Capítulo 1

Aceptación por umbrales

1.1. Preliminares

Dado un problema $\mathscr{P}$ de optimización y clasificado como *NP*-duro, sea S el conjunto de posibles soluciones a una instancia de $\mathscr{P}$. Vamos a suponer que tenemos una función $f : S \to \mathbb{R}^+$ (que llamaremos *función objetivo*), tal que $0 \leq f(s) < \infty$ para cualquier $s \in S$. Dadas $s, s' \in S$, si $f(s) < f(s')$, entonces consideraremos a la solución s mejor que a la solución s'.

Si, de alguna manera bien definida, podemos convertir una solución s en una solución distinta s' (y obviamente la operación inversa convierte s' en s), consideraremos a s y s' como soluciones *vecinas*. Esta relación determina una gráfica, que *no necesariamente* es conexa.

La idea central de la aceptación por umbrales es, dada una temperatura inicial $T \in \mathbb{R}^+$ y una solución inicial s (ambas obtenida de alguna manera), de forma aleatoria buscar una solución vecina s' tal que $f(s') \leq f(s) + T$, y entonces actualizar s para que sea s'; en este caso diremos que la solución s' es *aceptada*. Continuamos de esta manera mientras la temperatura T es disminuida paulatinamente siguiendo una serie de condiciones; el proceso termina cuando $T < \varepsilon$ (para $\varepsilon > 0$ muy pequeña); cuando hemos generado un determinado número de soluciones aceptadas; o cuando otra serie de condiciones es satisfecha.

Los algoritmos no deterministas de la heurística serán analizados en la siguiente sección.

1.2. La heurística

Como mencionábamos arriba, la heurística es buscar soluciones mejores a la solución actual, o peores dentro de un umbral determinado por la temperatura. Dicha temperatura va decrementando su valor, y cuando sea menor a ε, la heurística termina.

Las condiciones para decrementar la temperatura T estarán dadas por el comportamiento de lo que llamaremos lotes; un lote será un número determinado de soluciones aceptadas. Sea $L \in \mathbb{N}^+$, determinado por vías experimentales; el algoritmo (no determinista) para calcular un lote se puede ver en el procedimiento 1.

Procedimiento 1 Calcula lote

```
procedure CALCULALOTE(T, s)                    ▷ Temperatura T, solución s
    c ← 0
    r ← 0.0
    while c < L do
        s' ← VECINO(s)                 ▷ VECINO(s) obtiene un vecino aleatorio de s
        if f(s') ≤ f(s) + T then                ▷ Aceptamos la solución vecina s'
            s ← s'
            c ← c + 1
            r ← r + f(s')
    return r/L, s   ▷ Promedio de las soluciones aceptadas y última solución aceptada
```

Es necesario notar que el **while** puede no terminar; si VECINO no puede encontrar una solución vecina de s que sea aceptada, entonces el número de soluciones aceptadas puede nunca llegar a L, por lo que la condición de paro puede no alcanzarse. Deben existir medidas que permitan al ciclo terminar si el lote no puede completarse: lo más común es tener un número máximo de intentos, significativamente mayor que L.

Con los lotes definidos, podemos revisar el algoritmo principal de la aceptación por umbrales en el procedimiento 2. Como este algoritmo utiliza CALCULALOTE, tampoco es determinista.

Procedimiento 2 Aceptación por umbrales

```
procedure ACEPTACIONPORUMBRALES(T, s)  ▷ Temperatura T y solución s iniciales
    p ← 0
    while T > ε do
        q ← ∞
        while p ≤ q do                          ▷ Mientras no haya equilibrio térmico
            q ← p
            p, s ← CALCULALOTE(T, s)
        T ← φT
```

El valor ε ($0 \leq \varepsilon < \infty$) es otro cero virtual determinado de antemano. El valor φ ($0 < \varphi < 1$) es el factor de enfriamiento, y determina qué tan lento o rápido la temperatura T va disminuyendo; un valor muy cercano a 1 hará que el sistema se enfríe muy lentamente, lo que permitirá encontrar muy buenas soluciones, pero que causará que la ejecución del mismo sea excesivamente

tardada. Inversamente, si φ es muy cercano a 0, el sistema se enfriará muy rápido y la ejecución será igualmente rápida; pero probablemente las soluciones encontradas serán de muy baja calidad.

Obviamente, a lo largo de la ejecución del sistema guardamos en una variable global a la solución s mínima, para que esa sea la que entreguemos como resultado al finalizar la corrida; en aras de mantener sencillo el pseudocódigo, omitimos las instrucciones que inicializan y actualizan esta variable global.

La calidad del sistema (cuánto tiempo tarda, qué tan buenas son las soluciones que produce) está determinado por todos los parámetros libres: el tamaño L de lote; el número máximo de veces que intentaremos aceptar una solución vecina de la solución actual al tratar de completar un lote; el factor de enfriamiento φ; y el cero virtual ε para la temperatura.

No hay un consenso en la literatura de cómo deben calcularse estos valores; en general se recurre a la experimentación computacional, ejecutando corridas del sistema con distintos valores para los parámetros y analizando los resultados obtenidos.

Otro factor importante es el procedimiento VECINO, que genera una solución vecina **aleatoria** a partir de una solución dada. Dicho vecino debe ser obtenido de manera aleatoria con distribución uniforme, para poder garantizar una búsqueda no sesgada en nuestro espacio de soluciones.

Como se busca tener resultados reproducibles, el generador de números aleatorios (RNG, por sus siglas en inglés) debe ser cuidadosamente mantenido, y la semilla con la que es inicializado nos permite poder reproducir los resultados, además de que facilita la paralelización del sistema al poder ejecutar distintas corridas con semillas diferentes.

Por último, no hemos mencionado cómo calcular la solución inicial, ni cómo obtener una temperatura inicial adecuada; del manera similar a φ, la temperatura inicial puede ser muy alta (en cuyo caso la ejecución del sistema tardará demasiado) o muy baja (en cuyo caso las soluciones obtenidas sean probablemente de baja calidad). El escenario inicial es dependiente del problema en cuestión; pero sí existen algoritmos para calcular una temperatura inicial satisfactoria. Veremos esto en la siguiente sección.

1.3. La temperatura inicial

Como mencionábamos arriba, la temperatura inicial afecta en gran medida el comportamiento del sistema; queremos una temperatura inicial suficientemente alta para evitar caer en un mínimo local muy temprano; pero suficientemente baja para el sistema no tarde demasiado en terminar.

La temperatura inicial se puede determinar a través de experimentación computacional (como ocurre con todos los parámetros del sistema), pero contamos con un algoritmo que ha sido utilizado exitosamente en un sistema de distritación electoral.

La idea del algoritmo es obtener una temperatura T que aumente la probabilidad de que la heurística pueda desplazarse rápidamente por el espacio

de búsqueda (al menos al inicio), sin que la misma sea excesivamente grande. Para esto realizaremos una búsqueda binaria de una T tal que, con alta probabilidad, acepte un porcentaje P elevado de soluciones vecinas de un escenario inicial, pero que no acepte *todas*. En la distritación electoral un porcentaje $.85 \leq P \leq .95$, es utilizado.

Podemos ver el algoritmo en el procedimiento 3:

Procedimiento 3 Temperatura inicial

```
procedure TEMPERATURAINICIAL(s, T, P)
    p ← PORCENTAJEACEPTADOS(s, T)
    if |P − p| ≤ ε_P then
        return T
    if p < P then
        while p < P do
            T ← 2T
            p ← PORCENTAJEACEPTADOS(s, T)
        T_1 ← T/2
        T_2 ← T
    else
        while p > P do
            T ← T/2
            p ← PORCENTAJEACEPTADOS(s, T)
        T_1 ← T
        T_2 ← 2T
    return BUSQUEDABINARIA(s, T_1, T_2, P)
```

El algoritmo PORCENTAJEACEPTADOS lo podemos ver en el procedimiento 4, y el algoritmo BUSQUEDABINARIA en el procedimiento 5.

Procedimiento 4 Porcentaje aceptados

```
procedure PORCENTAJEACEPTADOS(s, T)
    c ← 0
    for i = 1 to N do
        s' ← VECINO(s)
        if f(s') ≤ f(s) + T then
            c ← c + 1
            s ← s'
    return c/N
```

Como el procedimiento 4 una vez más utiliza el algoritmo VECINO, tampoco es determinístico, y tampoco lo es TEMPERATURAINICIAL.

El valor ε_P es otro cero virtual que nos permite detener el algoritmo más rápido, dado que es muy poco probable que el resultado de PORCENTAJEACEPTADOS sea idéntico a P.

Procedimiento 5 Búsqueda binaria

```
procedure BUSQUEDABINARIA(s, T_1, T_2, P)
    T_m ← (T_1 + T_2)/2
    if T_2 − T_1 < ε_P then
        return T_m
    p ← PORCENTAJEACEPTADOS(s, T_m)
    if |P − p| < ε_P then
        return T_m
    if p > P then
        return BUSQUEDABINARIA(s, T_1, T_m)
    else
        return BUSQUEDABINARIA(s, T_m, T_2)
```

Puede parecer contradictorio que para buscar una temperatura inicial necesitemos proveer una temperatura incial; sin embargo, la búsqueda binaria funciona con una complejidad en tiempo (relativa) logarítmica, lo que permite cambiar rápidamente su valor sin importar demasiado cuál es el original, siempre y cuando $0 \leq T \leq 2^k$ para una k muy grande. En la distritación electoral una $T = 8$ funciona aparentemente bien.