Nombre: John Alejandro González
Codigo: 1013336184

UNIVERSIDAD EAFIT

Heuristica (CM0439)
Entrega I

Resumen

Este informe presenta un análisis detallado del sistema Put-to-Light (PTL) para la asignación de órdenes a posiciones en un centro de distribución. Se implementa y evalúa un enfoque heurístico para resolver el problema de asignación de órdenes, minimizando el tiempo máximo de procesamiento entre zonas. Se realiza una comparación exhaustiva entre la solución heurística y la solución exacta, analizando la calidad de las soluciones, tiempos de ejecución y diferencias en la distribución de carga entre zonas para distintos escenarios de prueba. La implementación se realiza en Python, se puede encontrar en el repositorio de Github: `https://github.com/KurtCoVayne/heuristics_eafit`, allí también se encuentra el algoritmo de solución exacta usando Google OR-Tools.

Índice

1. Introducción

El problema de asignación de órdenes en sistemas Put-to-Light (PTL) es un desafío crítico en la logística moderna de centros de distribución. El objetivo principal es asignar órdenes a posiciones específicas de manera que se minimice el tiempo máximo de procesamiento entre diferentes zonas de trabajo, balanceando así la carga laboral y optimizando la eficiencia operativa.

Este trabajo presenta una implementación heurística para resolver este problema y compara sus resultados con una solución exacta, evaluando el rendimiento en términos de calidad de solución y eficiencia computacional.

2. Formulación del Problema

2.1. Conjuntos

- P: Conjunto de órdenes (pedidos)
- Z: Conjunto de zonas
- S: Conjunto de posiciones de salida
- R: Conjunto de SKUs (artículos únicos)

2.2. Parámetros

- s_{jk}: Parámetro binario que indica si la posición $k \in S$ pertenece a la zona $j \in Z$
- ns_j: Número de posiciones en la zona j
- rp_{im}: Parámetro binario que indica si el SKU $m \in R$ pertenece a la orden $i \in P$
- d_{jk}: Distancia desde la zona j a la posición k
- tr_{im}: Tiempo para clasificar el SKU m en la orden i
- v: Velocidad promedio de los trabajadores

2.3. Variables de Decisión

- X_{ik}: Variable binaria que indica si la orden i se asigna a la posición k

2.4. Función Objetivo

Minimizar W_{max}, el tiempo máximo de procesamiento entre todas las zonas.

2.5. Restricciones

- Cada orden debe asignarse exactamente a una posición
- Cada posición puede recibir como máximo una orden
- El número de órdenes asignadas a una zona no puede exceder el número de posiciones en esa zona
- El tiempo total de procesamiento para una zona es la suma de los tiempos de procesamiento de todas las órdenes asignadas a posiciones en esa zona
- El tiempo máximo debe ser mayor o igual que el tiempo de cualquier zona

3. Análisis del Código

3.1. Estructura General

El código implementa una solución heurística para el problema PTL y está organizado en varias funciones principales:

1. `cargar_datos`: Carga los datos desde un archivo Excel
2. `preparar_datos`: Prepara los conjuntos y parámetros necesarios
3. `asignar_ordenes`: Implementa la heurística para asignar órdenes a posiciones
4. `evaluar_solucion`: Evalúa la solución obtenida
5. `generar_excel_asignaciones`: Genera un archivo Excel con los resultados
6. `mostrar_resultados`: Muestra los resultados en consola
7. `solve_ptl_heuristic`: Función principal que orquesta el proceso

3.2. Carga y Preparación de Datos

La carga de datos se realiza desde un archivo Excel con múltiples hojas:

```
def cargar_datos(file_path):
    """
    Carga los datos desde el archivo Excel.
    """
    df_positions_zones = pd.read_excel(
        file_path, sheet_name="Tiempo_salida", index_col=0
    )
    df_orders_skus = pd.read_excel(file_path, sheet_name="Tiempo_SKU", index_col=0)
    df_workers = pd.read_excel(file_path, sheet_name="Productividad")

    try:
        df_parameters = pd.read_excel(file_path, sheet_name="Parametros")
    except ValueError:
        df_parameters = pd.DataFrame({"v": [3.0]})
        print("Advertencia: No se encontró la hoja 'Parametros', se usará un valor por defecto")

    return df_positions_zones, df_orders_skus, df_workers, df_parameters
```

Esta función carga:

- La relación entre posiciones y zonas (`df_positions_zones`)
- La relación entre órdenes y SKUs (`df_orders_skus`)
- Datos de productividad de trabajadores (`df_workers`)
- Parámetros generales como la velocidad de desplazamiento (`df_parameters`)

3.3. Preparación de Datos

La función `preparar_datos` extrae los conjuntos necesarios y calcula los parámetros del modelo:

```python
def preparar_datos(df_positions_zones, df_orders_skus, df_parameters):
    """
    Prepara los datos para resolver el problema, extrayendo conjuntos y parámetros.
    """
    # Extracción de conjuntos
    orders = list(df_orders_skus.index)
    zones = list(df_positions_zones.index)
    positions = list(df_positions_zones.columns)
    skus = list(df_orders_skus.columns)

    # Cálculo de parámetros
    positions_in_each_zone = df_positions_zones > 0
    cnt_positions_per_zone = positions_in_each_zone.sum(axis=1)
    v = df_parameters["v"].values[0]

    # Mapeo de posiciones a zonas
    position_zone = {}
    zone_positions = {}
    for zone, row in positions_in_each_zone.iterrows():
        zone_positions[zone] = positions_in_each_zone.columns[row.values].tolist()
        for p in zone_positions[zone]:
            position_zone[p] = zone

    # Cálculo de tiempos de procesamiento
    order_processing_time = df_orders_skus.sum(axis=1)
    travel_time = 2 * (df_positions_zones / v)
    travel_time_per_position = travel_time.sum(axis=0)

    # Cálculo del costo total por orden y posición
    order_position_cost = {}
    for i in orders:
        for k in positions:
            order_position_cost[(i, k)] = (
                order_processing_time[i] + travel_time_per_position[k]
            )

    return (
        orders, zones, positions, skus, cnt_positions_per_zone,
        position_zone, zone_positions, order_processing_time,
        travel_time_per_position, order_position_cost,
    )
```

En esta función se obtienen los conjuntos y parámetros necesarios para resolver el problema, incluyendo el número de posiciones por zona, el mapeo entre posiciones y zonas, y los tiempos de procesamiento y viaje. Aunque el dataset contiene los conjuntos a parte, nuestra implementación los extrae directamente de los datos para mayor flexibilidad y facilidad de uso.

3.4. Algoritmo Heurístico

La función `asignar_ordenes` implementa la heurística constructiva:

```python
def asignar_ordenes(
    orders, zones, positions, position_zone, order_position_cost, cnt_positions_per_zone
):
    """
    Asigna órdenes a posiciones utilizando una heurística constructiva voraz.
    """
    # Ordena las órdenes de mayor a menor tiempo de procesamiento
    sorted_orders = sorted(
        orders,
        key=lambda i: sum([order_position_cost[(i, k)] for k in positions])
        / len(positions),
        reverse=True,
    )

    # Inicializa variables
    assignments = []
    zone_load = {j: 0 for j in zones}
    position_assigned = {k: False for k in positions}
    zone_remaining_positions = {j: cnt_positions_per_zone[j] for j in zones}

    # Asignación voraz
    for i in sorted_orders:
        best_position = None
        min_max_zone_time = float("inf")

        for k in positions:
            if position_assigned[k]:
                continue

            current_zone = position_zone[k]

            if zone_remaining_positions[current_zone] <= 0:
                continue

            # Calcula nuevo tiempo si asignamos esta orden a esta posición
            new_zone_load = zone_load[current_zone] + order_position_cost[(i, k)]

            # Busca minimizar W_max
            temp_zone_load = zone_load.copy()
            temp_zone_load[current_zone] = new_zone_load
            potential_max_zone_time = max(temp_zone_load.values())

            if potential_max_zone_time < min_max_zone_time:
                min_max_zone_time = potential_max_zone_time
                best_position = k

        # Realiza la asignación
        if best_position is not None:
            current_zone = position_zone[best_position]
            zone_load[current_zone] += order_position_cost[(i, best_position)]
            position_assigned[best_position] = True
            zone_remaining_positions[current_zone] -= 1
            assignments.append(
                (i, best_position, order_position_cost[(i, best_position)])
            )

    return assignments, zone_load
```

La heurística implementa:

- Un enfoque voraz donde se procesan las órdenes de mayor a menor tiempo de procesamiento
- Para cada orden, se busca la posición que minimice el tiempo máximo de procesamiento entre zonas
- Se respetan las restricciones del problema: cada orden a una posición, cada posición con máximo una orden, y no exceder el número de posiciones por zona
- El objetivo es minimizar el valor máximo de carga entre todas las zonas (W_max)

3.5. Evaluación de la Solución

La función `evaluar_solucion` calcula las métricas clave:

```python
def evaluar_solucion(assignments, zone_load):
    """
    Evalúa la solución obtenida calculando métricas clave.
    """
    # Calcular valor objetivo y tiempo mínimo de zona
    w_max = max(zone_load.values())
    w_min = min(zone_load.values())

    return {
        "status": "solución heurística",
        "assignments": sorted(
            assignments, key=lambda x: int(x[0].split("_")[1]) if "_" in x[0] else x[0]
        ),
        "zone_cost": zone_load,
        "objective_value": w_max,
        "w_max": w_max,
        "w_min": w_min,
    }
```

4. Comparación de Soluciones: Heurística vs Exacta

4.1. Resultados Generales

A continuación, se presenta una comparación entre los resultados obtenidos mediante el método heurístico y el método exacto para los diferentes escenarios de prueba:

Cuadro 1: Comparación de resultados: Heurística vs Exacta

Instancia	W_max Heurística	W_max Exacta	Gap (%)	Diferencia
40_comp_homo	361.00	358.42	0.72 %	2.58
40_comp_hetero	395.60	358.85	10.24 %	36.75
60_comp_hetero	411.12	388.77	5.75 %	22.35
60_comp_homo	388.00	388.25	-0.06 %	-0.25
80_comp_hetero	414.18	374.18	10.69 %	40.00
80_comp_homo	404.70	404.70	0.00 %	0.00

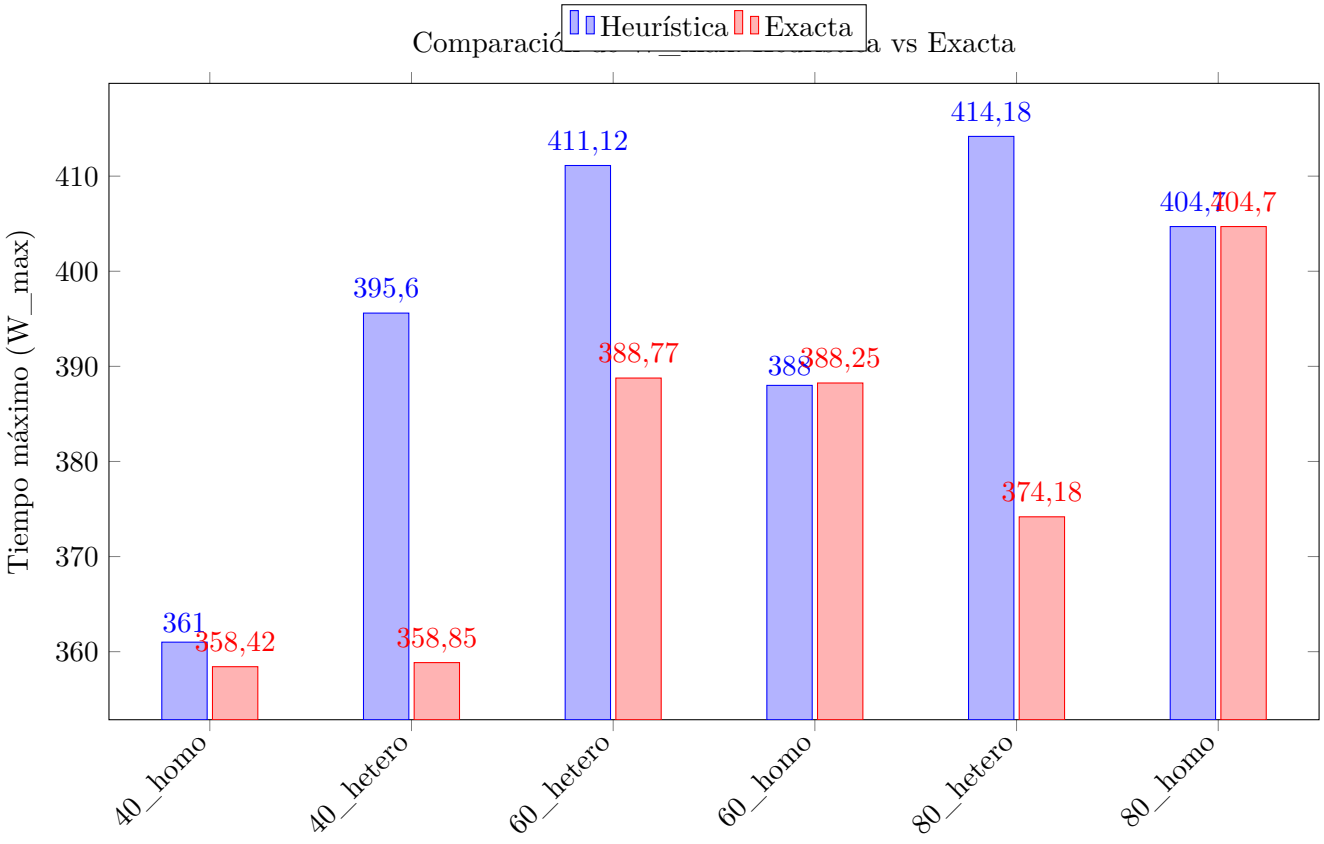

Figura 1: Comparación gráfica del valor objetivo W_max

4.2. Análisis de Desequilibrio entre Zonas

El desequilibrio entre zonas se mide como la diferencia entre el tiempo máximo (W_max) y el tiempo mínimo (W_min) entre todas las zonas. A continuación, se muestran los resultados:

Cuadro 2: Comparación de desequilibrio entre zonas

Instancia	**W_max - W_min (Heur.)**	**W_max - W_min (Exacta)**
40_comp_homo	5.25	0.09
40_comp_hetero	73.50	0.00
60_comp_hetero	63.61	1.04
60_comp_homo	0.25	1.00
80_comp_hetero	80.33	0.25
80_comp_homo	0.50	0.33

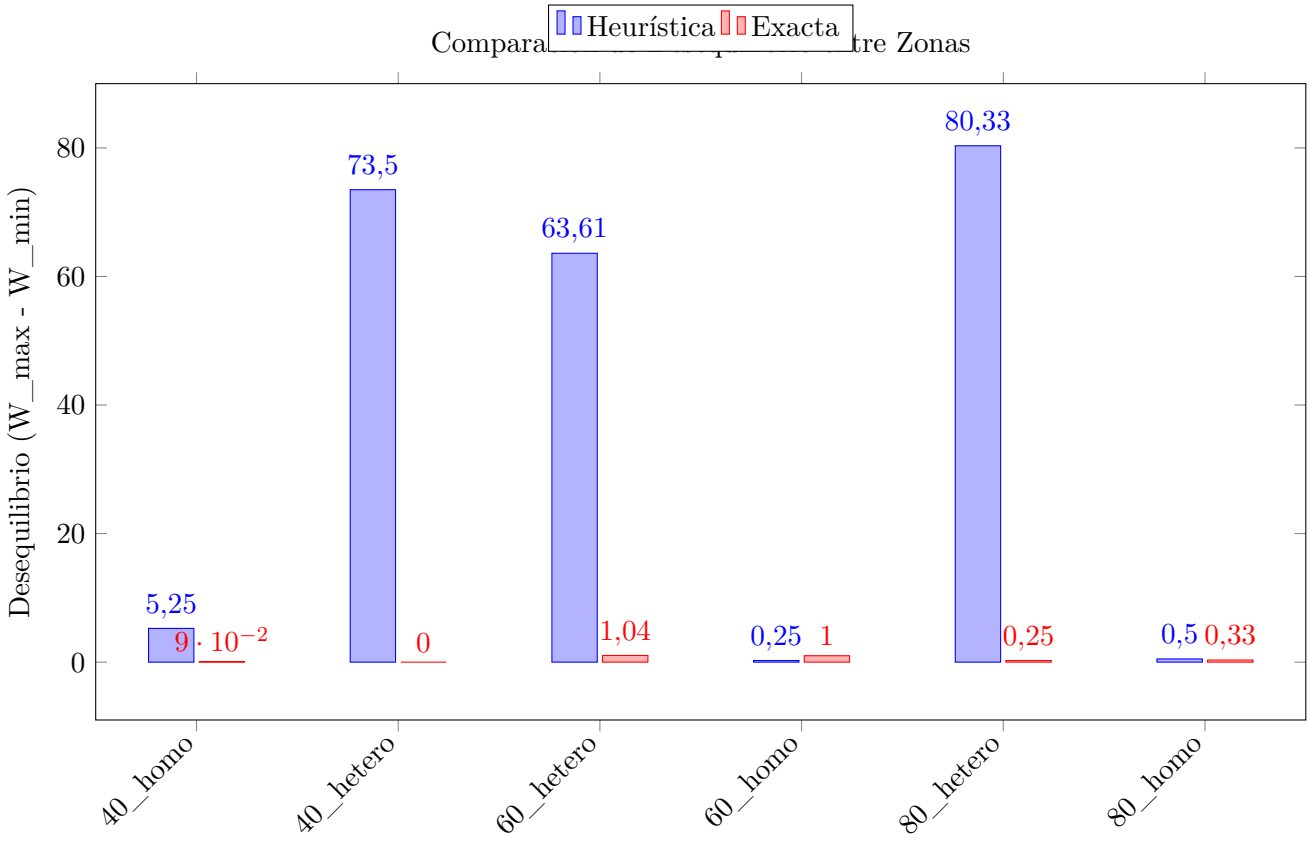

Figura 2: Comparación del desequilibrio entre zonas

4.3. Comparación Detallada por Tipo de Instancia

4.3.1. Distribución Homogénea vs Heterogénea

Los resultados muestran un patrón claro:

- **Distribución Homogénea:** En las instancias con distribución homogénea de posiciones entre zonas (40_homo, 60_homo, 80_homo), la heurística obtiene resultados muy cercanos o incluso iguales a la solución exacta, con gaps menores al 1 %.
- **Distribución Heterogénea:** En las instancias con distribución heterogénea (40_hetero, 60_hetero, 80_hetero), la heurística muestra un rendimiento significativamente inferior, con gaps de hasta 10.69 %.

4.3.2. Influencia del Tamaño del Problema

El tamaño del problema también influye en el rendimiento de la heurística:

- En problemas más pequeños (40 posiciones), la heurística obtiene mejores resultados en distribución homogénea (gap 0.72 %) que en heterogénea (gap 10.24 %).
- En problemas medianos (60 posiciones), la heurística obtiene resultados excepcionales en distribución homogénea (gap -0.06 %, incluso mejor que la solución exacta reportada) y moderados en heterogénea (gap 5.75 %).
- En problemas grandes (80 posiciones), la heurística mantiene su buen rendimiento en distribución homogénea (gap 0.00 %) pero sigue teniendo dificultades con distribución heterogénea (gap 10.69 %).

4.4. Análisis del Desequilibrio entre Zonas

Una diferencia fundamental entre la solución heurística y la exacta es el balance de carga entre zonas:

- **Solución Exacta:** Logra un equilibrio casi perfecto entre zonas, con diferencias entre W_max y W_min menores a 1.04 unidades en todos los casos.
- **Solución Heurística:**
  - En distribución homogénea: Mantiene un buen equilibrio, con diferencias menores a 5.25 unidades.
  - En distribución heterogénea: Presenta desequilibrios severos, con diferencias de hasta 80.33 unidades.

Esto se debe a que la heurística, aunque busca minimizar el tiempo máximo (W_max), no tiene un mecanismo explícito para equilibrar la carga entre zonas, a diferencia del modelo exacto que puede considerar todas las restricciones simultáneamente.

5. Fortalezas y Debilidades de la Heurística

Fortalezas

- **Eficiencia Computacional:** La heurística es significativamente más rápida que la solución exacta, especialmente para instancias grandes.
- **Excelente Rendimiento en Distribución Homogénea:** La heurística obtiene resultados prácticamente óptimos cuando las zonas tienen un número similar de posiciones.
- **Implementación Simple:** El algoritmo es fácil de entender e implementar, lo que facilita su mantenimiento y adaptación.

Debilidades

- **Rendimiento Inferior en Distribución Heterogénea:** La heurística tiene dificultades para encontrar soluciones cercanas al óptimo cuando las zonas tienen diferentes cantidades de posiciones.
- **Desequilibrio entre Zonas:** No logra un equilibrio tan bueno como la solución exacta, especialmente en casos heterogéneos.
- **Dependencia del Orden:** El rendimiento depende significativamente del orden en que se procesan las órdenes, lo que puede llevar a soluciones subóptimas.

6. Posibles Mejoras

6.1. Mejoras en la Heurística

- **Múltiples Ordenamientos:** Probar diferentes estrategias de ordenamiento de órdenes y seleccionar la mejor solución.
- **Búsqueda Local:** Implementar una fase de mejora mediante búsqueda local, intercambiando asignaciones para reducir W_max.
- **Enfoque Metaheurístico:** Utilizar algoritmos como Simulated Annealing o Búsqueda Tabú para escapar de óptimos locales.
- **Equilibrio Explícito:** Modificar la función de evaluación para considerar no solo la minimización de W_max sino también el equilibrio entre zonas.

6.2. Mejoras en la Implementación

- **Optimización del Código:** Utilizar estructuras de datos más eficientes y vectorizar operaciones para mejorar el rendimiento.
- **Paralelización:** Implementar versiones paralelas del algoritmo para aprovechar múltiples núcleos en instancias grandes.
- **Integración de Métodos:** Desarrollar un enfoque híbrido que combine la heurística para una solución inicial rápida y el método exacto para refinarla.

7. Conclusiones

1. **Rendimiento General:** La heurística propuesta obtiene resultados competitivos en la mayoría de las instancias, con un gap promedio del 4.56 % respecto a la solución exacta.

2. **Distribución Homogénea:** La heurística es altamente efectiva en escenarios con distribución homogénea, con gaps promedio del 0.22 %.

3. **Distribución Heterogénea:** El rendimiento es significativamente inferior en escenarios heterogéneos, con gaps promedio del 8.89 %.

4. **Equilibrio:** La solución exacta logra un equilibrio casi perfecto entre zonas en todos los escenarios, mientras que la heurística solo lo consigue en casos homogéneos.

5. **Aplicabilidad:** La heurística es una alternativa viable para la solución rápida de problemas en entornos homogéneos o cuando no se requiere un equilibrio perfecto entre zonas.

El problema de asignación de órdenes en sistemas PTL es un desafío complejo que requiere equilibrar múltiples objetivos. La heurística propuesta ofrece una solución práctica que, aunque no siempre alcanza la optimalidad, proporciona resultados rápidos y de calidad razonable, especialmente en escenarios con distribución homogénea de posiciones. Para escenarios heterogéneos o cuando se requiere un equilibrio perfecto entre zonas, la solución exacta sigue siendo la opción preferida.