Optimización y PSR
TPN°3

Barraquero Ignacio, Campo Camila, Villarreal Francisco, Marzari Agustina

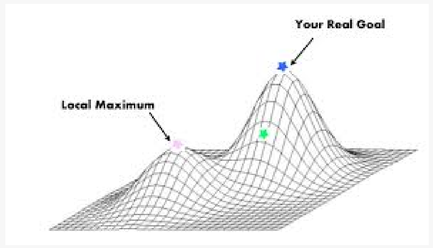
Facultad de Ingeniería

Introducción a la Optimización y PSR

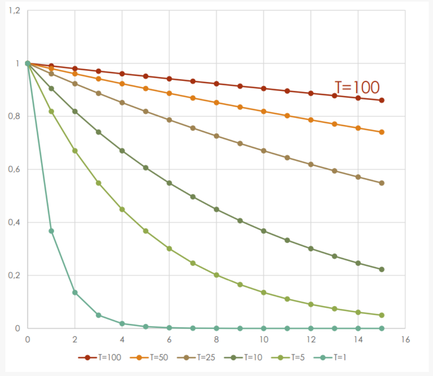
En este trabajo se abordaron estrategias de búsqueda no informada y basada en heurísticas, con foco en búsqueda local, algoritmos evolutivos y problemas de satisfacción de restricciones (PSR). Estos métodos se aplican en situaciones donde el espacio de búsqueda es muy grande y no resulta eficiente explorar todas las posibilidades. Por eso, se utilizan heurísticas o mecanismos de inspiración biológica que permiten encontrar soluciones aceptables en tiempos razonables.

Estrategias de búsqueda local

- La búsqueda local trabaja sobre un único estado y se mueve hacia vecinos, en lugar de mantener múltiples caminos.
- **Ejemplo: Ascensión de Colinas**
- Es un algoritmo que avanza hacia el estado vecino que presenta la mejor mejora respecto al estado actual.



- **Ejemplo: Recocido Simulado**
- Permite aceptar soluciones peores con cierta probabilidad que depende de una temperatura, lo que ayuda a escapar de máximos locales.



Algoritmos Evolutivos

- Los algoritmos evolutivos son metodologías de optimización inspiradas en la evolución natural, que trabajan con poblaciones de soluciones
- **Individuos:** Representación de soluciones posibles (por ejemplo, qué cajas cargar en una grúa).
- **Población:** Conjunto de individuos
- **Evaluación** / **Fitness:** Se asigna un valor que indica qué tan buena es la solución.
- **Selección:** Se eligen individuos para reproducción, normalmente usando métodos como la ruleta o el torneo

Ejercicio 5

Hay que buscar el maximo local de una funcion con el algoritmo de hill climbing

```python
# Preparar la figura
xs = np.linspace(x_min, x_max, 1000)
ys = f(xs)

fig, ax = plt.subplots()
ax.plot(xs, ys, label="f(x) = sin(x)/(x+0.1)")
point, = ax.plot([], [], 'ro', markersize=8)
ax.set_title("Hill Climbing en acción")
ax.set_xlabel("x")
ax.set_ylabel("f(x)")
ax.legend()

# Estado global de la animación
state = {"x": x0, "step": step}

# Inicialización del punto en la animación
def init():
    point.set_data([], [])
    return point,

# Función de actualización de cada frame
def update(frame):
    x, step = state["x"], state["step"]
```

implementamos una interfaz que muestra la curva en el intervalo dado y el candidato actual, va evolucionando en base al algoritmo

Ejercicio 6

Hay que buscar la mejor combinacion de cajas que maximice el precio sin sobrecargar el camion.
Esto se logra mediante el algoritmo de genetica

La resolución muestra como con distintos algoritmos se obtienen distintas trayectorias

```
=== RESULTADO FINAL ===
Mejor individuo: [1, 0, 0, 0, 1, 0, 0, 0, 0, 0]
Peso total: 964
Precio total: 300
```