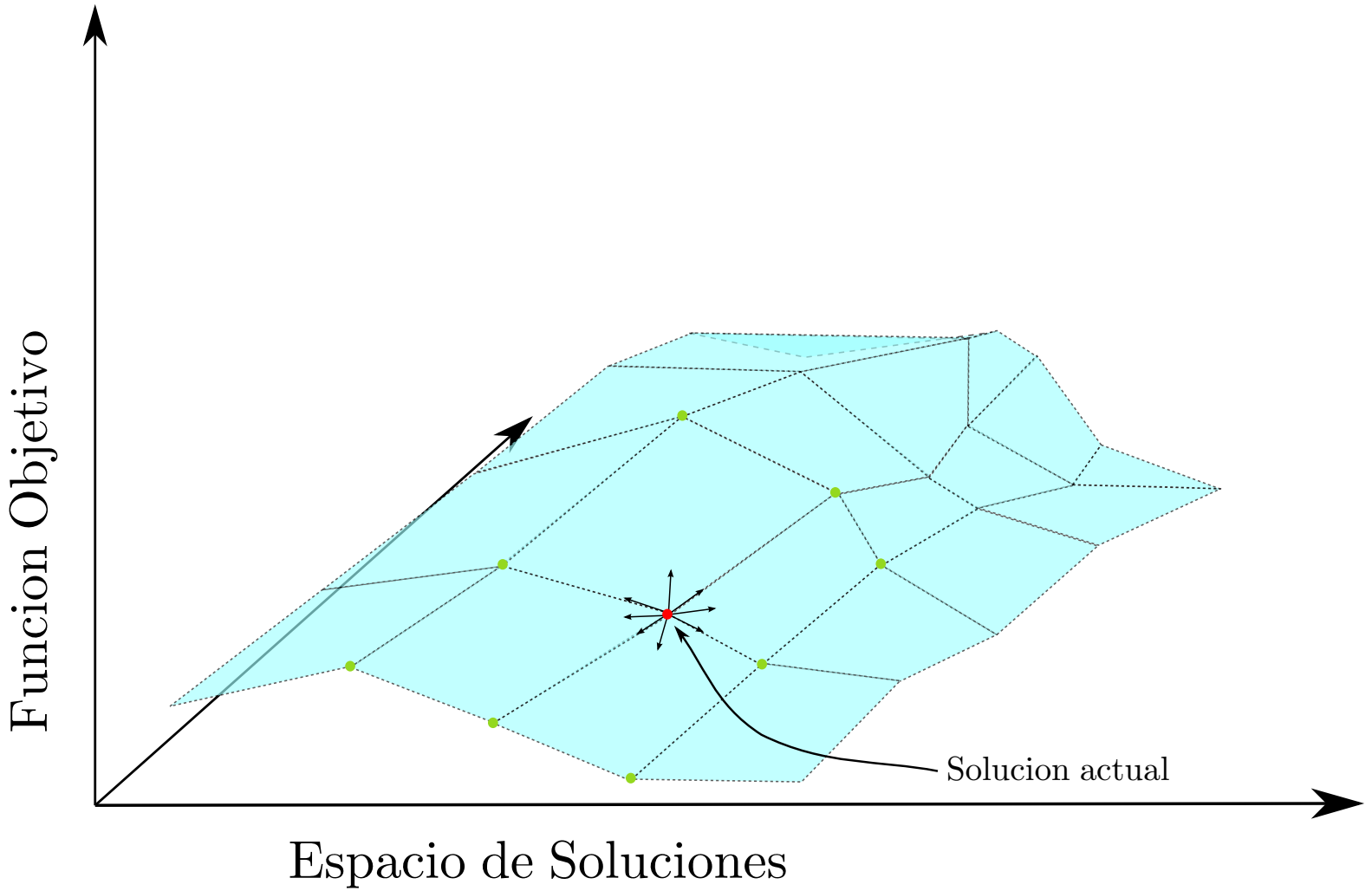
Búsqueda Local

- A veces el camino para llegar a la solución no nos importa, buscamos en el espacio de soluciones
- Queremos la mejor de entre las soluciones posibles alcanzable en un tiempo razonable (el óptimo es imposible)
- Tenemos una función que nos evalúa la calidad de la solución, pero que no esta ligada a ningún coste necesariamente
- La búsqueda se realiza desde una solución inicial que intentamos mejorar modificándola (operadores)
- Los operadores nos mueven entre soluciones vecinas

Búsqueda Local

- La función heurística:
  - Aproxima la calidad de una solución (no representa un coste)
  - Hemos de optimizarla (maximizarla o minimizarla)
  - Combinará los elementos del problema y sus restricciones (posiblemente con diferentes pesos)
  - No hay ninguna restricción sobre como ha de ser la función, solo ha de representar las relaciones de calidad entre las soluciones
  - Puede tomar valores positivos o negativos

Búsqueda Local



Búsqueda Local

- El tamaño del espacio de soluciones por lo general no permite obtener el óptimo
- Los algoritmos no pueden hacer una exploración sistemáticaProblems
- La función heurística se usará para podar el espacio de búsqueda (soluciones que no merece la pena explorar)
- No se suele guardar historia del camino recorrido (el gasto de memoria es mínimo)
- La falta total de memoria puede suponer un problema (bucles)

Escalada (Hill climbing)

- Escalada simple
  - Se busca cualquier operación que suponga una mejora respecto al padre
- Escalada por máxima pendiente (steepest-ascent hill climbing, gradient search)
  - Se selecciona el mejor movimiento (no el primero de ellos) que suponga mejora respecto al estado actual

Hill Climbing

Algoritmo: Hill Climbing

```
Actual ← Estado_inicial
fin ← falso
mientras no fin hacer
    Hijos ← generar_sucesores(Actual)
    Hijos ← ordenar_y_eliminar_peores(Hijos, Actual)
    si no vacio?(Hijos) entonces
        Actual ← Escoger_mejor(Hijos)
    sino
        fin ← cierto
    fin
fin
```

- Sólo se consideran los descendientes cuya función de estimación es mejor que la del padre (poda del espacio de búsqueda)
- Se puede usar una pila y guardar los hijos mejores que el padre para hacer backtracking, pero por lo general es prohibitivo
- Es posible que el algoritmo no encuentre una solución aunque la haya

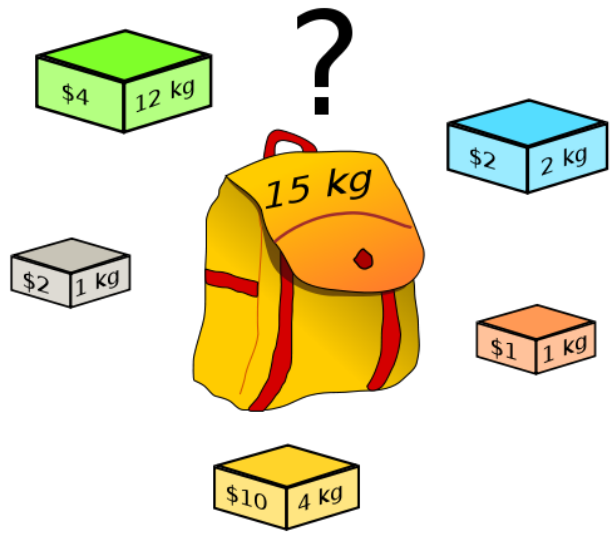
Hill climbing

- Las características de la función heurística y la solución inicial determinan el éxito y rapidez de la búsqueda
- La estrategia del algoritmo hace que la búsqueda pueda acabar en un punto donde la solución sólo sea la óptima aparentemente
- Problemas
  - Máximo local: Ningún vecino tiene mejor coste
  - Meseta: Todos los vecinos son iguales
  - Cresta: La pendiente de la función sube y baja (efecto escalón)

Hill climbing

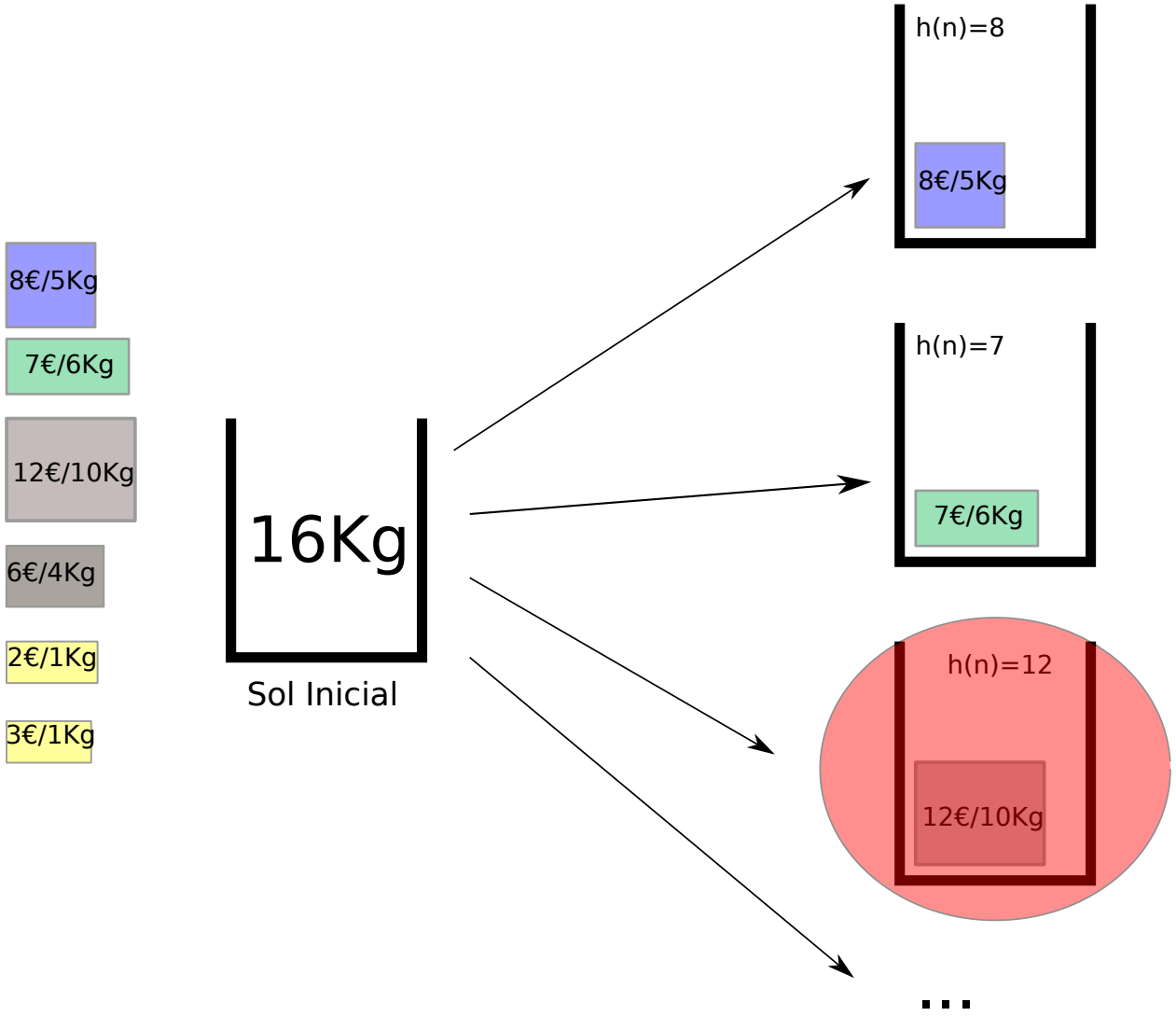
- Posibles soluciones
  - Reiniciar la búsqueda en otro punto buscando mejorar la solución actual (*Random Restarting Hill Climbing*)
  - Hacer backtracking a un nodo anterior y seguir el proceso en otra dirección (solo posible limitando la memoria para hacer el backtracking, *Beam Search*)
  - Aplicar dos o más operaciones antes de decidir el camino
  - Hacer HC en paralelo (p.ej. Dividir el espacio de búsqueda en regiones y explorar las más prometedoras, posiblemente compartiendo información)

Hill Climbing - Ejemplo - Knapsack problem

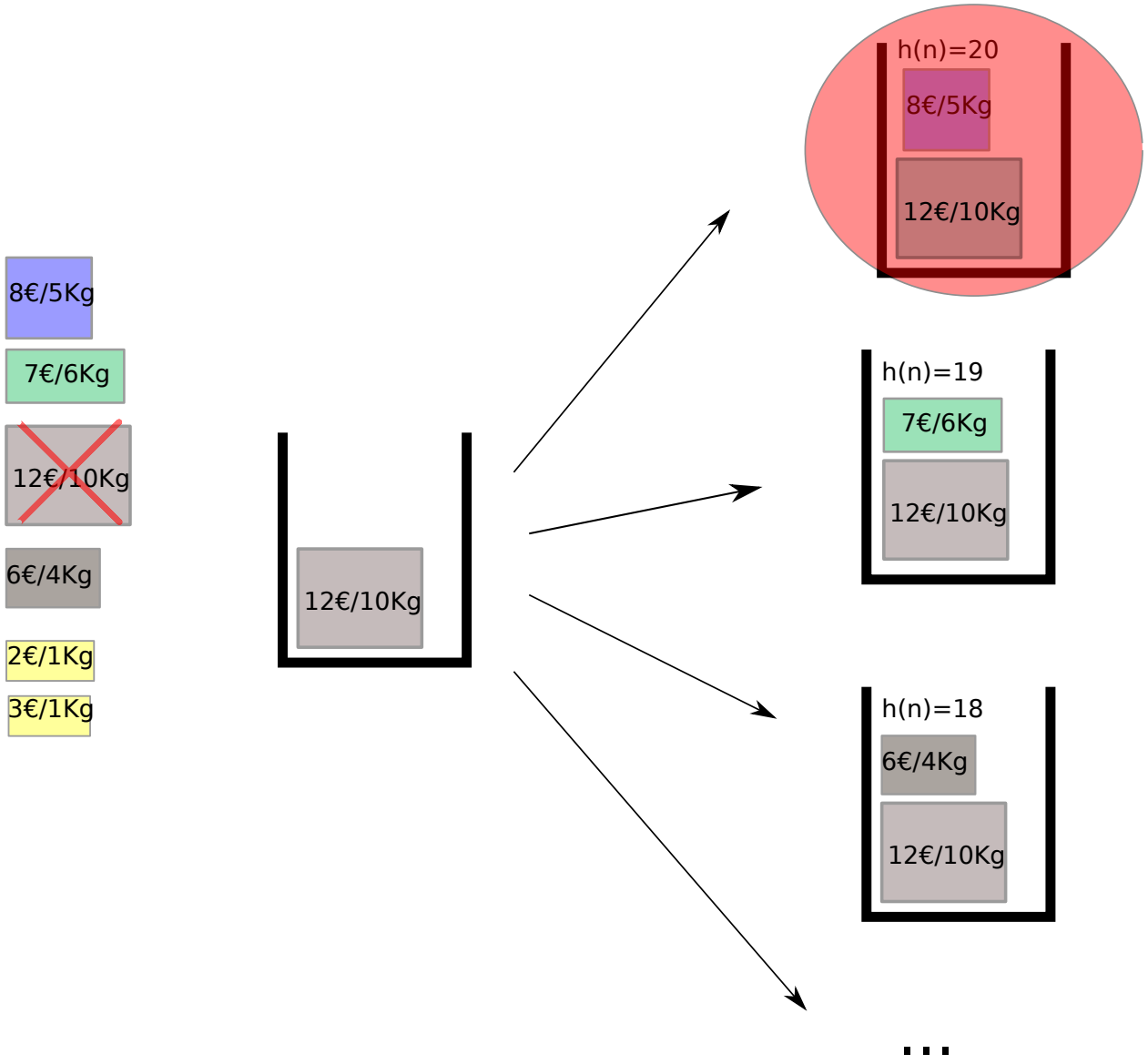


- Solución: Cualquier combinación de objetos en la mochila
- Solución Inicial: Mochila vacía
- Operadores: Meter y sacar objetos de la mochila
- Función heurística: $max \sum_i Valor_i$ o $max \sum_i \frac{Valor_i}{Peso_i}$

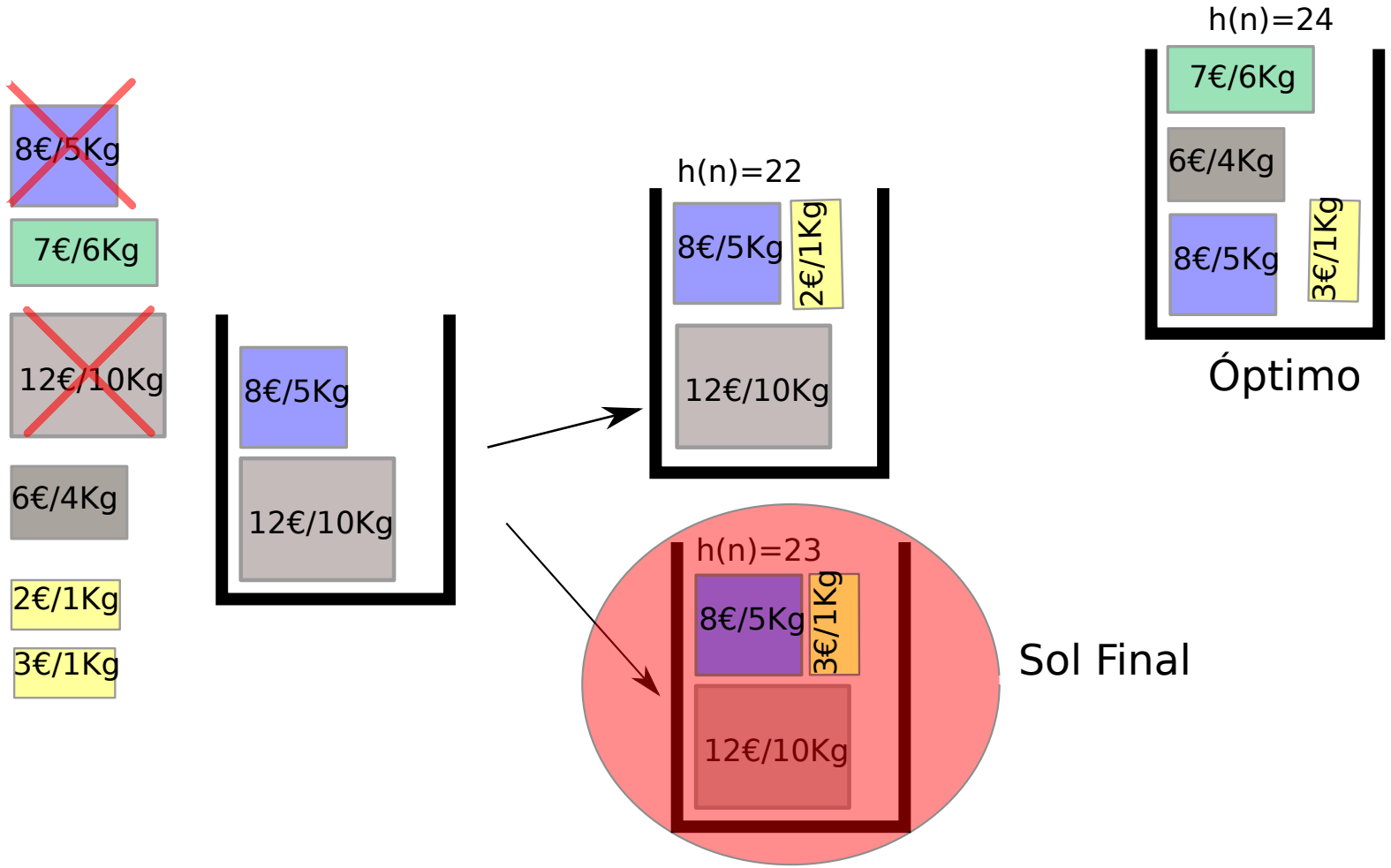
Hill Climbing - Ejemplo - Knapsack problem



Hill Climbing - Ejemplo - Knapsack problem



Hill Climbing - Ejemplo - Knapsack problem



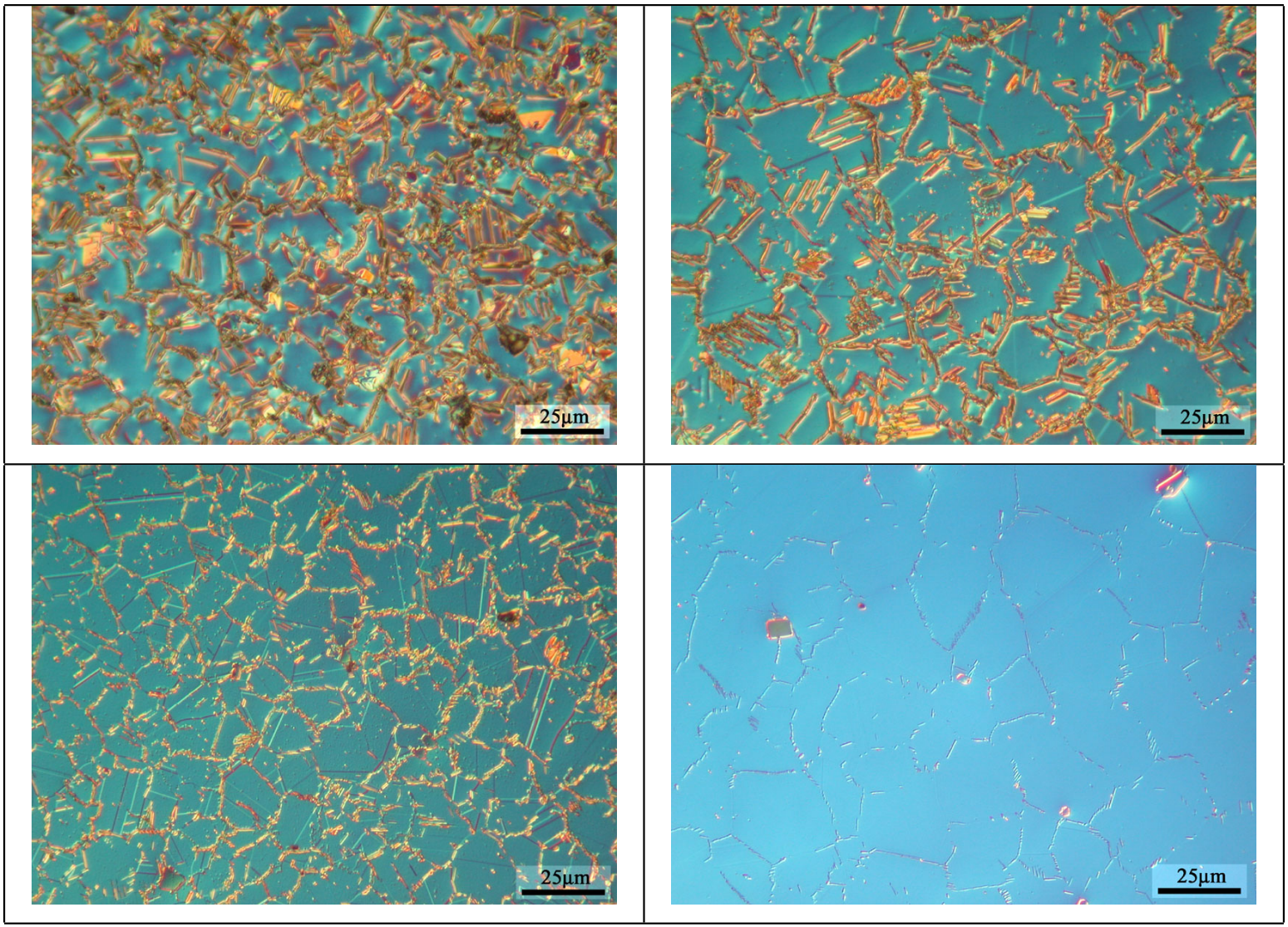
Otros algoritmos de búsqueda local

- Se han planteado otros algorimos inspirados en analogías físicas y biológicas:
  - **Simulated annealing:** Hill-climbing estocástico inspirado en el proceso de enfriamiento de metales
  - **Algoritmos genéticos:** Hill-climbing paralelo inspirado en los mecanismos de selección natural
  - Ambos mecanismos se aplican a problemas reales con bastante éxito
- Pero también Particle Swarm Optimization, Ant Colony Optimization, Intelligent Water Drop, Gravitational search algorithm, ...

Simulated Annealing

- Es un algoritmo de Hill-Climbing estocástico (elegimos un sucesor de entre todos los posibles según una distribución de probabilidad, el sucesor podría ser peor)
- Hacemos paseos aleatorios por el espacio de soluciones
- Inspirado en el proceso físico de enfriamiento controlado (cristalización, templado de metales)
- Se calienta un metal/disolución a alta temperatura y se enfría progresivamente de manera controlada
- Si el enfriamiento es adecuado se obtiene la estructura de menor energía (mínimo global)

Simulated Annealing



Simulated Annealing - Metodología

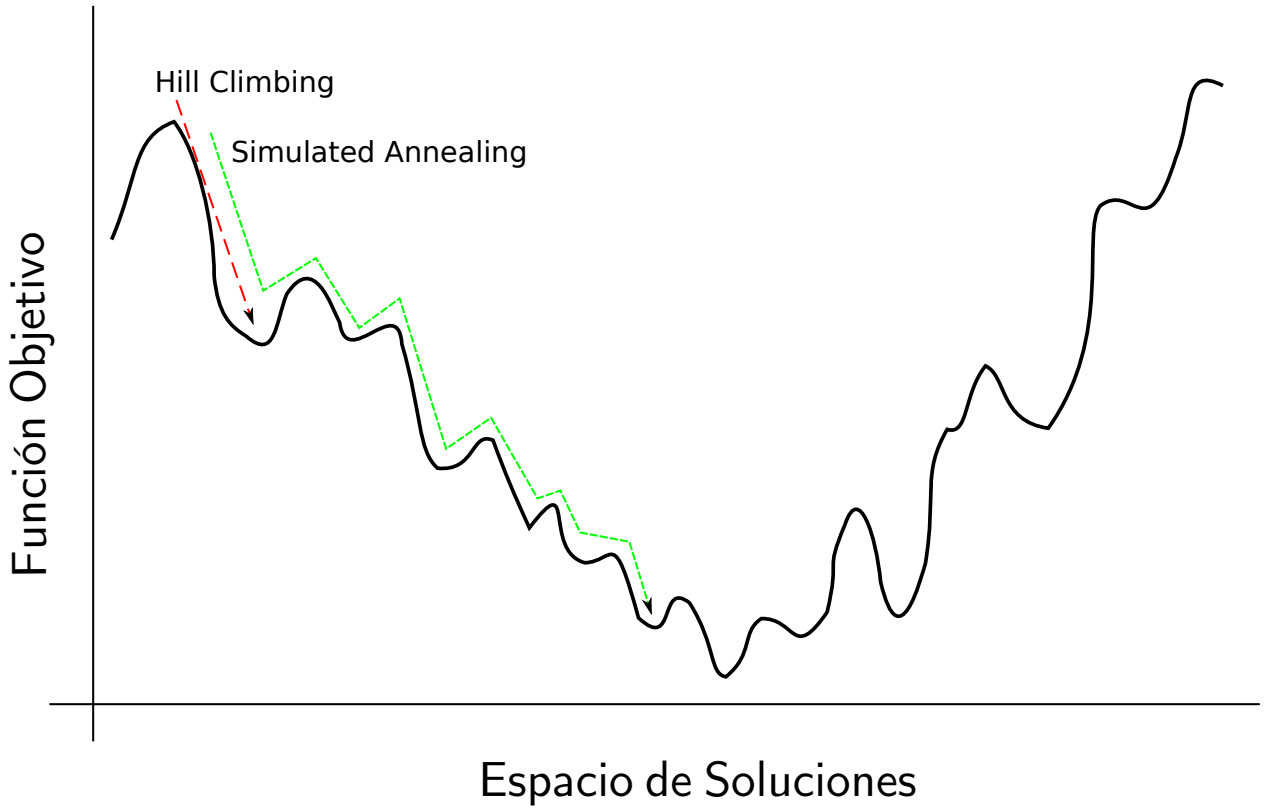
- Debemos identificar los elementos del problema con los del problema físico
- **Temperatura**, parámetro de control
- **Energía**, calidad de la solución $f(n)$
- **Función de aceptación**, permite decidir si escoger un nodo sucesor
  - $\mathcal{F}(\Delta f, T)$, función de la temperatura y la diferencia de calidad entre la solución actual y la solución candidata
  - A menor temperatura menor probabilidad de elegir sucesores peores
- **Estrategia de enfriamiento**, número de iteraciones a realizar, como bajar la temperatura y cuantos sucesores explorar para cada paso de temperatura

Simulated annealing - Algoritmo Básico

Algoritmo: Simulated Annealing

Partimos de una temperatura inicial
mientras *la temperatura no sea cero* **hacer**
    `// Paseo aleatorio por el espacio de soluciones`
    para *un numero prefijado de iteraciones* **hacer**
        Enuevo $\leftarrow$ Genera_sucesor_al_azar(Eactual)
        $\Delta E \leftarrow f(Eactual) - f(Enuevo)$
        si $\Delta E > 0$ **entonces**
            Eactual $\leftarrow$ Enuevo
        sino
            con probabilidad $e^{\Delta E/T}$: Eactual $\leftarrow$ Enuevo
        fin
    fin
    Disminuimos la temperatura
fin

Simulated Annealing



Simulated Annealing - Aplicación

- Adaptable a problemas de optimización combinatoria (configuración óptima de elementos) y continua (punto óptimo en un espacio N-dimensional)
- Indicado para problemas grandes en los que el óptimo esta rodeado de muchos óptimos locales
- Indicado para problemas en los que encontrar una heurística discriminante es difícil (una elección aleatoria es tan buena como otra cualquiera)
- Aplicaciones: TSP, Diseño de circuitos VLSI
- Problemas: Determinar los valores de los parámetros requiere experimentación

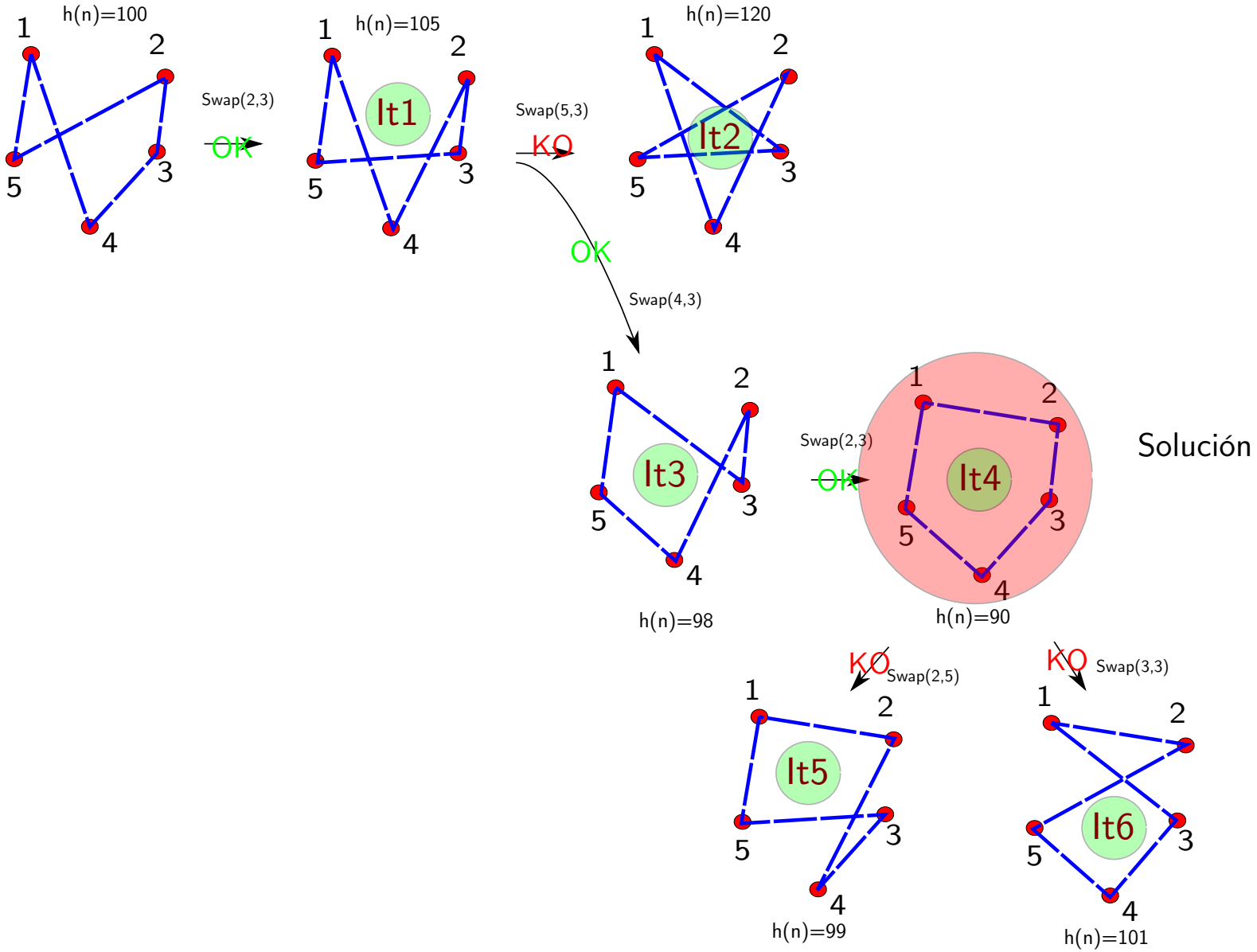
Simulated annealing - Ejemplo - TSP

- Viajante de comercio (TSP): Espacio de búsqueda $N!$
- Definimos posibles transformaciones de una solución (operadores): Inversiones, traslaciones, intercambios
- Definimos la función de energía (Suma de distancia entre ciudades, según el orden de la solución)

$$E = \sum_{i=1}^{n} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2} + \sqrt{(x_N - x_1)^2 + (y_N - y_1)^2}$$

- Definimos una temperatura inicial (experimentación)
- Determinamos cuantas iteraciones hacemos para cada temperatura y como disminuimos la temperatura

Simulated annealing - Ejemplo - TSP



Algoritmos Genéticos

- Inspirado en el mecanismo de selección natural
  - Los seres vivos se adaptan al entorno gracias a las características heredadas de sus progenitores
  - Las posibilidades de supervivencia y reproducción son proporcionales a la bondad de esas características
  - La combinación de buenos individuos puede dar lugar a individuos mejor adaptados
- Podemos trasladar la analogía a la búsqueda local
  - Las soluciones corresponden a individuos
  - La función de calidad indica la bondad de la solución
  - Combinando buenas soluciones podemos obtener soluciones mejores

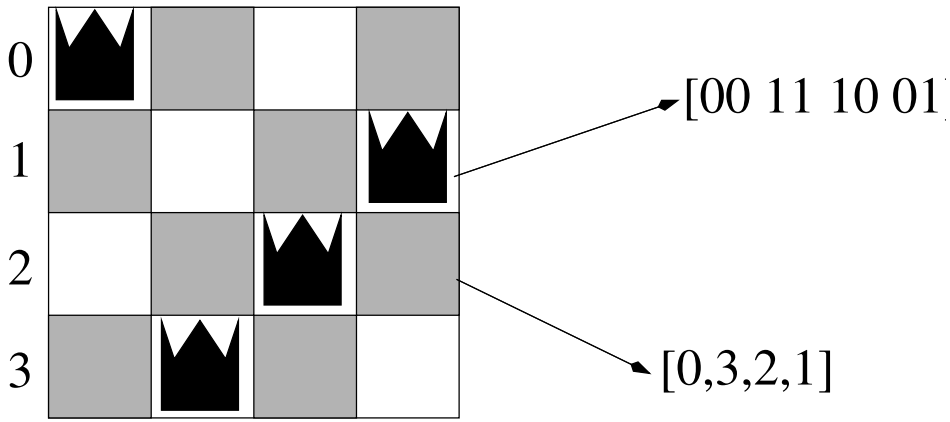
Algoritmos Genéticos (II)

Resolver un problema mediante AAGG requiere:

- Dar una codificación a las características de las soluciones (p ej: una cadena binaria)
- Tener una función que mida la calidad de la solución (función de fitness)
- Disponer de operadores que combinen las soluciones para obtener nuevas soluciones (operadores de crossover)
- Decidir el número de individuos inicial
- Decidir una estrategia para hacer la combinación de individuos

Algoritmos Genéticos - Codificación

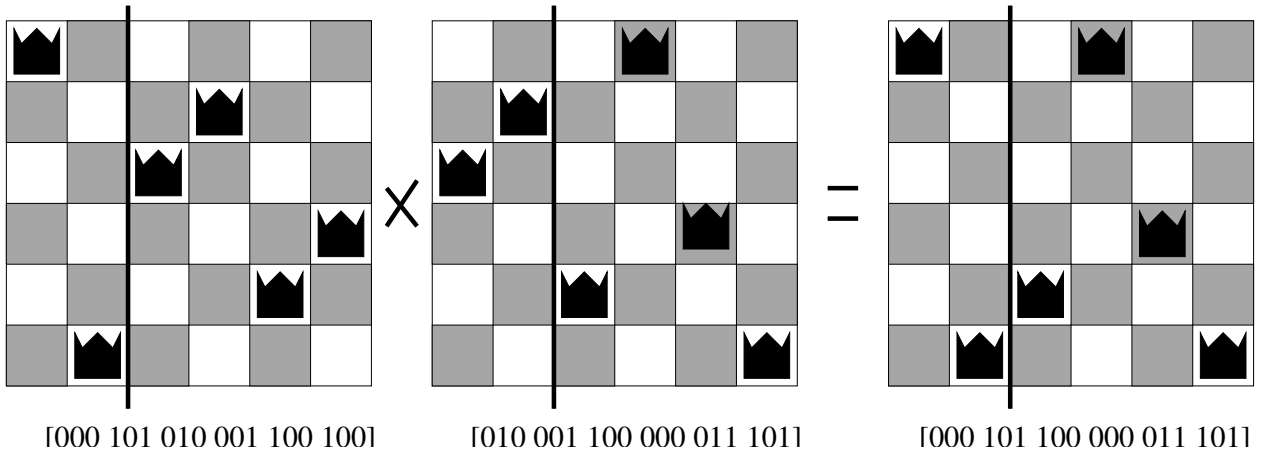
- Habitualmente la codificación de individuos es una cadena binaria (no tiene por que ser la mas adecuada)



- La codificación define el tamaño del espacio de búsqueda y el tipo de operadores de combinación necesarios

Algoritmos Genéticos - Operadores

- La combinación de individuos se realiza mediante operadores de cruce
- El operador básico es el cruce por un punto
  - Se elige aleatoriamente un punto de la codificación
  - La información de dos individuos se combina usando ese punto como referencia



Algoritmos Genéticos - Operadores (II)

- Existen otras posibilidades:
  - Cruce en dos puntos
  - Intercambio aleatorio de bits
  - Operadores adhoc según la representación
- Operadores de mutación:
  - Por analogía con la combinación de genes, a veces la información de parte de ellos cambia aleatoriamente
  - El operador básico de mutación consiste en cambiar el signo de un bit con cierta probabilidad

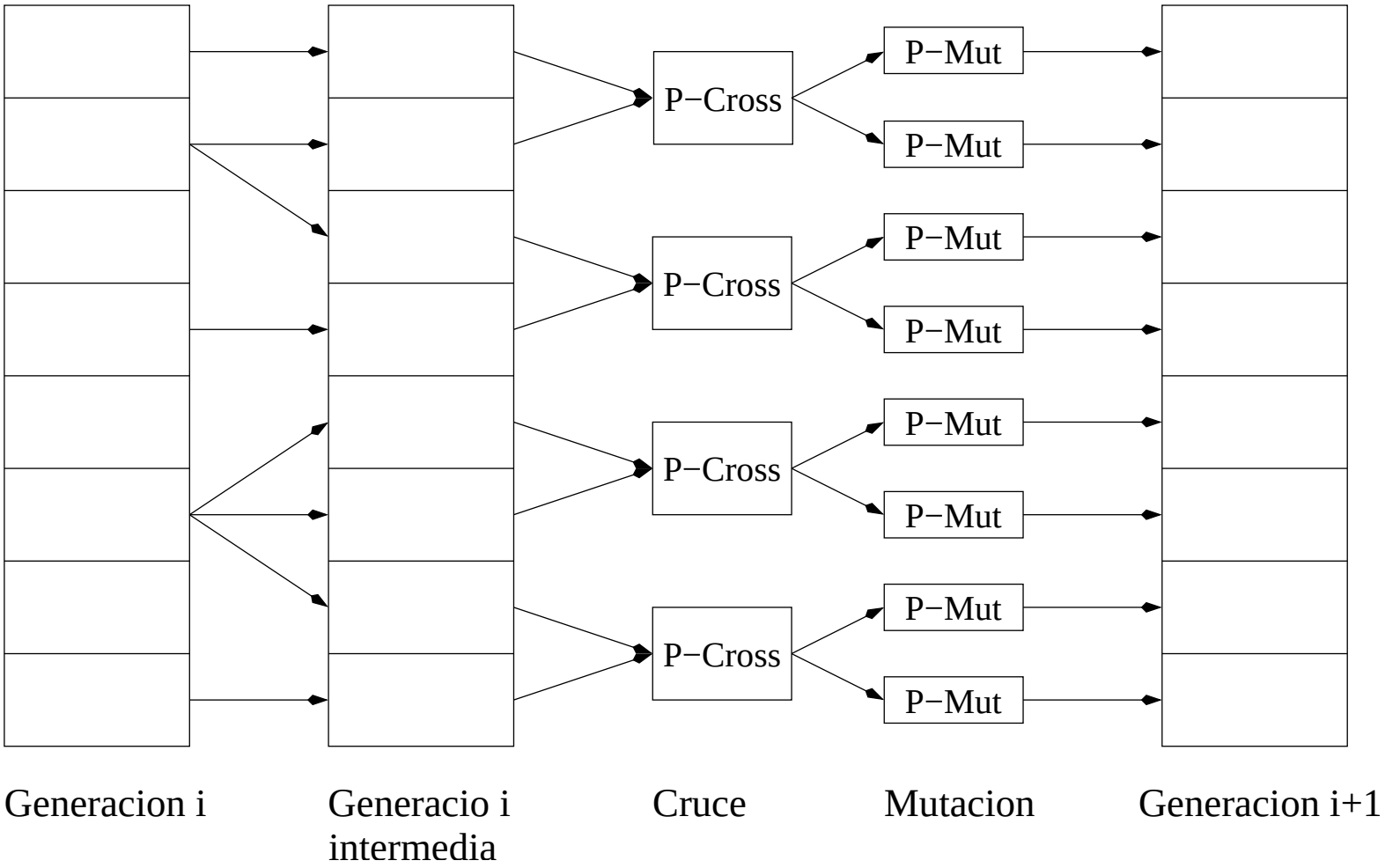
Algoritmos Genéticos - Combinación

- Cada paso de búsqueda es una generación de individuos, su tamaño se mantiene constante (N)
- Para pasar a la siguiente generación debemos elegir que individuos se han de combinar (generación intermedia)
- Elección de los individuos:
  - Cada individuo se elige con probabilidad proporcional a su calidad
  - Se establecen N torneos aleatorios entre parejas de individuos, se eligen los que ganan en cada torneo
  - Se define un ranking lineal entre individuos según su función de calidad
- Siempre habrá individuos que aparezcan mas de una vez e individuos que no aparezcan

Algoritmos Genéticos - Algoritmo canónico

- Los pasos que realiza el AG básico son estos:
  1. Se escogen N individuos de la generación actual para la generación intermedia (según el criterio escogido)
  2. Se emparejan los individuos y para cada pareja
     - Con una probabilidad (P_cruce) se aplica el operador de cruce a los individuos y se obtienen dos nuevos individuos
     - Con una probabilidad (P_mutación) se mutan los nuevos individuos
  3. Estos individuos forman la nueva generación
  4. Iterar hasta que la población converja o pase un número específico de iteraciones
- La probabilidad de cruce influirá en la variedad de la nueva generación
- La probabilidad de mutación siempre es muy pequeña para evitar una búsqueda aleatoria

Algoritmos Genéticos - Algoritmo canónico



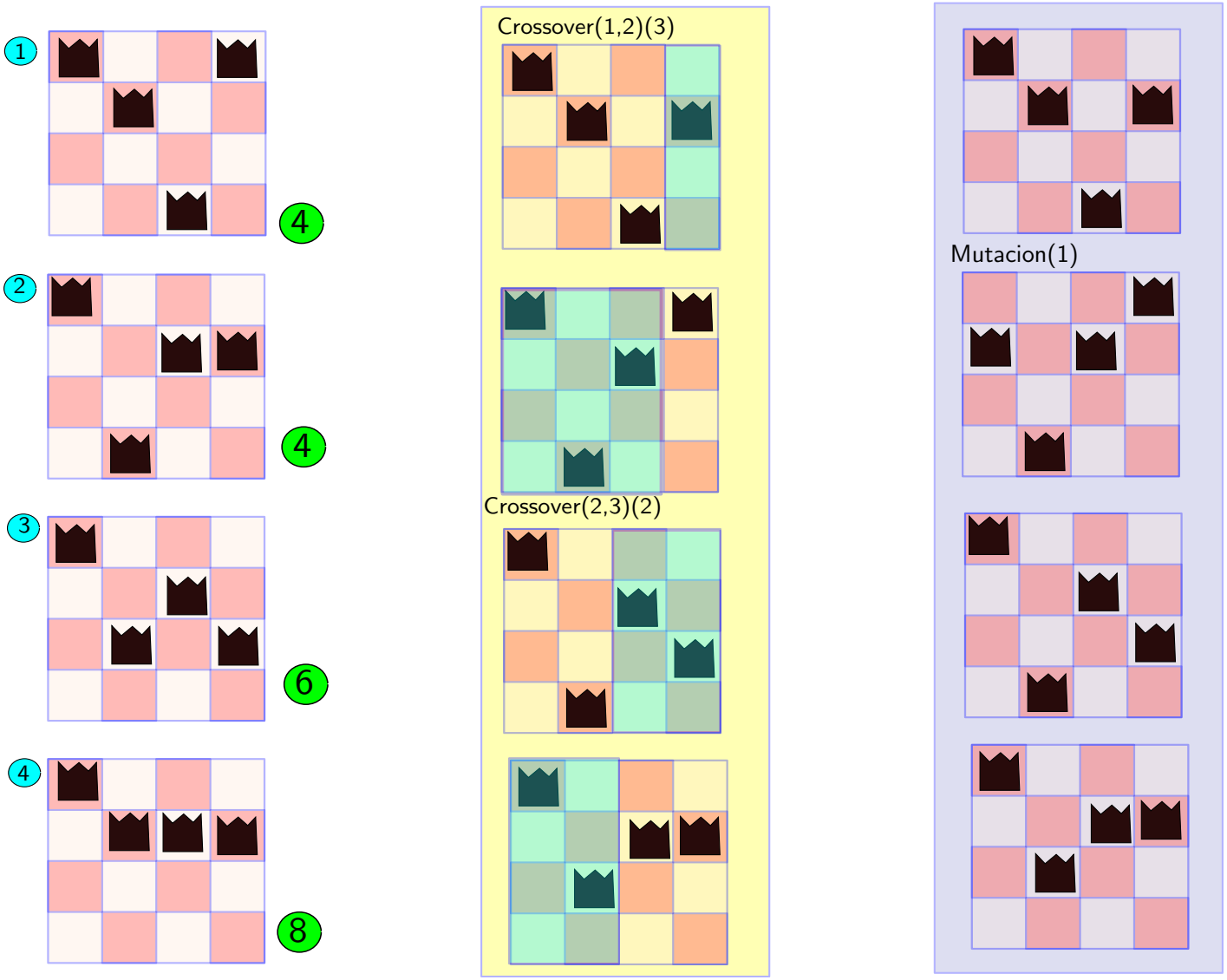
Algoritmos Genéticos - Aplicación

- Aplicable casi a cualquier tipo de problema
- Permite abordar problemas para los que no se dispone de una función heurística adecuada
- Por lo general serán peores que un algoritmo clásico con una buena heurística
- Aplicaciones: Incontables
- Problemas: Codificación de los estados, determinar los parámetros del algoritmo (tamaño de la población, iteraciones, probabilidad de cruce y mutación)
- En algunos tipos de problemas pueden no funcionar muy bien

Algoritmos Genéticos - Ejemplo - N reinas

- **Problema de las N reinas**
- Codificamos cada una de las posibles soluciones con un string binario
- Individuo= Concat(i=1...N; Binario(columna(reina$_i$)))
- Función de fitness= numero de parejas de reinas que se matan entre si
- Operador de cruce= Cruce en un punto
- Selección de la generación intermedia: Proporcional a la función de fitness
- Probabilidad de cruce $\rightarrow$ ¡experimentar!
- Probabilidad de mutación: $\rightarrow$ ¡experimentar!
- Tamaño población inicial: ¿aleatoria? (espacio de búsqueda n^n)

Algoritmos Genéticos - Ejemplo - N reinas



Algoritmos Genéticos - Ejemplo - N reinas