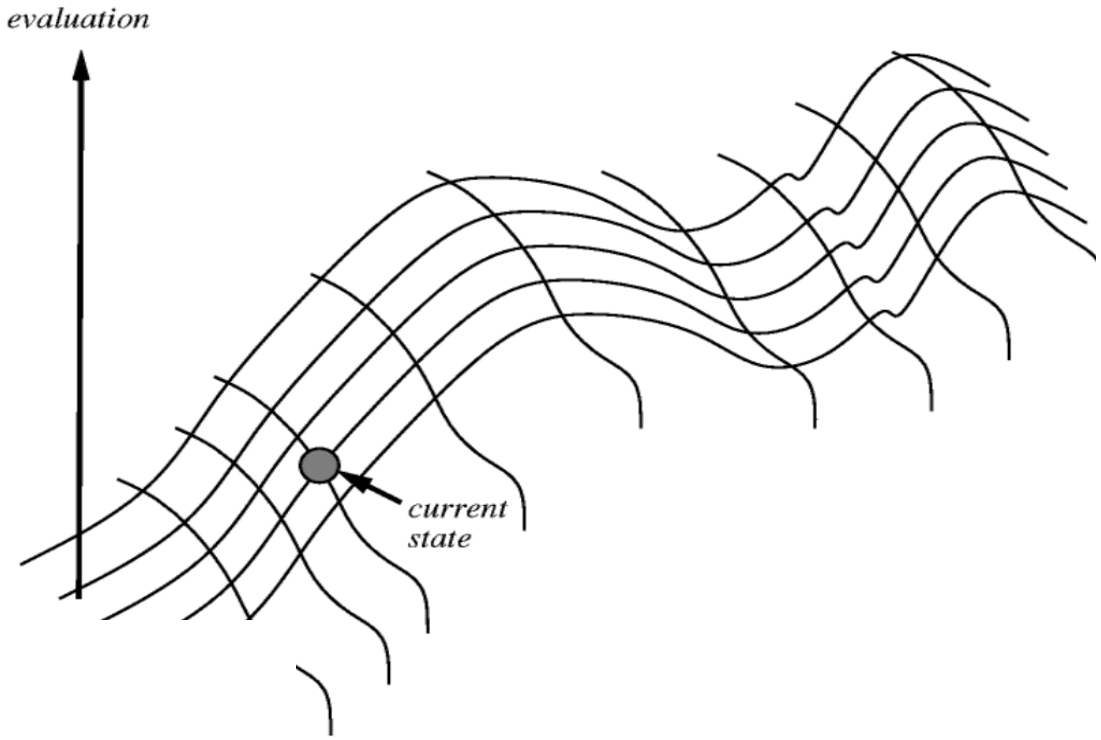
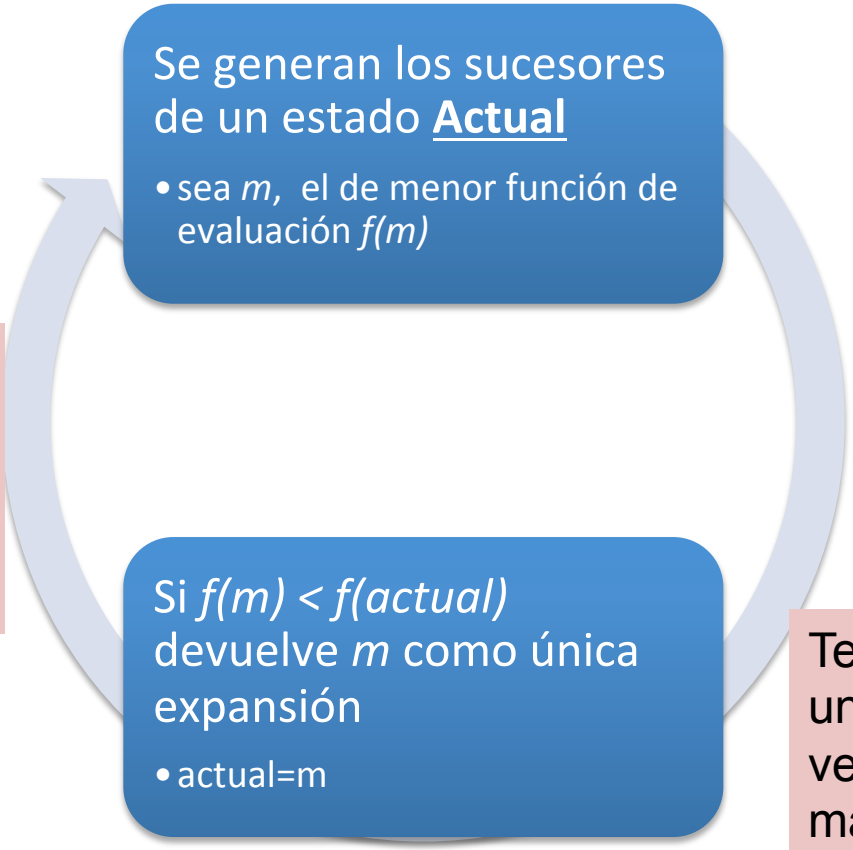
4. ALGORITMOS DE ESCALADA (Hill-Climbing)

- Bucle que continuamente se mueve en la dirección del valor:
  - creciente (si se trata de maximizar una función de coste)
  - decreciente (si se trata de minimizar la función de coste)



4.1 Búsqueda en Escalada

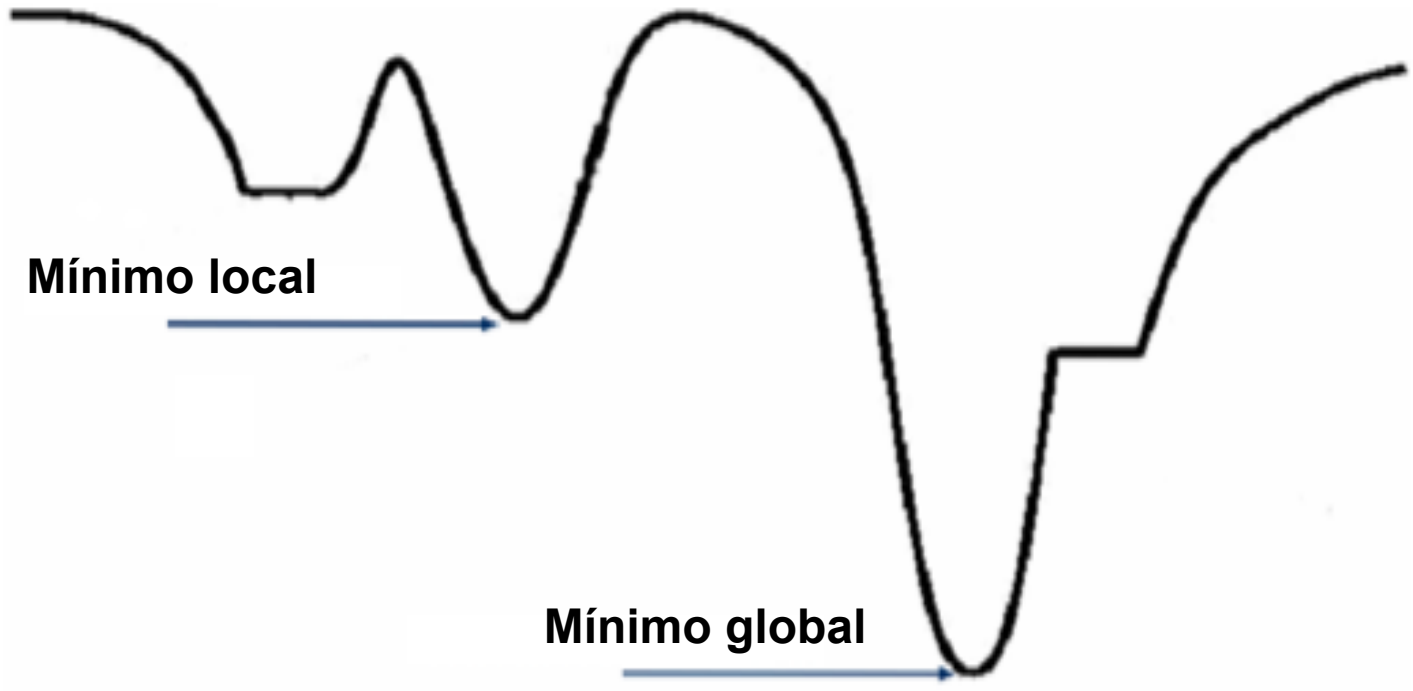
```
vecino = sucesor de actual con f menor
si f(vecino) ≤ f(actual)
        actual = vecino
```



Sigue el recorrido a través de los nodos en los que el valor de dicha función sea mínimo

Termina cuando alcanza un "pico" donde ningún vecino tiene un valor más bajo

4.2 Problemas



4.3 Alternativa: Búsqueda por Haz Local

- Comienza con estados generados aleatoriamente
- Guarda la pista de k estados.
- Mientras alguno no sea objetivo
  - Se generan todos los sucesores de los k estados.
  - Se seleccionan los k mejores sucesores de la lista completa y se repite el proceso

4.3 Búsqueda Local para PSR

- **Estados:** asignaciones completas (consistentes o inconsistentes)
- **Estado inicial:** escogido aleatoriamente
- **Estado final:** solución al PSR
- **Generación de un sucesor:** elegir una variable y cambiar el valor que tiene asignado (heurística y aleatoriedad)

- Heurística de **Mínimos Conflictos**:
  - Seleccionar **variable** distinta de la última modificada que participa en más restricciones NO satisfechas en el estado
  - Seleccionar **valor** que cause el mínimo número de conflictos con otras variables (Valor Menos Restringido)

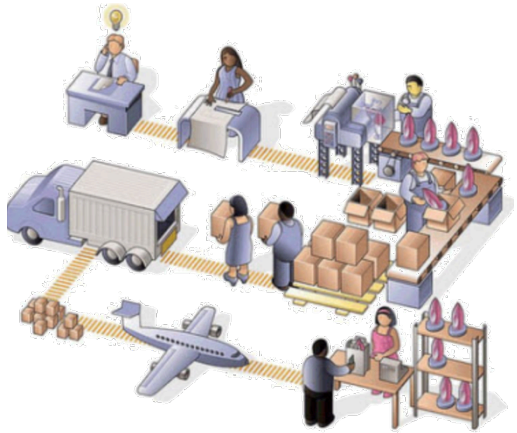
5. Conclusiones

Cuando el camino a la solución es irrelevante:

- Guardan sólo un estado en memoria: el estado actual
- Se mueven sólo a los nodos vecinos del nodo actual
- No son sistemáticos en la búsqueda
- Utilizan poca memoria
- Pueden encontrar soluciones razonables en espacios de estados grandes o infinitos
- Pueden quedar atrapados en máximos/mínimos locales

Problema de Planificación de Tareas

- Supongamos que N trabajadores tienen que realizar N tareas.



	Tarea 1	Tarea 2	Tarea 3	Tarea 4
trabajador 1	12	43	15	7
trabajador 2	9	10	6	4
trabajador 3	5	13	29	2
trabajador 4	4	11	17	9

- El problema consiste en asignar a cada trabajador una y sólo una tarea, de manera que se realicen todas las tareas en un tiempo total mínimo.

Implementación

- Asignacion=[1 2 3 4]
- Coste= 12+10+29+9=60

Tiempos = [

12	43	15	7
9	10	6	4
5	13	29	2
4	11	17	9]

- **Evaluación de una asignación de tareas (o lista de asignaciones)**

  function C=**Coste**(Tiempos, Asignacion)

- **Sucesores**: mediante intercambio de 2 tareas

  function Lista=**Sucesores**(Asignacion, Trabajador)

function [X, V]=busquedaLocal(Tiempos, actual)

```
%% X= asignación final encontrada  V=Coste de X
%% realiza una busqueda local para encontrar una asignacion mínima
%% 1. inicializaciones


%% 2. bucle de búsqueda
while  (condiciones)
   varactual=mod(itera,N)+1;
   L=Sucesores(actual,varactual);
   C=Costes(Tiempos,L);
  %% Elige sucesor con mínimo coste
  ..............
  %% Actualiza la asignación del nodo actual si procede
  ...........
end
```

Bibliografía

- Russell, S. y Norvig, P. *Inteligencia Artificial (un enfoque moderno) (Pearson Educación, 2004). Segunda edición.* Cap. 5: "Problemas de Satisfacción de Restricciones"
- Nilsson, N.J. *Inteligencia artificial (una nueva síntesis)* (McGraw–Hill, 2000). Cap. 11 "Métodos alternativos de búsqueda y otras aplicaciones"
- Poole, D.; Mackworth, A. y Goebel, R. *Computational Intelligence (A Logical Approach) (Oxford University Press,* 1998) "Constraint Satisfaction Problems"