Informe de Laboratorio Método Gráfico

José M. Benavides, Emmanuel M. Sierra

Ingeniería de Sistemas, Universidad de Los Llanos

Villavicencio, Colombia

jmbenavides@unillanos.edu.co

emartinez.sierra@unillanos.edu.co

Resumen – El laboratorio consistió en la resolución de un problema de programación lineal mediante el método gráfico, complementado con el uso de inteligencia artificial. Se definió una función objetivo a optimizar y se establecieron restricciones, obteniendo la región factible y la solución óptima en (50,50) con un valor de 4000. Adicionalmente, se implementó un módulo con IA que permite interpretar enunciados en lenguaje natural y transformarlos automáticamente en un modelo matemático con función objetivo y restricciones.

I.INTRODUCCION

La programación lineal es una herramienta matemática utilizada para optimizar una función objetivo sujeta a un conjunto de restricciones. El método gráfico es una de las formas más intuitivas para resolver problemas de programación lineal en dos variables, ya que permite visualizar la región factible y determinar la solución óptima.

En esta práctica, además de resolver manualmente un problema de optimización utilizando el método gráfico, se integró un componente de inteligencia artificial que procesa enunciados en lenguaje natural. De esta manera, el usuario puede ingresar directamente la descripción textual de un problema y la IA se encarga de extraer la función objetivo y las restricciones en formato algebraico. Esta integración convierte el sistema en una herramienta híbrida que combina el rigor matemático del método gráfico con la flexibilidad de la IA, haciendo más accesible la formulación y resolución de problemas de programación lineal.

II.REFERENTE TEORICO

La programación lineal es una técnica matemática utilizada para la optimización de recursos en problemas donde se busca maximizar o minimizar una función objetivo, sujeta a un conjunto de restricciones lineales. Se aplica en diversas áreas como economía, ingeniería, logística y administración de empresas. El método gráfico es una de las formas más intuitivas para resolver problemas de programación lineal en dos variables, ya que permite representar gráficamente las restricciones como inecuaciones en un plano cartesiano y encontrar visualmente la región factible.

Un problema de programación lineal se compone de:

- **Función objetivo**: Expresión matemática que se desea optimizar.

$$Z = ax + by$$

- **Restricciones**: Condiciones que limitan las posibles soluciones, expresadas como inecuaciones.

$$A1x + B1y \leq C1$$

$$A2x + B2y \leq C2$$

- **Condiciones de no negatividad**:

$$x \geq 0,\ y \geq 0$$

La intersección de las restricciones forma la región factible, dentro de la cual se encuentra la solución óptima. Para identificar esta solución, se evalúa la función objetivo en los vértices de la región factible y se selecciona el valor que optimiza el resultado.

En este trabajo también se incluye un componente de inteligencia artificial (IA) basado en modelos de lenguaje. La IA se encarga de procesar enunciados escritos en lenguaje natural y transformarlos en los elementos matemáticos de un problema de programación lineal: tipo de problema, función objetivo y restricciones. Esto se logra mediante el uso de un modelo de lenguaje de OpenAI, al cual se le envía el

enunciado y retorna un JSON estructurado que puede ser interpretado automáticamente por el programa.

III. PROCEDIMIENTO

A. Etapa De Revisión

Para resolver un problema de programación lineal mediante el método gráfico, primero se define la función objetivo que se desea optimizar, ya sea maximizar o minimizar. A continuación, se establecen las restricciones del problema en forma de inecuaciones, las cuales representan limitaciones o condiciones a cumplir en la solución del problema. Luego, se procede a representar gráficamente estas restricciones en un plano cartesiano. Cada inecuación define una región en el espacio bidimensional, y la intersección de todas estas regiones conforma la región factible, es decir, el conjunto de soluciones que cumplen todas las restricciones simultáneamente.

Una vez identificada la región factible, se determinan sus vértices resolviendo los sistemas de ecuaciones que surgen al igualar las restricciones. Finalmente, se evalúa la función objetivo en cada uno de estos vértices y se selecciona aquel punto que optimiza el valor buscado, ya sea el máximo o el mínimo.

En este laboratorio, el procedimiento se implementó en Python a través de una interfaz gráfica que ofrece dos modos de uso principales. En el modo manual, el usuario selecciona si desea maximizar o minimizar, ingresa la función objetivo y las dos restricciones en formato algebraico. A partir de esta información, el programa calcula los puntos de intersección de las restricciones, construye la región factible, evalúa los vértices y determina la solución óptima. Finalmente, se muestra la gráfica con la región factible sombreada y el punto óptimo identificado.

Por otro lado, el modo texto con IA permite una interacción más natural con el sistema. En este caso, el usuario introduce el enunciado del problema en lenguaje cotidiano, por ejemplo: “una fábrica produce camisas y pantalones con ciertas limitaciones de recursos”. Este enunciado se envía a un modelo de inteligencia artificial, que devuelve un JSON estructurado con la información clave del problema: el tipo de optimización, la función objetivo expresada en términos de las variables “x” y “y”, y las dos restricciones correspondientes. El programa interpreta este JSON, llena automáticamente los campos de la interfaz y ejecuta el mismo procedimiento gráfico para mostrar la región factible y la solución óptima.

B. Etapa Final

Se capturaron las respuestas arrojadas por la interfaz gráfica de usuario y se compararon esos resultados con los resultados obtenidos al resolverlo a mano.

C. Figuras.

```python
def intentar_extraer_json_de_texto(texto):
    if not texto or not isinstance(texto, str):
        return None
    possible_indexes = [m.start() for m in re.finditer(r"\{", texto)]
    if not possible_indexes:
        return None
    for start in possible_indexes:
        for end_match in reversed(list(re.finditer(r"\}", texto))):
            end = end_match.end()
            sub = texto[start:end]
            try:
                data = json.loads(sub)
                return data
            except Exception:
                continue
    try:
        return json.loads(texto)
    except Exception:
        return None
```

Figura 1. Código para extraer json del texto.

```python
def _graficar(vertices, r1, r2, punto_optimo, canvas_widget, min_x=0.0, min_y=0.0):
    verts = _ordenar_poligono(vertices)
    x_v, y_v = zip(*verts)
    plt.clf()
    plt.fill(x_v, y_v, alpha=0.2, label="Región factible")
    def dibujar_recta(A, B, C, etiqueta):
        xs_max = max(max(x_v + (punto_optimo[0],) if punto_optimo else x_v), 1) * 1.2
        xs = np.linspace(min_x, xs_max, 400)
        ys = []
        for x in xs:
            if abs(B) < 1e-12:
                ys.append(np.nan)
            else:
                ys.append((C - A * x) / B)
        plt.plot(xs, ys, label=etiqueta)
    dibujar_recta(*r1, etiqueta="Restricción 1")
    dibujar_recta(*r2, etiqueta="Restricción 2")
    # Líneas de mínimos visuales
    if min_x > 0:
        plt.axvline(min_x, linestyle=":", linewidth=0.8)
    if min_y > 0:
        plt.axhline(min_y, linestyle=":", linewidth=0.8)

    if punto_optimo:
        plt.scatter([punto_optimo[0]], [punto_optimo[1]], s=60, marker="o", label="Óptimo")
    maxx = max([p[0] for p in vertices] + ([punto_optimo[0]] if punto_optimo else [0])) * 1.15 + 1e-9
    maxy = max([p[1] for p in vertices] + ([punto_optimo[1]] if punto_optimo else [0])) * 1.15 + 1e-9
    maxx = max(maxx, min_x + 1)
    maxy = max(maxy, min_y + 1)
    plt.xlim(min_x, maxx)
    plt.ylim(min_y, maxy)
    plt.axhline(0, linewidth=0.7, color="black")
    plt.axvline(0, linewidth=0.7, color="black")
    plt.grid(True, linestyle="--", linewidth=0.5)
    plt.xlabel("x")
    plt.ylabel("y")
```

Figura 2. Fragmento código para graficar los datos ingresados.

```python
def resolver_y_graficar(obj_str, tipo_str, r1_str, r2_str, canvas_widget, resultado_var, min_x=0.0, min_y=0.0):
    try:
        a, b = _parse_objetivo(obj_str)
        A1, B1, C1 = _parse_restriccion(r1_str)
        A2, B2, C2 = _parse_restriccion(r2_str)
    except Exception as e:
        messagebox.showerror("Error de formato", str(e))
        return

    # Candidatos de intersección
    candidatos = set()

    # Siempre incluir el punto de mínimos
    candidatos.add((float(min_x), float(min_y)))

    def inter_con_xmin(A, B, C, xmin):
        if abs(B) < 1e-12:
            return []
        y = (C - A * xmin) / B
        return [(xmin, y)]

    def inter_con_ymin(A, B, C, ymin):
        if abs(A) < 1e-12:
            return []
        x = (C - B * ymin) / A
        return [(x, ymin)]

    # Intersecciones con las rectas x=min_x e y=min_y
    for A, B, C in [(A1, B1, C1), (A2, B2, C2)]:
        for p in inter_con_xmin(A, B, C, min_x):
            candidatos.add((float(p[0]), float(p[1])))
        for p in inter_con_ymin(A, B, C, min_y):
            candidatos.add((float(p[0]), float(p[1])))
```

Figura 3. Fragmento de código para resolver los datos ingresados.

```python
def call_openai_raw(prompt):
    if not API_KEY or API_KEY.strip() == "":
        raise RuntimeError("Falta OPENAI_API_KEY en el entorno. Configura tu API key en variables de entorno.")
    headers = {
        "Content-Type": "application/json",
        "Authorization": f"Bearer {API_KEY}",
    }
    payload = {
        "model": MODEL,
        "input": prompt,
    }
    r = requests.post(OPENAI_URL, headers=headers, json=payload, timeout=30)
    return r
```

Figura 4. Código para enviar el promp a OpenAI

```python
def _enviar_a_ia(self):
    texto = self.texto_widget.get("1.0", tk.END).strip()
    if not texto:
        messagebox.showerror("Error", "Pega primero el enunciado.")
        return

    # Prompt robusto: fuerza a usar x e y y a construir restricciones por consumo de recursos
    prompt = f"""
        Eres un asistente experto en programación lineal de 2 variables.
        Lee el enunciado y devuelve ÚNICAMENTE un JSON con estas claves exactas:
        - tipo: "Maximizar" o "Minimizar"
        - objetivo: usa variables x e y (ej: "Z = 5x + 3y")
        - restricciones: lista con EXACTAMENTE 2 strings en forma algebraica (ej: "2x + y <= 8").

        Reglas:
        - Usa SIEMPRE variables x e y.
        - Si dos lotes consumen un mismo recurso (p. ej., ambos llevan 1 pantalón), la restricción de ese recurso es la SUMA de
        - No agregues texto adicional ni explicaciones. Devuelve SOLO JSON válido.

        Enunciado:
        {texto}
        """
    try:
        resp = call_openai_raw(prompt)
    except Exception as e:
        messagebox.showerror("Error API", f"No se pudo conectar con OpenAI:\n{e}")
        return

    try:
        # Tomar mínimos actuales del UI para respetarlos al resolver
        min_x = float(self.min_x_entry.get() or 0)
        min_y = float(self.min_y_entry.get() or 0)
```

Figura 5. Fragmento de código del uso de la IA en la resolución de problemas ingresados.

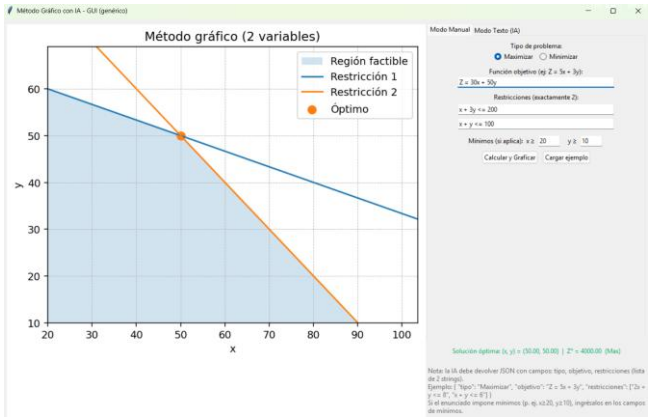

Figura 6. Respuesta del programa ingresando la función.

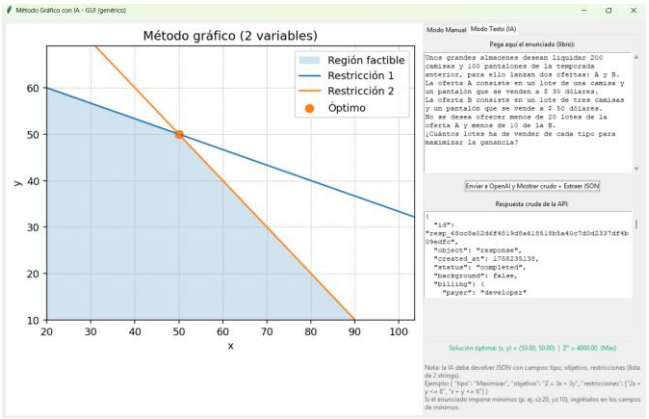

Figura 7. Respuesta del programa al ingresar el enunciado del problema.

IV.CONCLUSIÓN

El método gráfico permitió visualizar de manera clara la región factible y la solución óptima en problemas de programación lineal con dos variables. En el desarrollo del laboratorio, se comprobó que este enfoque sigue siendo una herramienta pedagógica valiosa para comprender los fundamentos de la optimización lineal.

En este caso, se encontró que la mejor solución es en el punto (50.0, 50.0), con un valor óptimo de 4000 siendo la misma encontrada al hacerlo a mano.

La integración de inteligencia artificial añadió un valor significativo al sistema, al permitir que un enunciado en lenguaje natural sea transformado en un modelo matemático con su respectiva función objetivo y restricciones.

En conclusión, el trabajo realizado no solo permitió aplicar el método gráfico tradicional, sino también explorar el potencial de la IA como apoyo en la enseñanza y aplicación de la programación lineal.

REFERENCIAS

[1] Winston, W. L. (2004). Operations Research: Applications and Algorithms. Cengage Learning.
[2] Hillier, F. S., & Lieberman, G. J. (2010). Introduction to Operations Research. McGraw-Hill Education.
[3] Taha, H. A. (2017). *Operations Research: An Introduction* (10th ed.). Pearson.