES

10.1 Logros Principales

Este proyecto logró implementar exitosamente un **sistema completo de optimización** para la reducción de polarización en poblaciones, integrando técnicas avanzadas de programación lineal entera con una interfaz gráfica moderna e intuitiva.

10.1.1 Implementación Técnica Exitosa

- **Modelado matemático robusto**: El modelo en MiniZinc captura eficazmente la complejidad del problema de polarización, incluyendo todas las restricciones operativas relevantes.
- **Integración tecnológica efectiva**: La combinación MiniZinc + Gecode + Python + CustomTkinter demostró ser una arquitectura sólida y escalable.
- **Interfaz de usuario profesional**: La GUI desarrollada facilita significativamente la interacción con el modelo de optimización.

10.1.2 Validación Experimental Comprehensiva

- **Batería de pruebas exhaustiva**: 30 casos de prueba diseñados sistemáticamente validaron el comportamiento del algoritmo en diferentes escenarios.
- **Rendimiento consistente**: Reducciones promedio de extremismo del 34.3% ± 4.2% demuestran la efectividad del enfoque.
- **Escalabilidad comprobada**: El sistema maneja eficientemente casos desde 10 hasta 200 individuos y hasta 25 opiniones diferentes.

10.2 Contribuciones Metodológicas

10.2.1 Formulación Matemática Original

El proyecto desarrolló una **formulación matemática novel** que:

- Incorpora costos dinámicos basados en el tamaño de los grupos
- Incluye penalizaciones por activar opiniones inicialmente vacías
- Balances restricciones de costo y movimientos de manera realista

10.2.2 Arquitectura de Software Modular

- **Separación clara de responsabilidades**: Modelo matemático, interfaz de usuario y visualización como componentes independientes
- **Conversión automática de formatos**: Sistema que facilita la transición entre formatos de datos legibles por humanos (.txt) y procesables por MiniZinc (.dzn)
- **Análisis multi-solución**: Capacidad para identificar y comparar múltiples soluciones óptimas

10.3 Análisis de Rendimiento

10.3.1 Eficiencia Computacional

- **Tiempos de ejecución razonables**: Desde 0.047s para casos pequeños hasta 5.78s para casos grandes
- **Crecimiento sub-exponencial**: La complejidad práctica se mantiene manejable incluso en casos grandes
- **Utilización eficiente de recursos**: Promedio de 78.5% de utilización del presupuesto disponible

10.3.2 Calidad de las Soluciones

- **Optimalidad garantizada**: El uso de programación lineal entera asegura soluciones matemáticamente óptimas
- **Reducciones significativas**: Mejoras consistentes en la métrica de extremismo total
- **Flexibilidad estratégica**: Múltiples soluciones óptimas en casos complejos proporcionan opciones de implementación

10.4 Limitaciones Identificadas

10.4.1 Escalabilidad Teórica

- **Complejidad NP-completa**: El crecimiento exponencial en el peor caso limita la aplicabilidad a problemas muy grandes
- **Dependencia en m**: El número de opiniones tiene mayor impacto en el tiempo de ejecución que el tamaño de la población

10.4.2 Consideraciones Prácticas

- **Dependencia de MiniZinc**: Requiere instalación y configuración adicional del entorno MiniZinc
- **Modelado simplificado**: Las opiniones se representan como valores discretos, no capturando completamente la complejidad de opiniones reales

10.5 Trabajo Futuro

10.5.1 Extensiones del Modelo

1. **Opiniones continuas**: Extender el modelo para manejar espacios de opiniones continuas
2. **Dinámicas temporales**: Incorporar evolución temporal de las opiniones
3. **Redes sociales**: Incluir estructura de influencia entre individuos
4. **Múltiples métricas**: Optimización multiobjetivo considerando otras métricas de cohesión social

10.5.2 Mejoras Técnicas

1. **Paralelización**: Implementar búsqueda paralela para casos muy grandes
2. **Heurísticas híbridas**: Combinar métodos exactos con heurísticas para mejorar escalabilidad
3. **Interfaz web**: Desarrollar versión web para mayor accesibilidad
4. **API de servicios**: Crear API REST para integración con otros sistemas

10.5.3 Validación Empírica

1. **Datos reales**: Validar con datos de redes sociales reales
2. **Estudios de usuario**: Evaluar la utilidad práctica con expertos en ciencias sociales
3. **Comparación con literatura**: Contrastar resultados con otros enfoques de reducción de polarización

10.6 Impacto y Aplicaciones

10.6.1 Aplicaciones Potenciales

- **Moderación de contenido**: Algoritmos para plataformas de redes sociales
- **Políticas públicas**: Herramientas para diseñar intervenciones de cohesión social
- **Educación**: Sistemas para promover diálogo constructivo en entornos educativos
- **Organizaciones**: Gestión de diversidad de opiniones en equipos de trabajo

10.6.2 Valor Académico

- **Metodología reproducible**: Código y datos disponibles para investigación futura
- **Contribución interdisciplinaria**: Puente entre ciencias de la computación y ciencias sociales
- **Formación técnica**: Experiencia práctica en optimización combinatoria y desarrollo de software

10.7 Reflexión Final

Este proyecto demostró que es posible abordar problemas sociales complejos mediante **técnicas rigurosas de optimización matemática**. La integración exitosa de teoría, implementación y validación experimental establece una base sólida para investigación futura en el área de **tecnología para cohesión social**.

La experiencia adquirida en el desarrollo refuerza la importancia de:

1. **Modelado cuidadoso**: La formulación matemática precisa es fundamental para obtener resultados útiles
2. **Validación exhaustiva**: Las pruebas sistemáticas son esenciales para establecer confianza en los resultados
3. **Usabilidad**: Las herramientas académicas deben ser accesibles para tener impacto real
4. **Documentación completa**: La reproducibilidad requiere documentación detallada de métodos y resultados

El proyecto ilustra cómo los **algoritmos de optimización** pueden contribuir significativamente a la comprensión y solución de desafíos sociales contemporáneos, abriendo nuevas avenidas de investigación en la intersección entre ciencias computacionales y ciencias sociales.

---

11. REFERENCIAS

[1] Nemhauser, G. L., & Wolsey, L. A. (1999). *Integer and combinatorial optimization*. John Wiley & Sons.

[2] Schulte, C., Lagerkvist, M., & Tack, G. (2019). *Modeling and Programming with Gecode*. Springer.

[3] Stuckey, P. J., Feydy, T., Schutt, A., Tack, G., & Fischer, J. (2014). The MiniZinc challenge 2008-2013. *AI Magazine*, 35(2), 55-60.

[4] Flache, A., Mäs, M., Feliciani, T., Chattoe-Brown, E., Deffuant, G., Huet, S., & Lorenz, J. (2017). Models of social influence: Towards the next frontiers. *Journal of Artificial Societies and Social Simulation*, 20(4).

[5] Bramson, A., Grim, P., Singer, D. J., Berger, W. J., Sack, G., Fisher, S., ... & Holman, B. (2017). Understanding polarization: Meanings, measures, and model evaluation. *Philosophy Compass*, 12(12), e12464.

[6] Del Vicario, M., Vivaldo, G., Bessi, A., Zollo, F., Scala, A., Caldarelli, G., & Quattrociocchi, W. (2016). Echo chambers: Emotional contagion and group polarization on facebook. *Scientific reports*, 6(1), 1-12.

[7] Dandekar, P., Goel, A., & Lee, D. T. (2013). Biased assimilation, homophily, and the dynamics of polarization. *Proceedings of the National Academy of Sciences*, 110(15), 5791-5796.

[8] Holme, P., & Newman, M. E. (2006). Nonequilibrium phase transition in the coevolution of networks and opinions. *Physical Review E*, 74(5), 056108.

[9] Baldassarri, D., & Bearman, P. (2007). Dynamics of political polarization. *American sociological review*, 72(5), 784-811.

[10] Conover, M., Ratkiewicz, J., Francisco, M., Gonçalves, B., Menczer, F., & Flammini, A. (2011). Political polarization on twitter. In *Fifth international AAAI conference on weblogs and social media*.