Tarea 2: Recocido Simulado para Balancear Cargas en PTL

Heurística 2025-1

1 Introducción

En esta segunda tarea se requiere diseñar un método aleatorizado para el problema de asignación de pedidos a salidas en un sistema Put-to-Light (PTL). La formulación general y varias consideraciones específicas se describen en el informe de la Tarea 1 y en el documento base (Problem Definition). Aquí se mostrará el Recocido Simulado (Simulated Annealing), destacando tres aspectos esenciales:

1. **Representación de las soluciones**,
2. **Manera de perturbarlas/mutarlas** aleatoriamente,
3. **Control general** del algoritmo mediante temperatura y criterio de aceptación.

2 Representación de las Soluciones

La instancia se compone de:

- Conjunto de **pedidos** P, indexado por i.
- Conjunto de **salidas** S, indexado por k.
- Conjunto de **zonas** Z, indexado por j, donde cada zona j tiene capacidad n_sal_j (número de salidas reales).

2.1 Variable de decisión

La asignación de pedidos se modela con un diccionario (o arreglo) sol, donde

$$\text{sol}[i] \;=\; k \quad (\forall\, i \in P)$$

significa que el pedido i se ubica en la salida k. Con esta estructura:

- Cada pedido se asigna a exactamente una salida (no se parte).
- No se permite que dos pedidos coincidan en la misma salida.

2.2 Restricciones y tiempo por zona

Asumimos que la salida k pertenece a la zona j si $s[(j,k)] = 1$. El tiempo total de la zona j, llamado W_j, se define como la suma de los tiempos de los pedidos asignados a las salidas de esa zona. La meta es minimizar:

$$W_{\max} \;=\; \max_{j \in Z} \{W_j\}.$$

Para cumplir la restricción de no exceder la capacidad de la zona, se limita la cantidad de pedidos en cada j a n_sal_j.

3 Perturbación (Mutación Aleatoria)

El espacio de soluciones se explora aplicando modificaciones (vecindades) a la solución actual. Las dos operaciones principales son:

3.1 Swap (Intercambio)

$$\text{Sean } i_1, i_2 \in P, \quad \text{swap} : \big(\text{sol}[i_1],\, \text{sol}[i_2]\big) \mapsto \big(\text{sol}[i_2],\, \text{sol}[i_1]\big).$$

Es decir, se intercambian las salidas de dos pedidos distintos.

3.2 Move (Reubicación)

$$\text{Sea } i \in P, \quad \text{move} : \text{sol}[i] \leftarrow k' \neq \text{sol}[i].$$

Se elige un pedido i y se le cambia su salida a otra que no sea la actual.

En ambos casos, si la nueva asignación viola las restricciones (p.ej. dos pedidos en la misma salida o la zona excedida), se descarta sin evaluar la función objetivo.

4 Control General del Recocido Simulado

El Recocido Simulado (RS) introduce una temperatura T que decrece, reduciendo progresivamente la probabilidad de aceptar soluciones de peor calidad.

4.1 Parámetros Principales

- T_0: **temperatura inicial**, alta para tolerar saltos amplios.
- $T_{\min}$: **temperatura mínima** que define el fin del proceso.
- α: **factor de enfriamiento**, con $0 < \alpha < 1$.
- iter_por_temp: número de vecinos evaluados por cada nivel de temperatura.
- $C(\cdot)$: **función de costo**, que en este caso es $W_{\max}$ (o alguna otra métrica de balance).

4.2 Mecánica de Aceptación de Soluciones

Se define:

$$\Delta \;=\; C(\mathrm{sol}_{\mathrm{vecino}}) \;-\; C(\mathrm{sol}_{\mathrm{actual}}).$$

- Si $\Delta < 0$ (mejora), la solución vecina se *acepta* inmediatamente.
- Si $\Delta \geq 0$ (empeora), se acepta con probabilidad $\exp(-\Delta/T)$.

Así, con temperaturas altas, la probabilidad de aceptar soluciones peores es más grande, promoviendo la exploración del espacio. Conforme T desciende, se vuelve más rígido y sólo se aceptan muy pocas soluciones peores.

4.3 Pseudocódigo Detallado

```
RecocidoSimulado(T0, Tmin, alpha, iter_por_temp, modo_vecindad, objetivo):
  1. sol_actual = GenerarSolucionInicial()
  2. Mientras (no es_factible(sol_actual)):
       sol_actual = GenerarSolucionInicial()
  3. costo_actual = C(sol_actual, objetivo)
  4. mejor_sol = sol_actual
  5. mejor_costo = costo_actual

  6. T = T0
  7. Mientras (T > Tmin):
     7.1. Para n = 1 hasta iter_por_temp:
          (a) vecino = GenerarVecino(sol_actual, modo_vecindad)
          (b) Si (not es_factible(vecino)): continuar
          (c) costo_vecino = C(vecino, objetivo)
          (d) delta = costo_vecino - costo_actual

          (e) Si (delta < 0):
                 sol_actual = vecino
                 costo_actual = costo_vecino
              Sino:
                 prob = exp(-delta / T)
                 si random() < prob:
                    sol_actual = vecino
                    costo_actual = costo_vecino

          (f) Si costo_actual < mejor_costo:
               mejor_sol = sol_actual
               mejor_costo = costo_actual

     7.2. T = alpha * T

  8. Retornar (mejor_sol, mejor_costo)
```

Interpretación de variables:

- `sol_actual`: solución corriente del RS.
- `costo_actual`: valor de la función de costo para `sol_actual`.
- `vecino`: una solución perturbada mediante *swap* o *move*.
- `delta` (Δ): diferencia de costos ($C(\text{vecino}) - C(\text{actual})$).
- `T`: la temperatura actual, que se va reduciendo al multiplicarse por *alpha*.
- `mejor_sol` y `mejor_costo`: memoria de la mejor solución global encontrada.

5 Conclusiones

La solución propuesta combina:

- **Representación concisa**: cada pedido se mapea a una salida, sin duplicaciones.
- **Mutación aleatoria**: con operaciones swap y *move*, se generan vecinos y se descartan si no cumplen las restricciones (exceder zonas o compartir salidas).
- **Control de Recocido Simulado**: a través de la temperatura T y el criterio de aceptación probabilístico de soluciones peores, el algoritmo evita atascarse en mínimos locales y converge paulatinamente a una buena asignación, minimizando $W_{\max}$.

Este esquema, con parámetros bien afinados ($T_0, \alpha, \text{iter\_por\_temp}$), permite obtener soluciones de alta calidad en balanceo de cargas.