

Tema 1. Técnicas de IA. Búsqueda de soluciones

El problema de la búsqueda de soluciones

Satisfactibilidad vs. Optimalidad. Explosión combinatoria

Tipología de la Búsqueda

Búsqueda Global (sistemática). Búsqueda Local

Métodos Constructivos, Métodos de Mejora y Métodos Poblacionales

Búsqueda Heurística

Tipos. Algoritmos A y A*. Variantes

Metaheurísticas

Tipología de Metaheurísticas. Exploración vs. Explotación

Elección de una metaheurística. No free lunch

Bibliografía

- Monografía: Metaheurísticas. *Inteligencia Artificial, Vol 7, No 19 (journal.iberamia.org)* (2003).
 - Handbook of Metaheuristics, Springer 2^a ed. M. Gendreau, J.Y.Potvin, 2010
 - Essentials of Metaheuristics. Sean Luke. 2013. (Online)
 - Metaheuristics Network Website
-
- Inteligencia Artificial. Técnicas, métodos y aplicaciones. Palma, Marín. McGraw Hill (2008)
 - Inteligencia Artificial. Un enfoque moderno. S. Russell, P. Norvig. Prentice Hall 4^a ed
 - Computational Intelligence. A. Engelbrecht. Wiley & Sons. 2^aed. (2007)
 - How to Solve It: Modern Heuristics. Z. Michalewicz, D. Fogel. 2ed. 2004 (Springer)
-

Diversos Artículos

Journal of Heuristics, Int. Journal of Metaheuristics,
Int. Journal of Applied Metaheuristic Computing, Applied Intelligence, etc.



Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins



Artículos en Poliformat

A survey on optimization metaheuristics

Ilhem Boussaïd^a, Julien Lepagnot^b, Patrick Siarry^{b,*}

^aUniversité des sciences et de la technologie Houari Boumediene, Electrical Engineering and Computer Science
16111 Algiers, Algeria

^bUniversité Paris Est Créteil, LISSI, 61 avenue du Général de Gaulle 94010 Créteil, France

Artificial Intelligence Review (2020) 53:753–810

<https://doi.org/10.1007/s10462-018-09676-2>

Artif Intell Rev (2019) 52:2191–2233
<https://doi.org/10.1007/s10462-017-9605-z>

Metaheuristic research: a comprehensive survey



Review

Review of Metaheuristics Inspired from the Animal Kingdom

From ants to whales: metaheuristics for all tastes

Artificial Intelligence Review (2020) 53:501–593

<https://doi.org/10.1007/s10462-018-9667-6>

A state of the art review of intelligent scheduling

No Free Lunch Theorems for Optimization

David H. Wolpert and William G. Macready

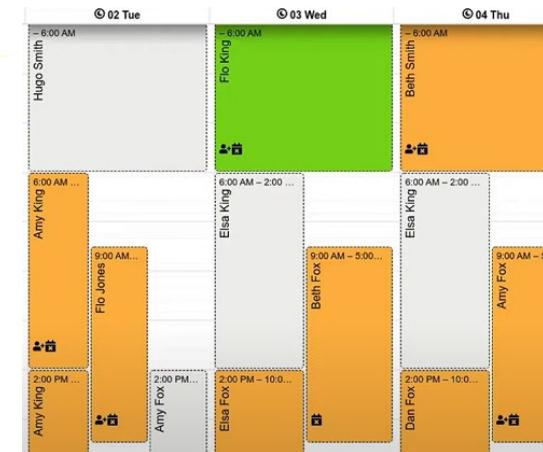
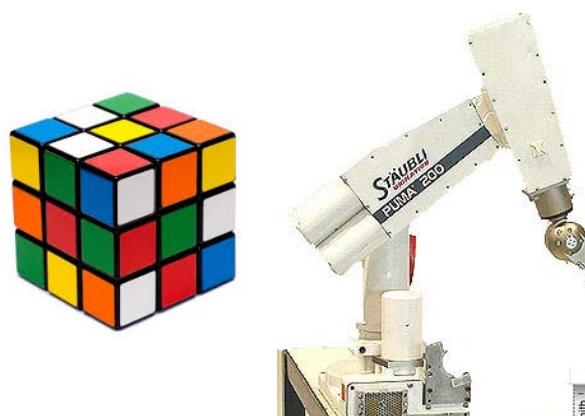
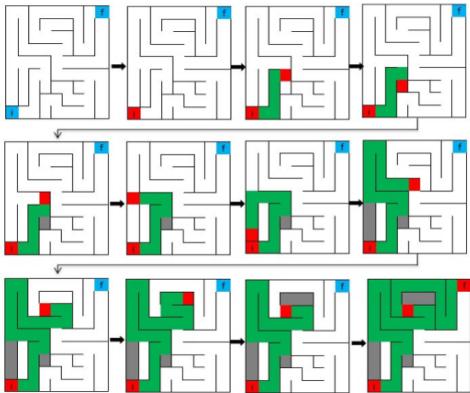
IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, VOL. 1, NO. 1, APRIL 1997

El problema de la búsqueda de soluciones

La búsqueda de soluciones es un método típico para la resolución de problemas en IA:

Toma de decisiones, Planificación (y robótica), Diseño y configuración, Respuestas a preguntas, Sistemas de recomendación, Juegos, Aprendizaje inductivo (en base a ejemplos), Generación de rutas, N-recubrimiento, Scheduling y asignación de recursos, etc.

La solución debe ser **obtenida** (generada y/o mejorada) mediante un proceso de búsqueda en un **amplio espacio de estados/soluciones**



5	3		7			
6		1	9	5		
	9	8			6	
8			6			3
4		8	3			1
7			2			6
	6			2	8	
		4	1	9		5
			8		7	9

El problema de la búsqueda de soluciones

Normalmente **no nos basta** con obtener una solución, sino que queremos la **mejor** (solución óptima)



Distintos enfoques de resolución

- **Métodos exactos (matemáticos, de investigación operativa o algorítmicos):** capaces de solucionar problemas complejos (de optimización) en tiempos computacionales bajos. Ej. algoritmo simplex, programación lineal (entera), ramificación y poda, programación dinámica, algoritmo A/A*, etc.

Pero hay problemas para los que **no existe o no es viable un método exacto**

- **Métodos aproximados (o de IA):** métodos que utilizan heurísticas para tratar de resolver el problema de forma más eficiente, aunque igual no siempre encuentran la solución exacta. Ej. búsqueda local, metaheurísticas, etc.

El problema de la búsqueda de soluciones

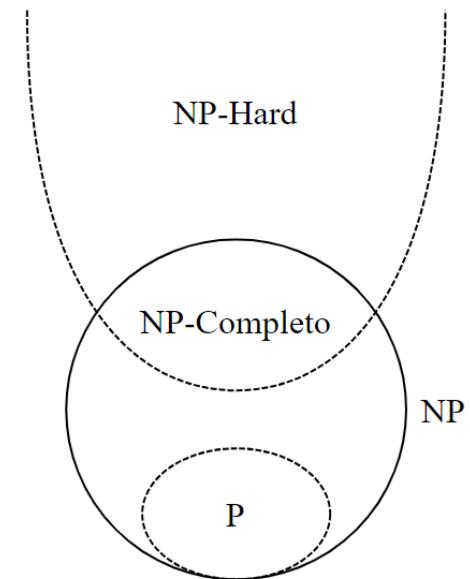


¿Y si queremos conseguir el valor 193 con el menor número de operandos u operadores?

Los problemas de optimización son **particularmente difíciles** de resolver

Satisfactibilidad: **NP**

Optimalidad: **NP-duro**



A menudo no se puede garantizar encontrar la solución óptima en tiempo computacional razonable

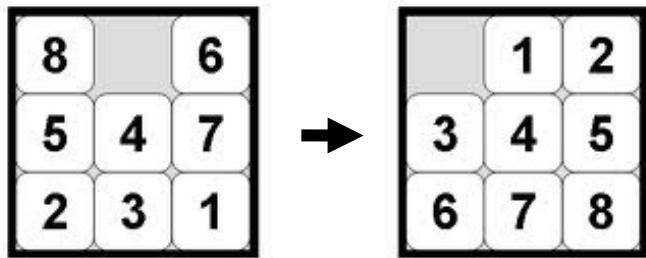
⇒ Objetivo: obtener una *buenasolución* (que tiende a la optimalidad), pero no necesariamente óptima

Satisfactibilidad vs. Optimalidad

Problemas de optimización sujetos a restricciones. Obtención de soluciones que:

- Cumplan los criterios de **factibilidad** (satisfagan las restricciones/condiciones del problema)
- Optimicen una determinada función objetivo (**maximizar/minimizar**)

8-puzzle. SATISFACTIBILIDAD



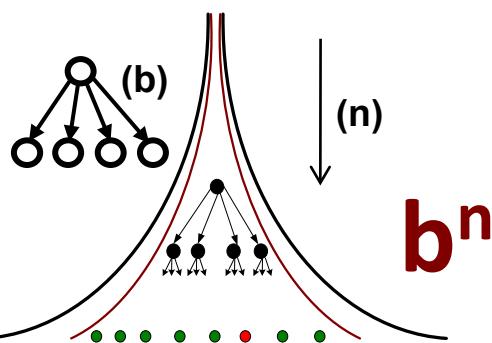
Difícil encontrar una solución (**problema de satisfactibilidad**), aunque también existen soluciones mejores que otras (**problema de optimalidad**)

Problema del viajante de comercio (TSP). OPTIMALIDAD



Es trivial encontrar soluciones factibles.
El problema es encontrar soluciones optimizadas (**problema de optimalidad**)

Explosión combinatoria en la búsqueda

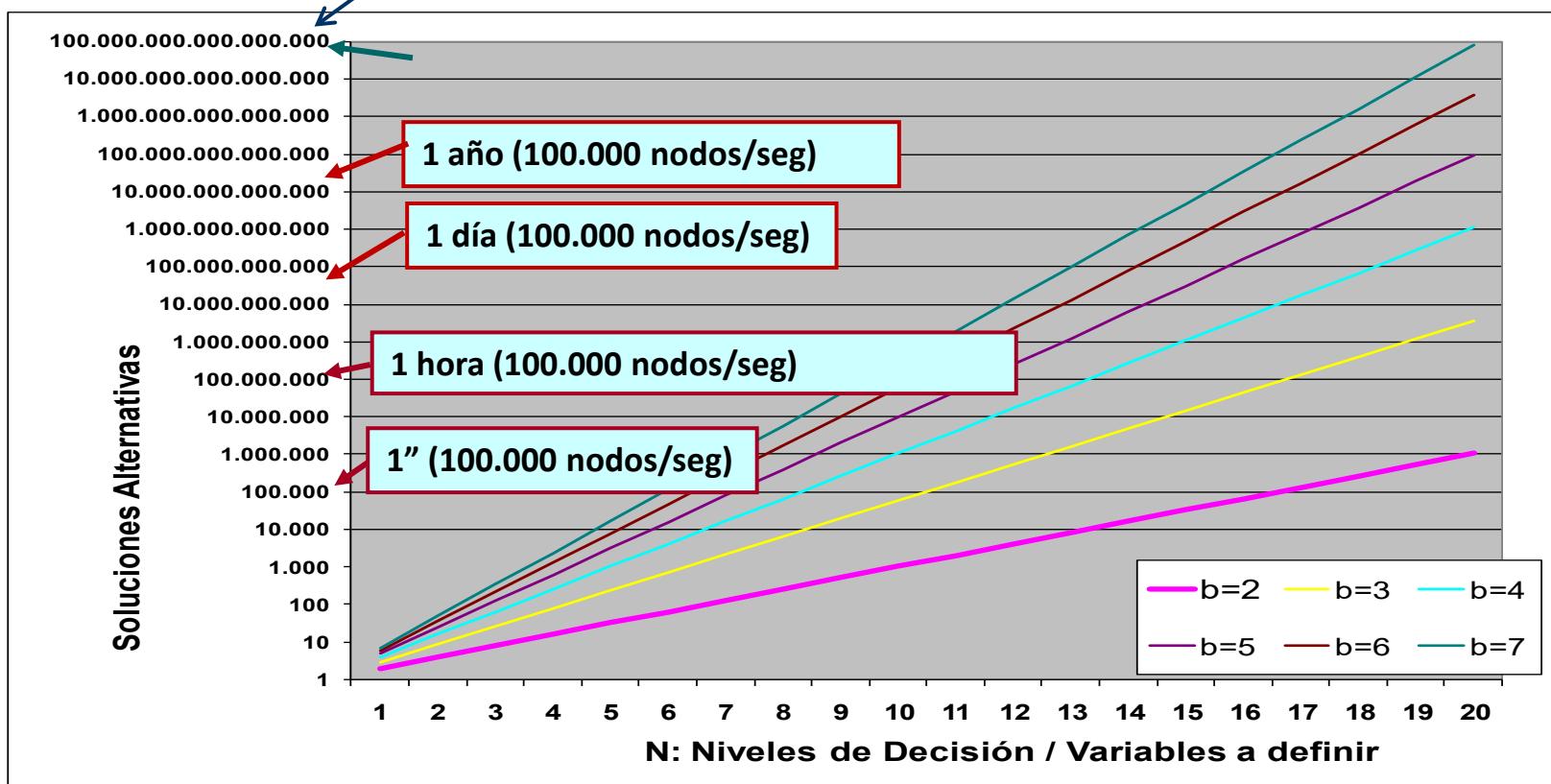


La explosión combinatoria depende del factor de ramificación (**b**) y los niveles de decisión/profundidad (**n**)

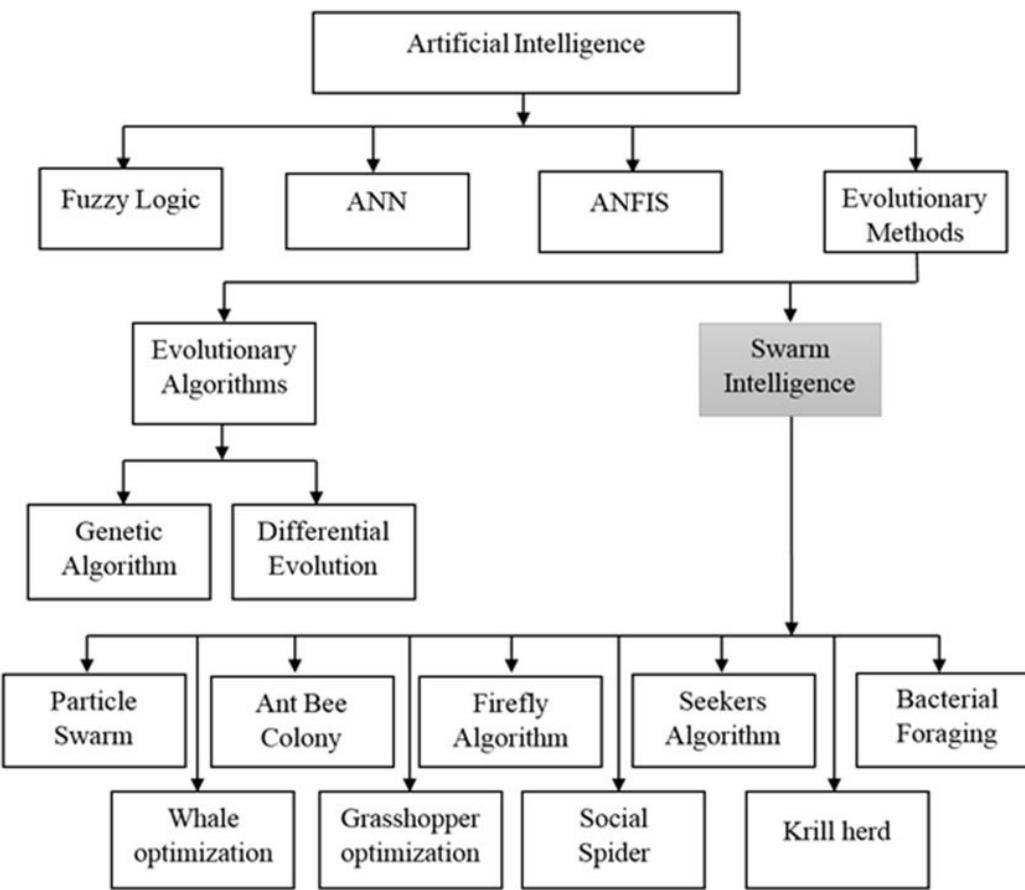
En dominios finitos (optimización combinatoria), el conjunto de soluciones es finito y numerable (pero muy grande)

Una fina hoja de papel (0.1 mm), doblada 50 veces, alcanza un espesor de...

Combinaciones Cubo de Rubik $\approx 10^{18}$ posibles estados

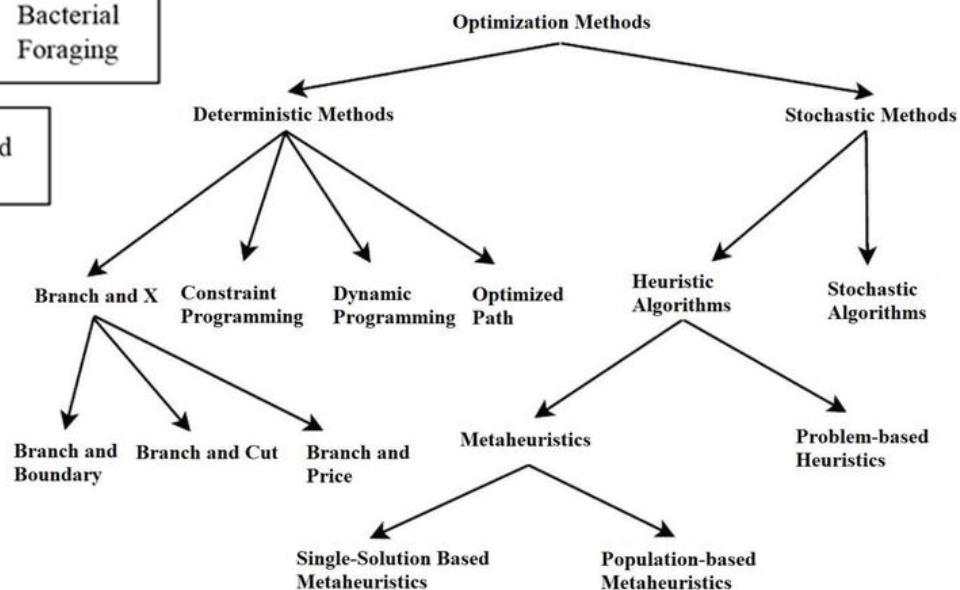


Múltiples clasificaciones de resolución en AI

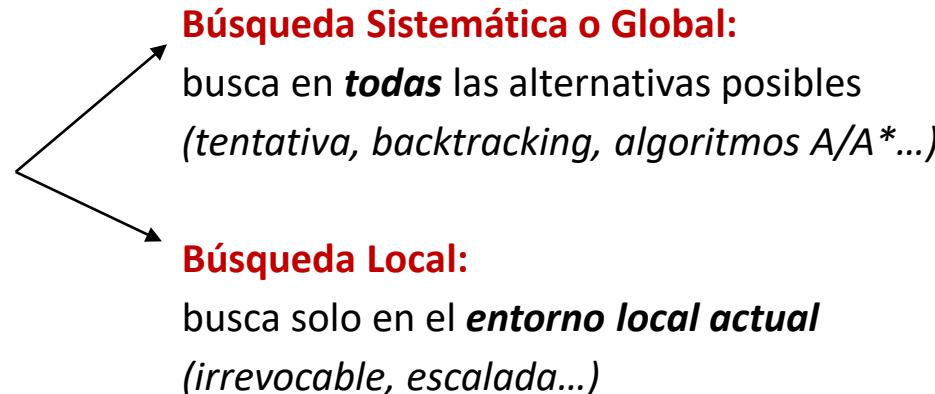


No es viable un método exacto que garantice encontrar la solución óptima a problemas realistas en tiempo computacional razonable

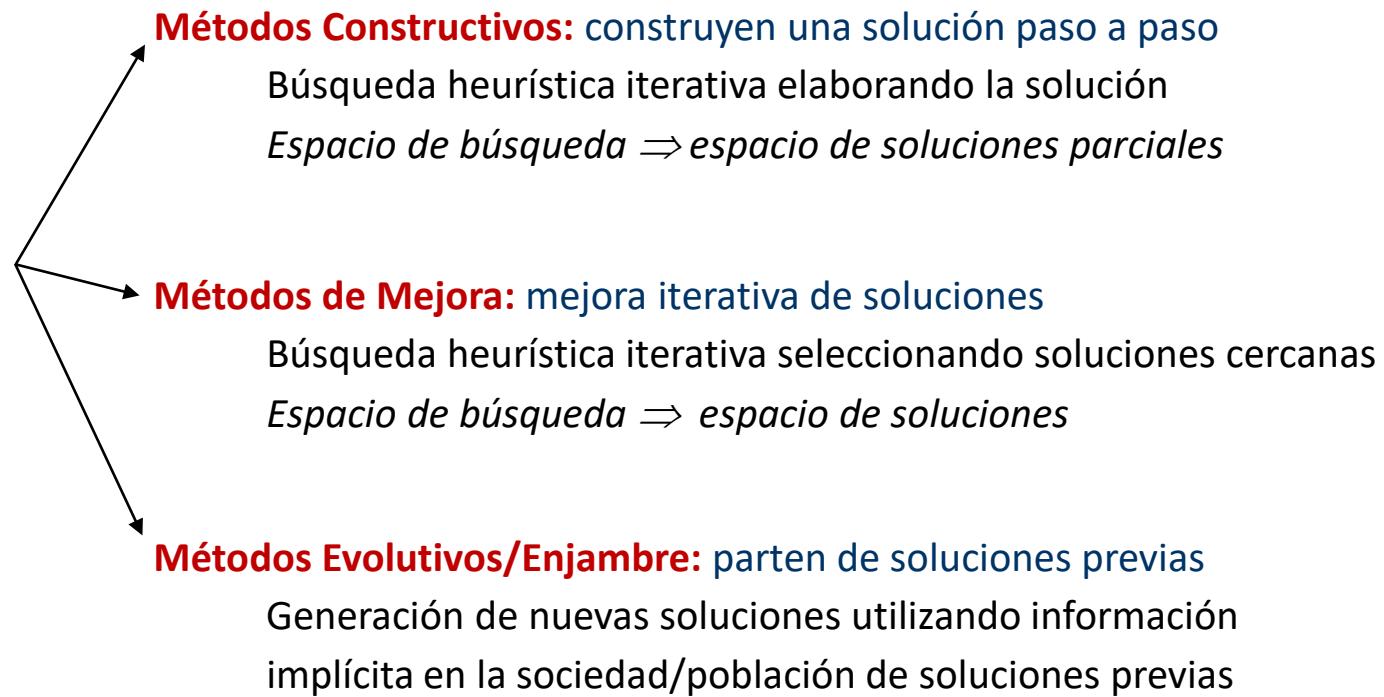
Uso de **métodos aproximados de IA** que devuelvan soluciones razonablemente buenas



Estrategias de Búsqueda (Global / Local)



Métodos (procesos) (Búsqueda en un Espacio de Estados / Soluciones)

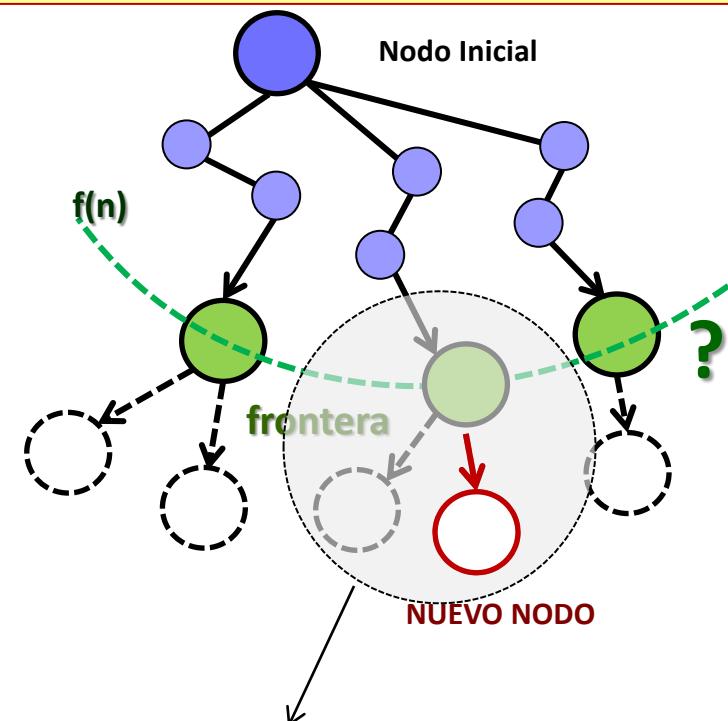


Estrategias de búsqueda. Criterio para seleccionar el siguiente estado

Estrategias de Búsqueda Global (*global search*)

Búsqueda **sistemática** del espacio de búsqueda: por **cualquier nodo (frontera) del espacio de búsqueda**

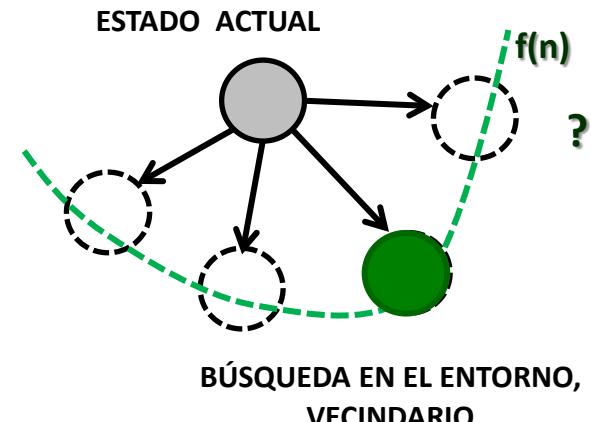
- ⇒ Alto coste en memoria (explosión combinatoria)
- Puede ser completa y admisible (garantizar solución óptima)
- Típicamente se usan para generar soluciones



Estrategias de Búsqueda Local (*local search*)

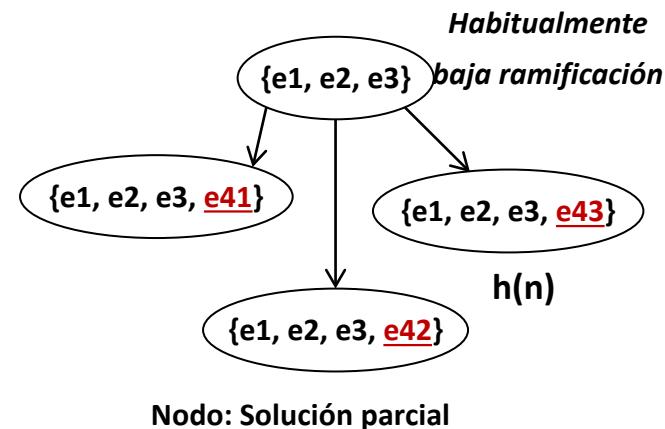
En cada iteración se busca solo en el **entorno del estado actual**

- Retienen solo el estado actual (≈ esquema irrevocable)
⇒ Eficientes en memoria
- Pueden utilizarse para **generar** una solución o **mejorar** una solución previa
- Requieren definición de **vecindario** de una solución
- **Inconvenientes:** pueden quedar atrapadas en soluciones que no admiten mejoras en su entorno (óptimos locales, valles, mesetas), problemas de incompletitud, ciclos, etc.



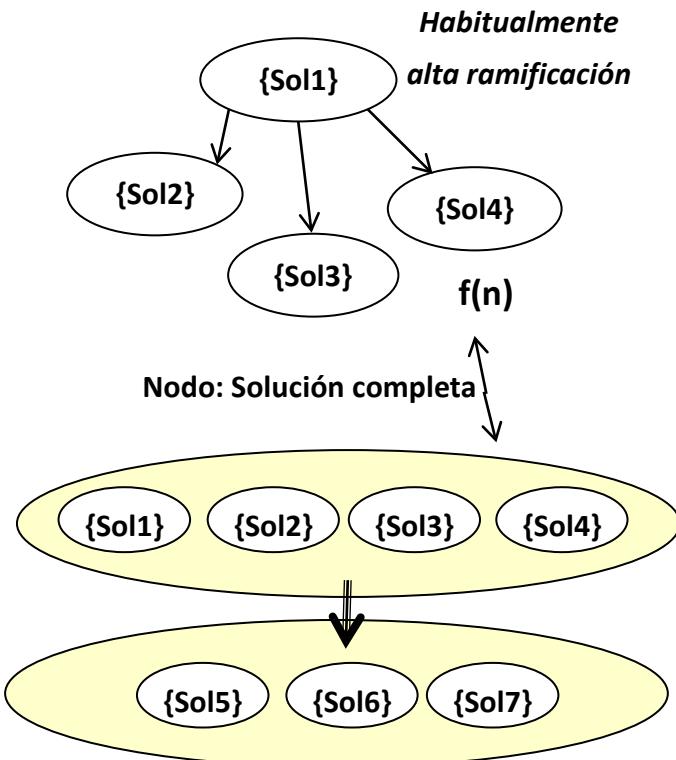
Métodos Constructivos

- Construyen iterativamente una solución al problema
- En cada paso se añade (heurísticamente) un elemento a la solución parcial
- No se requiere solución inicial: de aplicación en problemas en donde lo difícil es obtener una solución (problemas con muchas restricciones)
- Puede aplicarse con una búsqueda local o búsqueda global



Métodos de Mejora (habitualmente llamados Búsqueda Local)

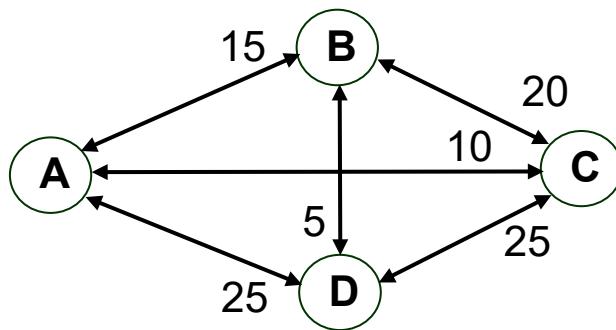
- Requieren una solución inicial (constructiva / aleatoria)
- Cada iteración modifica solución actual (eliendo heurísticamente en su entorno) intentando mejorarla
- Ramificación muy alta \Rightarrow búsqueda local
- Ventaja *any-time*



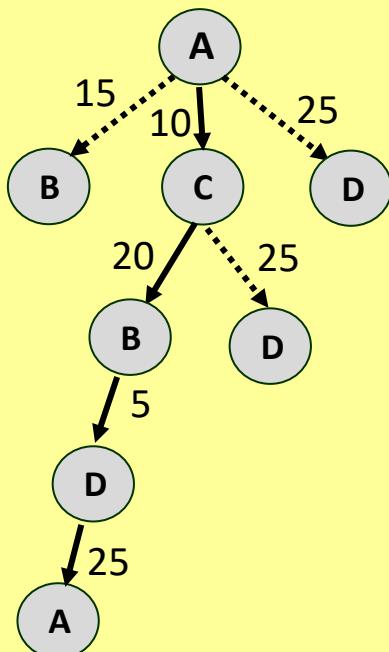
Métodos Poblacionales / Evolutivos / Enjambre

- Parten de un conjunto (**población/enjambre**) de soluciones, que se seleccionan, recombinan, modifican o colaboran para obtener un nuevo conjunto de soluciones
- Ventaja *any-time*

Ejemplo. Problema del viajante

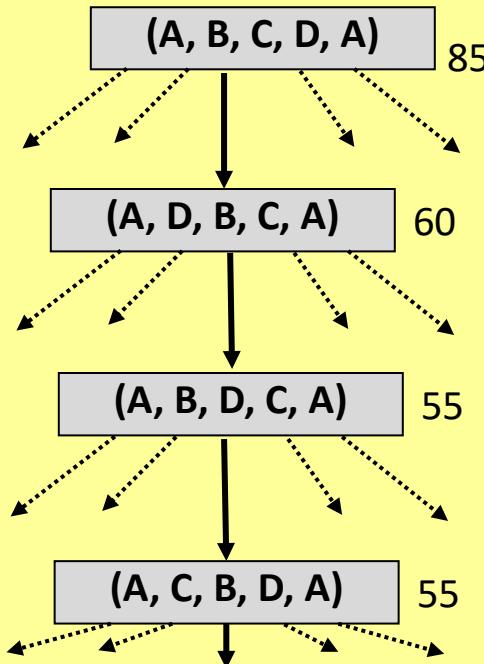


Método Constructivo



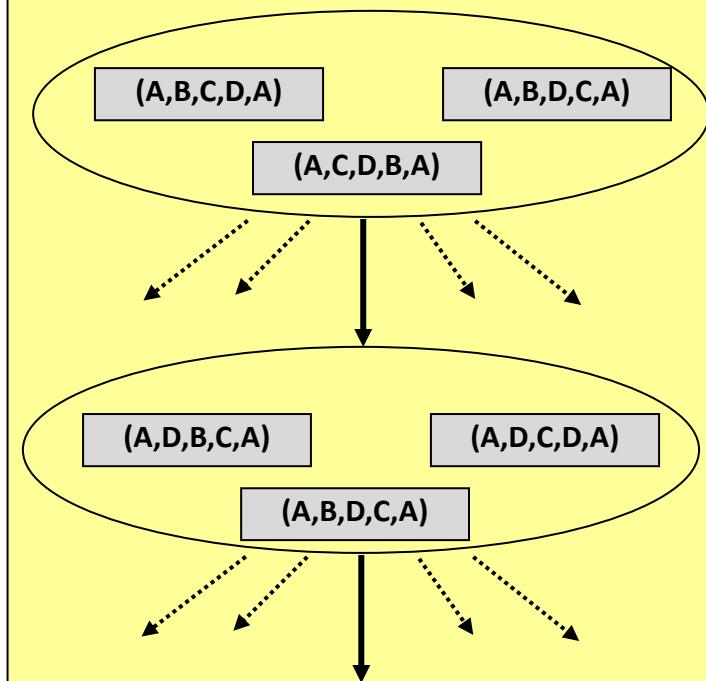
Branching: nuevos elementos

Método de Mejora



Branching: soluciones alternativas

Método Evolutivo

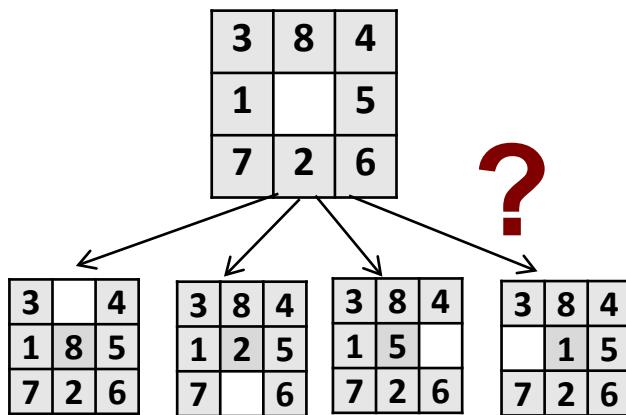


Branching: soluciones alternativas

Búsqueda heurística en IA (heuriskein / ευρισκειν: encontrar o descubrir)

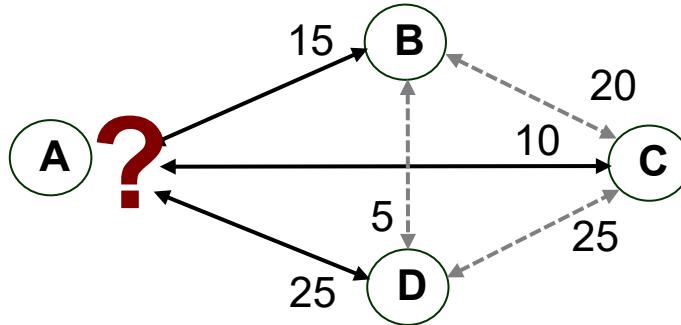
Criterios aplicados a la resolución de problemas complejos, esperando que obtengan una buena solución (no necesariamente óptima) de un modo sencillo y rápido

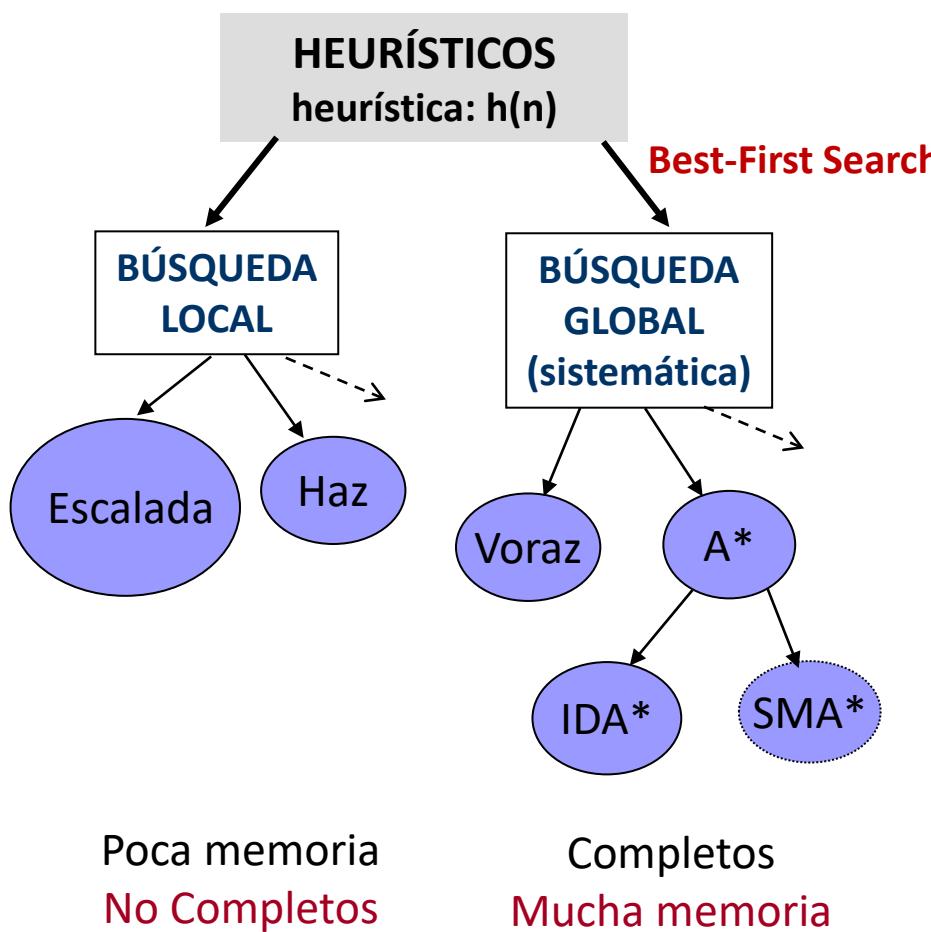
Métodos aproximados, contrapuestos a los exactos. No son resultado de un riguroso análisis formal, sino de **conocimiento intuitivo**, experimental, práctico, de experto, etc. sobre el problema



Pueden ser generales o dependientes del problema (más eficientes, menos generalizables)

- ✓ Relajación (relajación de condiciones/restricciones del problema)
- ✓ Inductivas (generalización de soluciones a versiones sencillas del problema)
- ✓ Descomposición (descomposición en subproblemas más sencillos de resolver)
- ✓ Reducción (identificar propiedades que se cumplen mayoritariamente en las buenas soluciones e introducirlas como restricciones del problema)
- ✓ Abstracción (abstracción de estados distintos del problema en un único estado)
- ✓ Landmarks (identificación de estados intermedios que deben ser alcanzados)





Búsqueda Heurística

Se usa una función $f(n)$, que mide la bondad del nodo (estado) y tiene una componente **heurística $h(n)$** :

$$h(n) = \text{estimación del coste al objetivo}$$
$$\forall n, h(n) \geq 0, h(\text{meta}) = 0$$

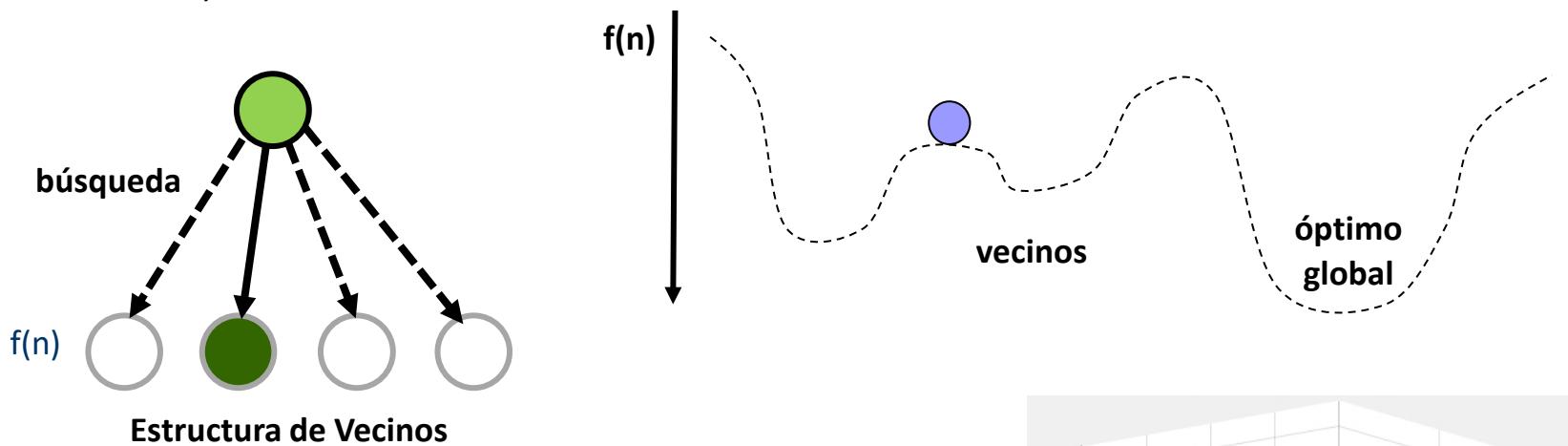
Según las Estrategias:

- Búsqueda Local (Escalada, Haz...):
 - Requieren muy poca memoria
 - Pero tiene **problemas de valles, ciclos, etc.** y **no garantiza la óptima**
- Búsqueda Global (en grafo/arbol):
 - Primero el mejor (*Best First Search*): expande el mejor nodo frontera
 - Permite soluciones óptimas, pero suele requerir mucha memoria

Búsqueda local (no sistemática)

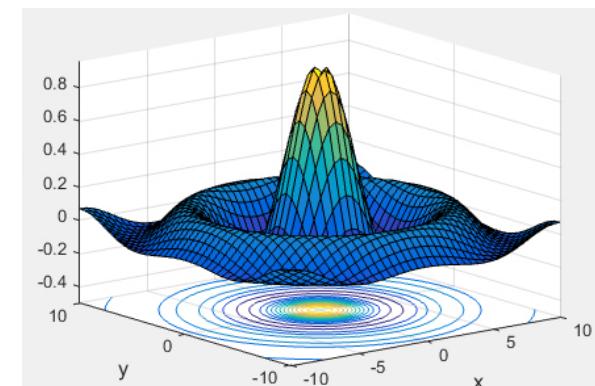
Función de Escalada (*hill-climbing*), Descenso por gradiente, Voraz local, etc.

- Selección de un sucesor (vecino) que mejore $f(n)$ respecto al padre: escalada por máximos locales
- Local: no mantiene un árbol de búsqueda, sino solo el estado actual (y sucesores inmediatos)
- Realiza unos bucles de búsqueda dirigido hacia el crecimiento de $f(n)$ (colina arriba, máximo gradiente). El proceso acaba cuando no hay mejora posible en el conjunto de soluciones vecinas (valles, llanuras, ciclos, etc.).



Ventaja: memoria limitada (\cong constante), no necesita mantener caminos alternativos

Inconvenientes: óptimos locales (no garantiza óptimo global, valles, mesetas, ciclos, etc.)



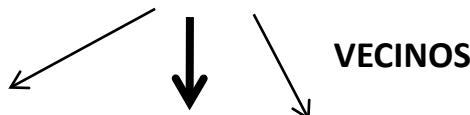
Ejemplo búsqueda local. 8-puzzle

3	8	4
1		5
7	2	6

INICIO

1	2	3
8		4
7	6	5

OBJETIVO



3		4
1	8	5
7	2	6

Hill climbing algorithm

1. $n = \text{initial node}$
2. $\text{while } n \neq \text{goal}$
 - if $\text{Sucessors}(n) = \emptyset$, finish with failure
 - Select $n' \in \text{Sucessors}(n)$ to minimize $f(n')$
 - if $f(n) < f(n')$, then finish with failure
 - else $n = n'$



	3	4
1	8	5
7	2	6

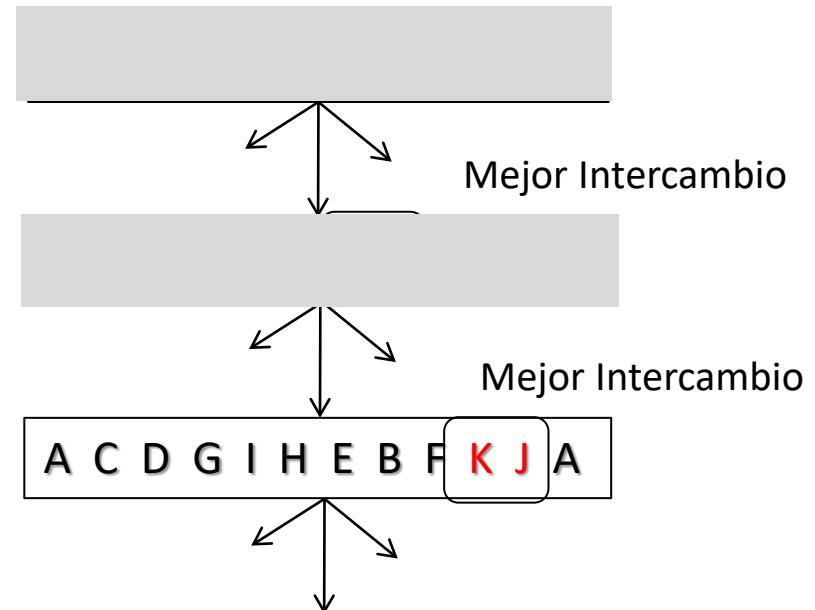
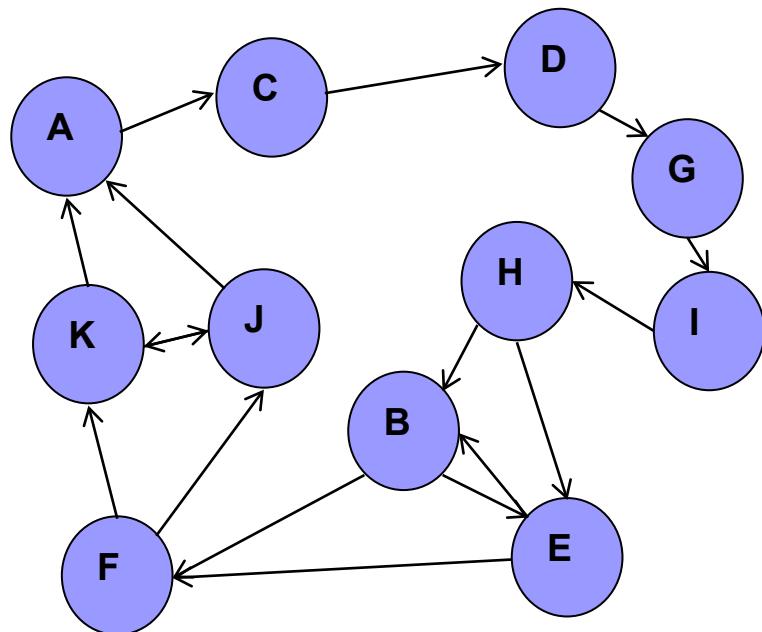
1	3	4
	8	5
7	2	6

1	3	4
8		5
7	2	6

VECINOS

VECINOS

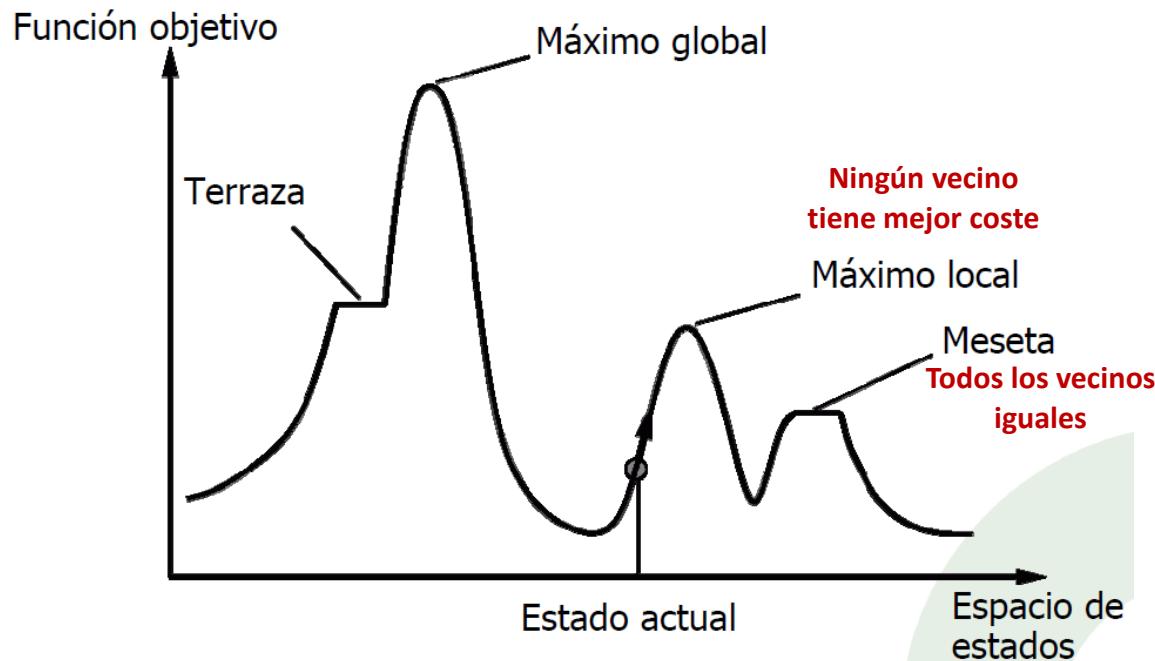
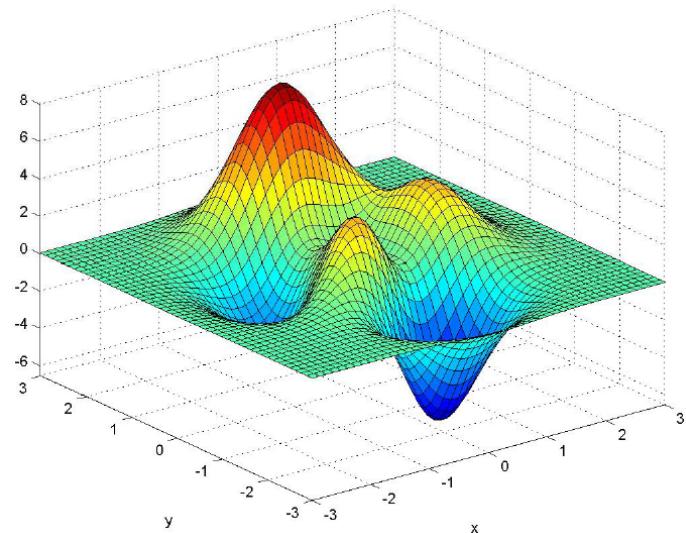
Vecinos implica una **permutación** en el orden de los nodos (intercambio de pareja)



Ningún intercambio mejora:
Parada Máximo Local

Inconvenientes:

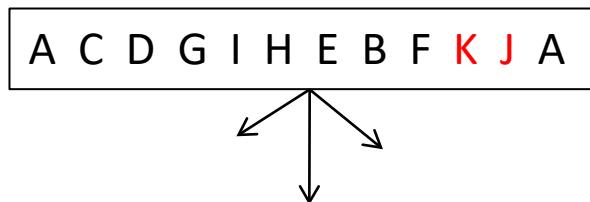
- No hay memoria en la búsqueda, no hay visión global
- Búsqueda puede quedar atascada (pararse) en óptimo local (todos los vecinos son peores), llanuras (todos los vecinos son iguales), bucles (debido a no almacenar en memoria los nodos visitados)
- Se obtienen soluciones localmente óptimas, pero pueden estar muy lejos del óptimo global



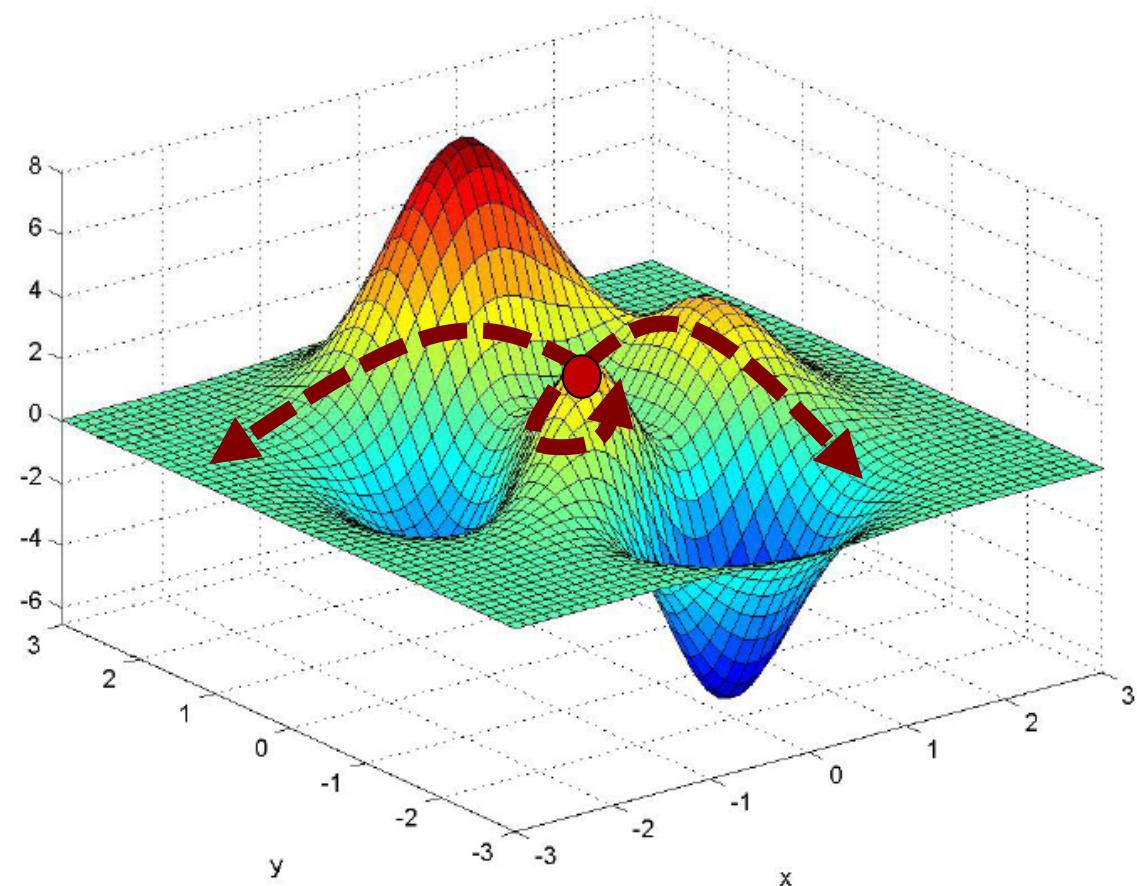
Después de escalar una gran colina, uno se encuentra solo con que hay muchas más colinas que escalar. Nelson Mandela

Posibles Soluciones: escape de un óptimo local, en cuyo entorno no existen mejores valores que el actual

- Salto a otra zona de búsqueda (previamente guardada) \approx (Búsqueda en Haz o Beam Search)
- Avanzar al siguiente nivel, o a más niveles (aunque puedan ser peores)
- Re-arranque \approx Random Restarting



Varias opciones si ningún intercambio mejora



Búsqueda global (sistemática)

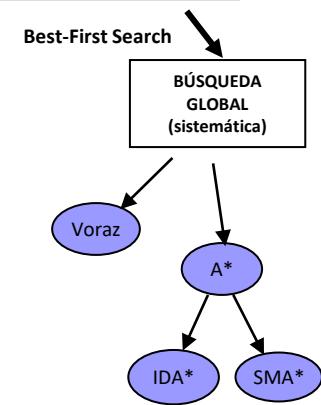
Mantienen abierta la frontera de búsqueda: árbol/grafo de búsqueda

HEURÍSTICOS
heurística: $f(n)$

Búsqueda Voraz, A y A* son ‘Búsquedas Primero el Mejor’, Sistémicas

Búsqueda Voraz (Greedy)

- Caso típico de “búsqueda el primero mejor”, donde no importa $g(n)$:
$$f(n) = h(n)$$
- No siempre encuentran solución o acaban, no son completos ni son admisibles

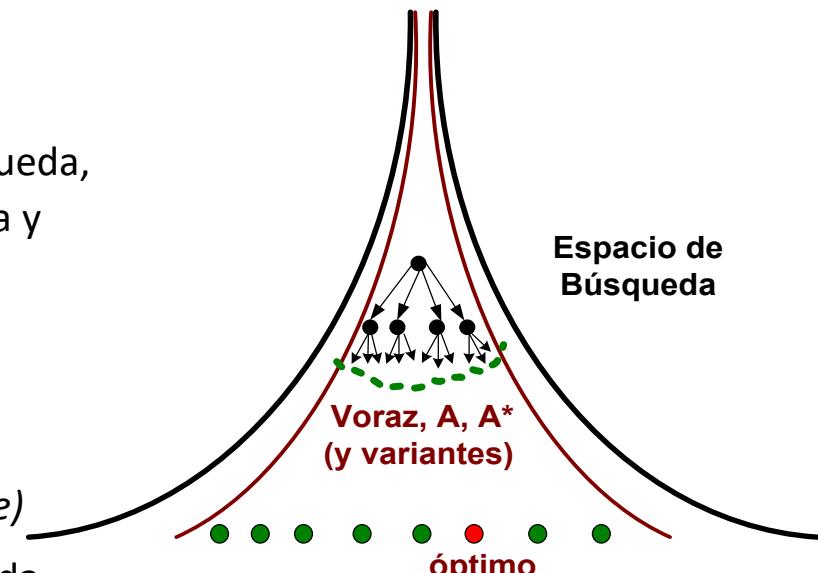


Algoritmo A

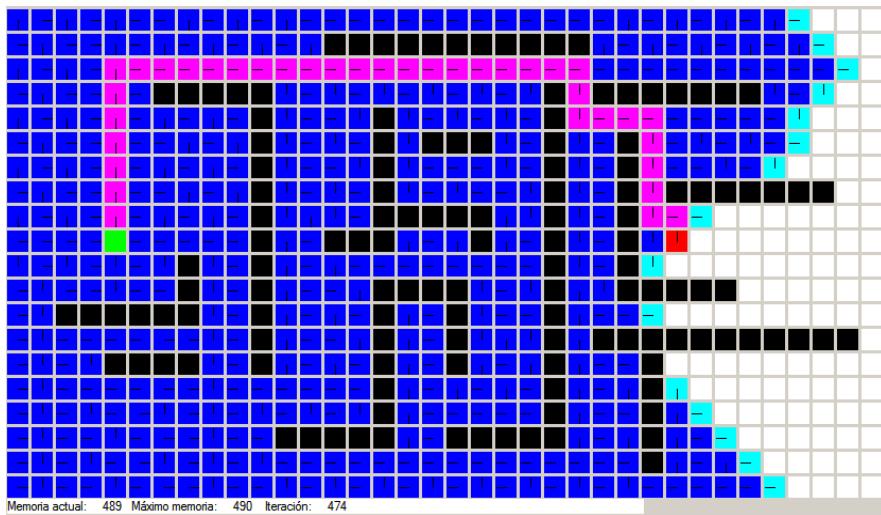
- Combinación de **búsqueda voraz** (reduce coste de búsqueda, pero no óptima ni completa) y **coste uniforme** (completa y óptima, pero ineficiente)

$$f(n) = g(n) + h(n)$$

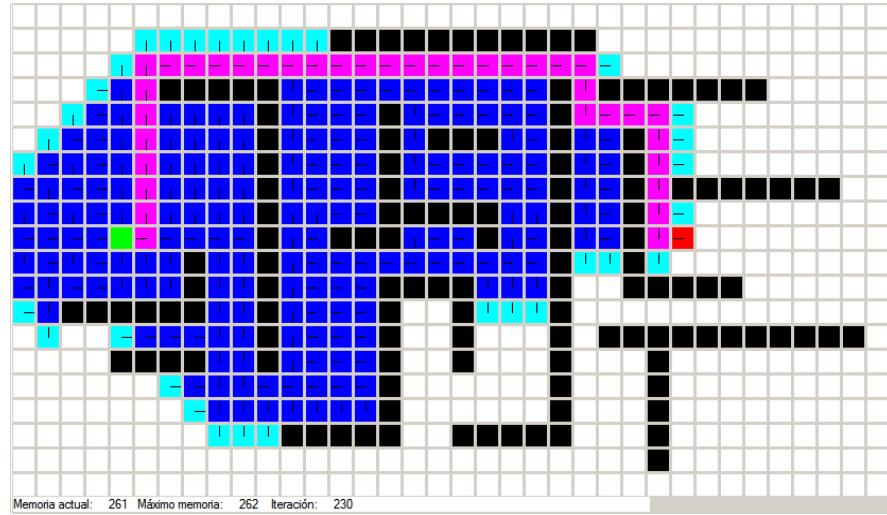
- **Algoritmo A***: $\forall n, h(n) \leq h^*(n)$
- **Completa y admisible.** ($h(n)=0 \cong$ Dijkstra, coste uniforme)
- **Pero requiere mucha memoria** (un algoritmo que expanda menos nodos que A* no garantizará la admisibilidad)



Búsqueda A (h=0)



Búsqueda A (h(n): Manhattan)



Además del tiempo computacional, **hay un problema de memoria**

Variantes Algoritmo A:

Weighted A*

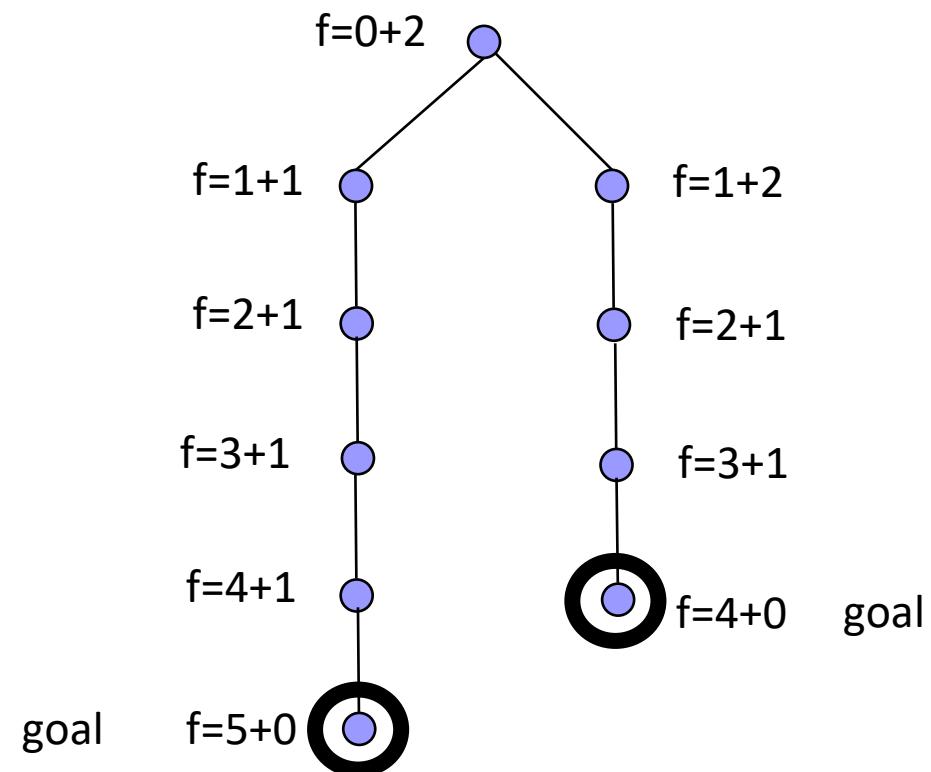
- $f(n) = g(n) + a * h(n)$, donde $a > 1$
- Es una búsqueda completa, pero no admisible
- Se garantiza encontrar una solución menor que 'a' veces el coste de la solución óptima

Iterative-Deepening A* (IDA*)

- Adapta la Búsqueda en Profundidad Iterativa (BPI) a A*
- El **criterio de corte** no es el nivel (profundidad), sino el valor de $f(n) = g(n) + h(n)$
- Cada interacción se reinicia desde la raíz, con un valor de corte: **coste ($g+h$) más pequeño de los nodos que fueron cortados en la iteración anterior**

En cada iteración, cuando el valor $f(n)$ de un nodo supera el valor de corte, se aplica backtracking

- Si $h(n) \leq h^*(n)$ IDA* es admisible
- Útil para problemas de costes $g(n)$ unitarios
- IDA* es completo y óptimo y necesita **menos memoria** que A* (es un iterativo en profundidad), aunque puede generar más nodos en total.
- **Complejidad Temporal:** $O(b^{2d})$
- **Complejidad Espacial:** $O(b.d)$



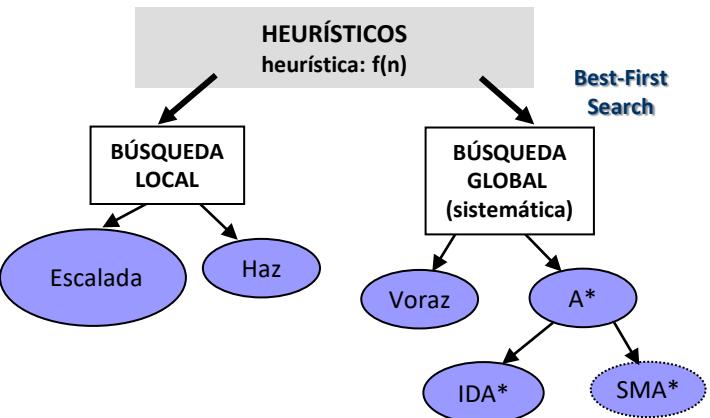
Búsqueda con memoria acotada: determina un **máximo número de nodos a almacenar** (N_{max})

Modificación del algoritmo A*, de forma que cuando no puede almacenar más nodos, se elimina de la frontera del grafo (lista open) el nodo que tiene el mayor valor de $f(n)$: ‘nodo olvidado’

- En cada nodo se recuerda el hijo ‘olvidado’ con mejor $f(n)$
- El algoritmo puede ‘recuperar’ los nodos olvidados si resultan más prometedores conforme avance la búsqueda (el resto de nodos tiene mayor $f(n)$)

Características

- Es capaz de adaptarse a la memoria disponible
- Es completo, si la memoria disponible es suficiente para almacenar la senda más corta
- Mejor para costes $g(n)$ no unitarios
- Es admisible, si la memoria disponible es suficiente para un camino óptimo
- En otro caso, devuelve la mejor solución que puede alcanzar con la memoria disponible



En resumen:

Métodos exactos (no heurísticos)	Inviáveis en problemas complejos
Búsqueda heurística local	No Admisibles, Poca memoria Problemas de ciclos, etc.
Búsqueda Heurística Global: Algoritmo A, A*	Búsqueda Global, sistemática Admisibles y Completos. Alto coste memoria / tiempo
Incrementar potencia heurística $h(n) > h^*(n)$ o Variantes Algoritmo A*	¿Pierden Admisibilidad? Bajan / Acotan coste memoria Siguen requiriendo mucha memoria
Metaheurísticos: Diseño y estrategias heurísticas	Soluciones Optimizadas Bajo coste temporal

Introducidos en “Future paths for integer programming and links to artificial intelligence” F. Glover, 1986

La idea básica es mejorar la búsqueda local

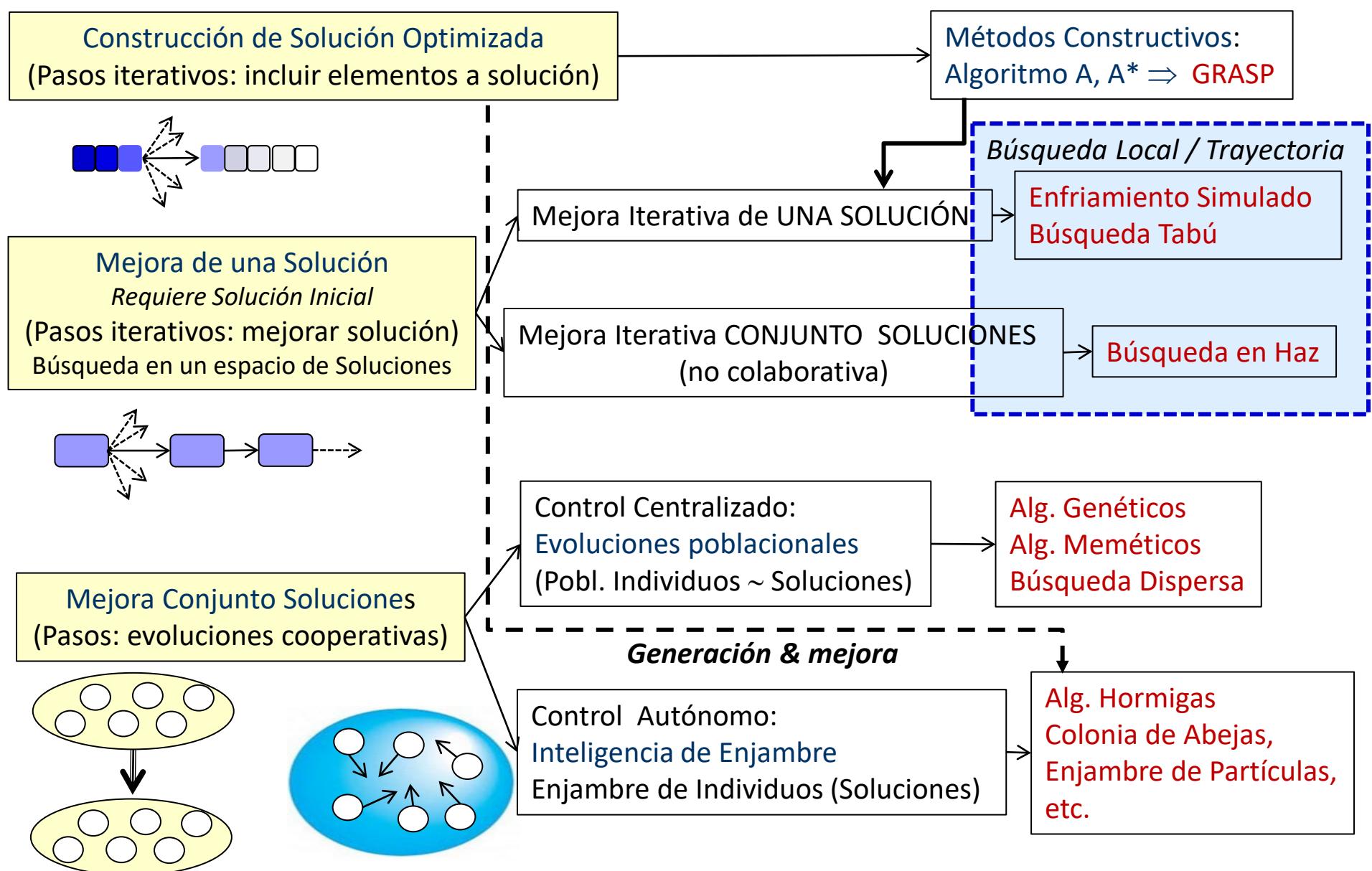
- se sitúan “por encima” de los algoritmos heurísticos: procedimientos iterativos, que modifican (modulan) las decisiones durante el proceso de búsqueda para producir eficientemente soluciones de alta calidad
- modulan la aplicación de la heurística mediante componentes de decisión estocásticos y/o métodos bio-inspirados
- incorporan simultáneamente ventajas de la búsqueda global, minimizando la probabilidad de quedar atrapados en óptimos locales
- proporcionan un marco general para crear nuevos algoritmos híbridos combinando diferentes conceptos derivados de la IA, ej. comportamientos sociales la evolución biológica y mecanismos estadísticos

- ✓ Utilizan poca memoria. Suelen ser fácilmente paralelizables
- ✓ Rápidas respuestas iniciales, que se continúan optimizando (*any time*)
- ✓ Mejoran la búsqueda local, sin la complejidad espacial/temporal de una búsqueda global
- ✓ Implementación sencilla. Dificultad: procesos no determinísticos que requieren muchos ajustes

Metaheurísticas. Orígenes

- **Enfriamiento simulado:** Kirkpatrick, S.; Gelatt Jr., C.D.; Vecchi, M.P. (1983). "Optimization by Simulated Annealing". *Science* 220 (4598):671–680. DOI 10.1126/science.220.4598.671
- **Búsqueda tabú:** Fred Glover and C. McMillan (1986): "The general employee scheduling problem: an integration of MS and AI". *Computers and Operations Research*, 1986.
- **Búsqueda por Haz Local (Beam Search):** Reddy, D. Raj, (1977): "Speech Understanding Systems: A Summary of Results of the Five-Year Research Effort. Dept. Comp. Science", Carnegie Mellon Univ. 1977.
- **Algoritmos genéticos:** John H. Holland (1975): "Adaptation in Natural and Artificial Systems". University of Michigan Press, 1975
- **Algoritmos Mémeticos:** Moscato P., (1989) "On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms". Caltech Concurrent Computation Program (report 826).
- **Scatter search:** Fred Glover (1977): "Heuristics for Integer programming Using Surrogate Constraints". *Decision Sciences* 8 (1): 156–166, 1977.
- **GRASP:** T.A. Feo and M.G.C. Resende (1989): "A probabilistic heuristic for a computationally difficult set covering problem". *Operations Research Letters*, 8:67–71, 1989.
- **Inteligencia de Enjambre:** Beni, G., Wang, J. (1989): Swarm Intelligence in Cellular Robotic Systems, Proceed. NATO Advanced Workshop on Robots and Biological Systems, Tuscany, Italy, June 26–30 (1989)
- **Colonias de hormigas:** Marco Dorigo (1992): "Optimization, Learning and Natural Algorithms" PhD thesis, Politecnico di Milano, 1992.
- **Colonia de Abejas:** Karaboga, D, (2005). An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.
- **Enjambre de partículas:** J. Kennedy & R. Eberhart (1995): "Particle Swarm Optimization". Proceedings of IEEE International Conference on Neural Networks, 1995.

Metaheurísticas. Tipología



PROBLEMA

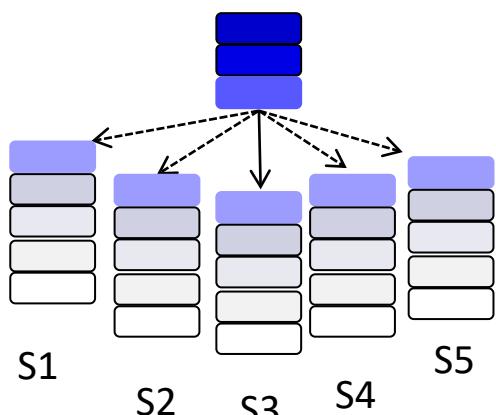
modelado

- Modelo Soluciones
- Transformación / Construcción
- Evaluación Soluciones

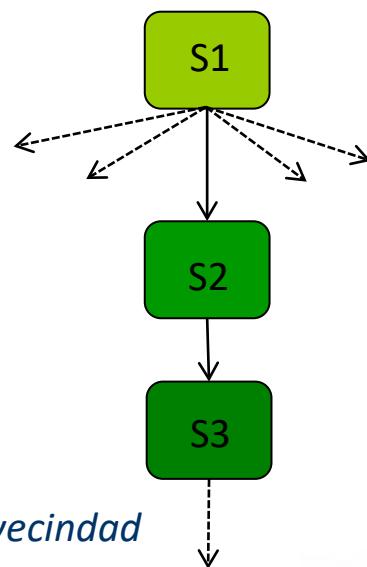
búsqueda

SOLUCIÓN

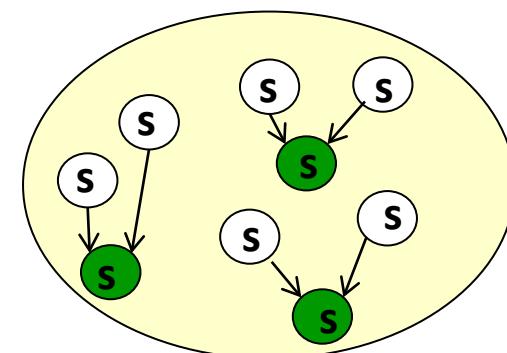
¿Qué padres, vecinos, sucesor...?



construcción

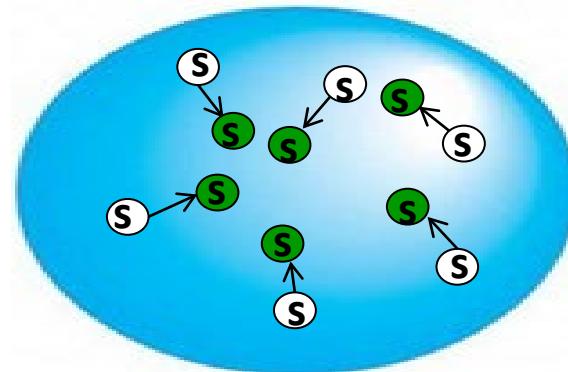


vecindad



cruce / colaboración

colaboración social



Metaheurísticas de Trayectorias:

- Búsqueda tabú
- Búsqueda local guiada
- Enfriamiento simulado, etc..

Metaheurísticas Poblacionales:

- Búsqueda dispersa
- Algoritmos evolutivos, genéticos
- Algoritmos meméticos
- Sistemas de hormigas
- Inteligencia de enjambre, etc.

Multi-arranque / Constructivos

- GRASP

Inspiradas o no en la Naturaleza

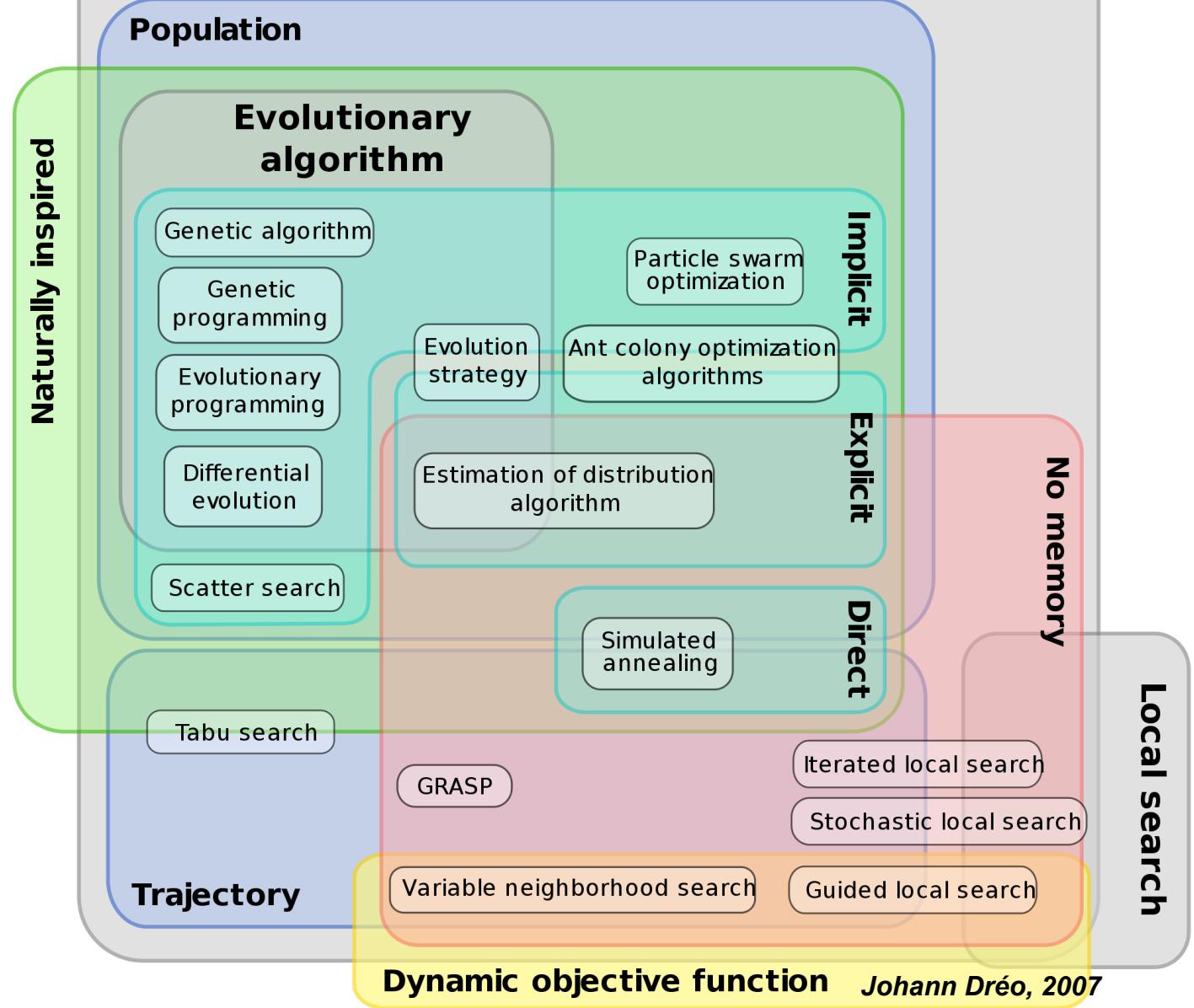
- Inspiradas: Genéticos Hormigas, Enfr. Simulado, etc.
- No Inspiradas: Búsqueda dispersa, GRASP...

Aleatorias vs. Determinísticas

- Aleatorias: Genéticos, ES, etc.
- Deterministas: Tabú, Búsqueda Dispersa, etc.

Incluso, las redes neuronales, la programación por restricciones, etc., son a veces incluidas en el término ‘metaheurística’

Metaheuristics



Exploración vs. explotación de soluciones

En el proceso de búsqueda, se debe hacer continuamente una elección fundamental: *exploración vs. explotación*

Exploración de alternativas en el espacio de soluciones

- La exploración busca alternativas un amplio espacio de estados ≈ Búsqueda global
- Evita convergencia prematura

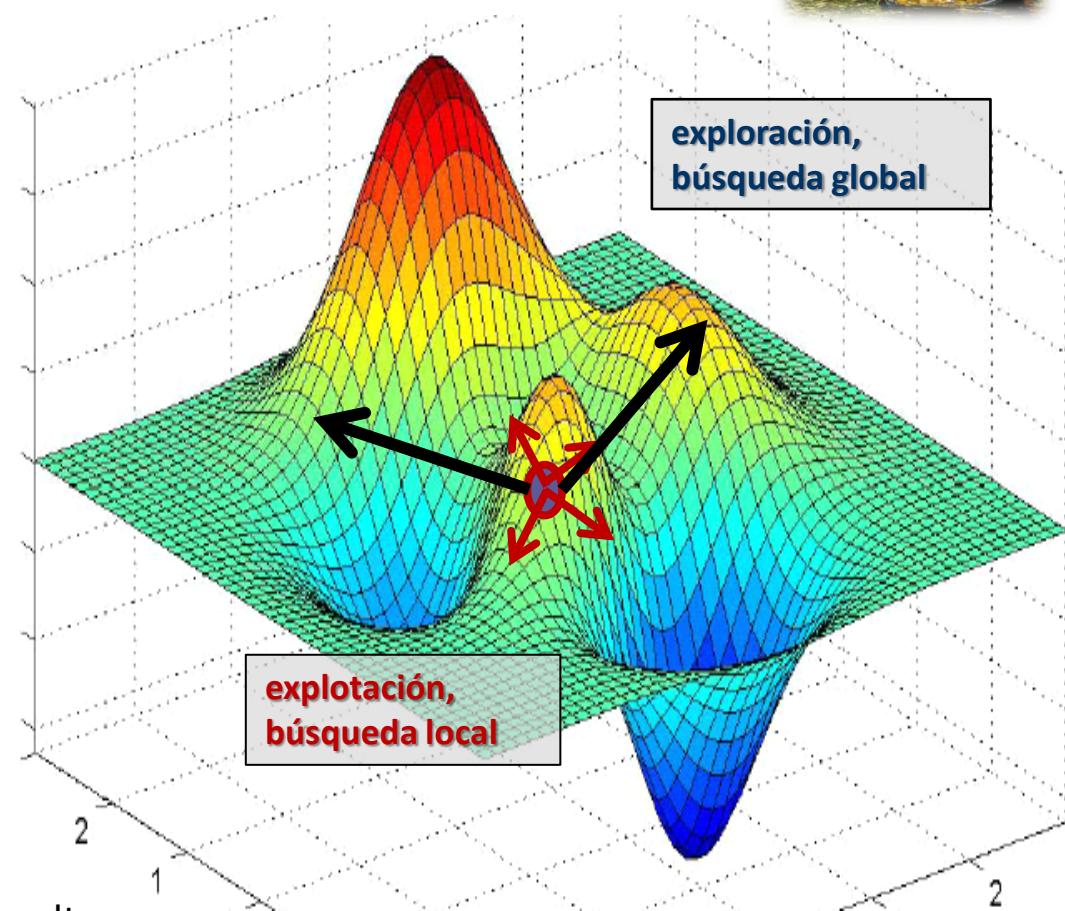
Tiende a obtener más información

Explotación de alternativas vecinas de una buena solución

- Permite focalizar sobre el entorno de buenas soluciones (alto fitness) ≈ Búsqueda local en la vecindad

Tiende a tomar la mejor decisión, dada la información actual

Las metaheurísticas tienden a combinar y/o alternar, con mayor o menor éxito, ambas opciones (*exploración vs. explotación*)



Exploración vs. explotación. Ejemplo típico (Falkenauer)

Tenemos dos tragaperras. Una de las dos da premios con mayor frecuencia que la otra. Por ejemplo, una da premio un 60% de las veces y la otra un 40%. Pero no se sabe cuál es cada una. Lo ideal es jugar solo a la máquina buena, pero ¿cómo saberlo?

La cuestión es:

¿cómo se debe jugar para maximizar las ganancias?

Estrategia posible:

- 1) Jugar n veces la izquierda y n veces la derecha Exploración
- 2) Comparar resultados y determinar mejor palanca
- 3) Jugar solo a la máquina buena Explotación

Pero *¿Cuántas ($n+n$) veces probar al principio?*

¿Cabe volver a probar ambas máquinas?



- La división entre las etapas de exploración/explotación es artificiosa
- Parece mejor integrar ambas etapas y seguir jugando ambas palancas (aunque cada vez se juegue más veces la palanca ganadora). ¿Cómo distribuir n_1 vs. n_2 ?
- Por ejemplo, (Holland – Alg. Genéticos) propone la estrategia de que el nº de ensayos con la palanca ganadora (o probabilidad de jugarla) crezca exponencialmente respecto a la mala

¿Y si tenemos k máquinas tragaperras?

Opt4J. Entorno libre



A Modular Framework for Meta-heuristic Optimization

Disponible (ejecutable y fuente) en: <https://sdarg.github.io/opt4j/>

Formulación sencilla de problemas utilizando librerías implementadas en Java

Existe un boletín que explica su instalación, uso e integración en ECLIPSE (java).

Entornos libres:

HeuristicLab <http://dev.heuristiclab.com/>

Entorno generalista en C#

PARADISEO <https://nojhan.github.io/paradiseo/>

API de programación en C++

jMetal <https://jmetal.sourceforge.net/>

Framework de optimización multi-objetivo en Java

Entornos comerciales:

MATLAB & Global Optimization Toolbox. Sistema Comercial MATLAB

<http://es.mathworks.com/products/global-optimization/>

<http://es.mathworks.com/help/gads/index.html>

Elección de una metaheurística

Elegir una metaheurística no es fácil y no siempre existe una única y mejor elección

Algoritmo de las Hormigas

Enfriamiento Simulado

BÚSQUEDA DISPERSA

Algoritmos Genéticos

GRASP

Algoritmos Meméticos

Búsqueda Tabú

Algoritmo de las Abejas



Aspectos a tener en cuenta:

1. Estudiar el problema. ¿Es un problema de **satisfactibilidad** o de **optimalidad**? ¿Hay que optimizar una única métrica o tenemos varias métricas (multi-objetivo)?
2. ¿Qué información heurística sobre el dominio se conoce y cómo se puede representar?
3. ¿Podemos **generar** fácilmente soluciones sub-óptimas? ¿Generamos estas soluciones de forma cuidadosa (**inteligentemente**) o utilizamos un método **aleatorio**? ¿Cómo generaremos los vecinos (búsqueda local)?
4. ¿Podemos **estimar** el valor de la solución óptima (problema del viajante de comercio vs. problema del cartero chino)?
5. ¿Una solución tiene que cumplir muchas restricciones? ¿Qué pasa si se viola alguna restricción? ¿Qué hacemos ante esa situación?
6. ¿Se puede prever el comportamiento esperado en la mejora de las soluciones?
7. ¿Existe alguna metaheurística que se adapta mejor al problema?

Evaluaciones empíricas de metaheurísticas

Table 1
Summary of the analysis: techniques and their properties.

(López et al., *Engineering Applications of Artificial Intelligence*
doi:10.1016/j.engappai.2008.10.014)

Method	Modeling	Anytime	Time	Memory	Tuning	Tool
Chronological backtracking	Coded	Partial	High	Low	No	No
Branch and bound	Coded	Partial	High	Low	No	No
Mixed integer programming	Declarative	No/yes	Low	High	No	GLPK
Constraint logic programming	Declarative	No	High	Low	No	GProlog
Forward checking	Declarative	No	Medium	Medium	No	ConFlex
Synchronous backtracking	Declarative	No/yes	Medium	Medium	No	No
Asynchronous backtracking	Declarative	No	High	High	No	ADOPT
Tabu	Coded	Yes	Low	Medium	Yes	No
GRASP	Coded	Yes	Low	Medium	Yes	No
Combinatorial auctions	Coded	Partial	Medium	High	No	GLPK
Genetic algorithms	Coded	Yes	Low	Low	Yes	No
Ant colony optimization	Coded	No	Low	Medium	Yes	No

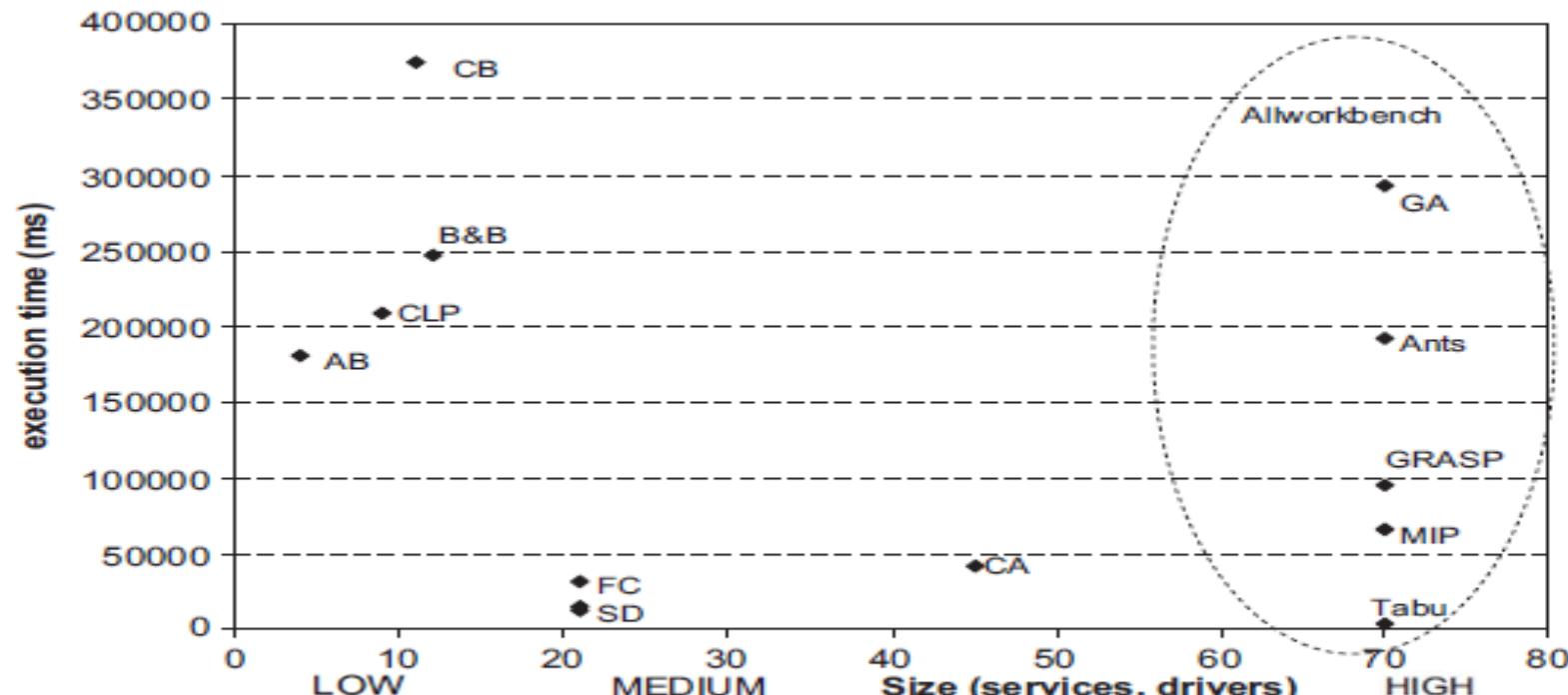


Fig. 7. Maximum problems managed by the methods and the associated execution time.

Evaluaciones empíricas de metaheurísticas. Artículos en Poliformat

Instance	GA		ACO		SA	
	Best Distance	Time (s)	Best Distance	Time (s)	Best Distance	Time (s)
bier127	419224	~3	124651.524	~27	265289	~0.3
berlin52	11551	~1	7721.432	~4	10586	~0.2
ali535	9466	~5	1510.637	~62	6471	~0.3

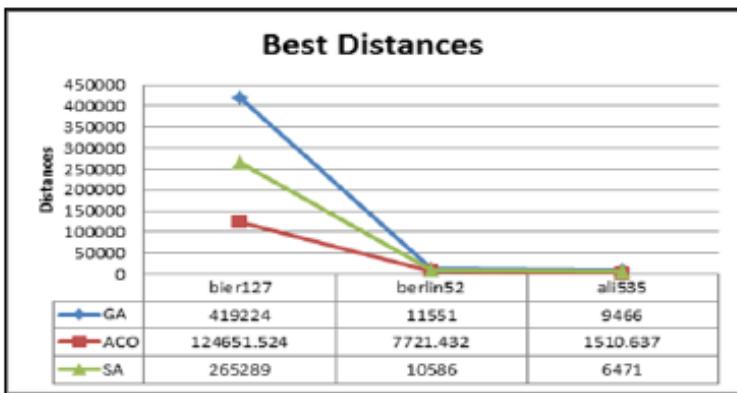


Figure 12. Evolution of GA, ACO and SA results over best distance for bier127, berlin52, and ali535

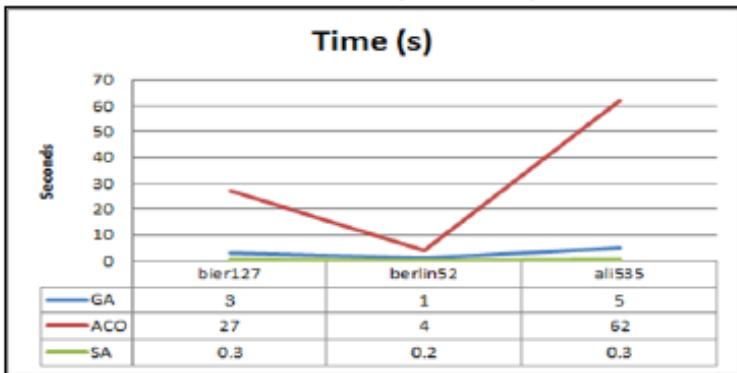


Figure 13. Evolution of GA, ACO and SA results over time(s) for bier127, berlin52 and ali535

Contents lists available at SciVerse ScienceDirect
Information Sciences
journal homepage: www.elsevier.com/locate/ins

A survey on optimization metaheuristics
Ilhem Boussaïd ^a, Julien Lepagnot ^b, Patrick Siarry ^{b,*}

^aUniversité des sciences et de la technologie Houari Boumediene, Electrical Engineering and Computer Science Department, El-Alia BP 32 Bab-Ezzouar, 16111 Algiers, Algeria
^bUniversité Paris Est Créteil, LISSI, 61 avenue du Général de Gaulle 94010 Créteil, France

Artif Intell Rev (2015) 44:1–21
DOI 10.1007/s10462-013-9399-6

CrossMark

Review of state of the art for metaheuristic techniques in Academic Scheduling Problems
Chong Keat Teoh · Antoni Wibowo · Mohd Salihin Ngadiman

In: **Performance Comparison of Simulated Annealing, GA and ACO Applied to TSP.** Int J of Intelligent Computing Research 6, 4, Dec2015

Teorema de “no free lunch”

(*No free lunch theorems for optimization*, Wolpert, D.H.; Macready, W.G.; *IEEE Transactions on Evolutionary Computation*, 1, 1997)



***“The computational cost of finding a solution,
averaged over all problems in the class, is the same for any solution method”***

***“For any algorithm, any elevated performance over one class of problems
is exactly paid for in performance over another class”***

- “Todos los algoritmos (de búsqueda o de optimización) tienen la misma evaluación cuando se promedia su aplicación sobre el conjunto de problemas”

“Para cada algoritmo de búsqueda/optimización, cualquier superioridad que pudiera tener sobre una clase de problemas es exactamente penalizada sobre las otras clases de problemas”

- “En un método de búsqueda, todas las funciones heurísticas (de búsqueda u optimización) tienen la misma evaluación cuando se promedian sus aplicaciones sobre el conjunto de funciones de coste”

“La superioridad de una heurística, definida sobre una función de coste, es exactamente penalizada cuando se aplica sobre las otras funciones de coste”

Por lo tanto, para cualquier algoritmo de búsqueda/optimización, su alto rendimiento sobre una clase concreta de problemas, es exactamente penalizado sobre otra clase de problemas

Diversas discusiones relacionadas: <http://www.no-free-lunch.org/>

La elección entre búsqueda heurística y metaheurística depende del problema específico y de los objetivos de optimización



ChatGPT

Búsqueda Heurística

Las heurísticas son reglas generales o métodos aproximados que suelen funcionar bien para ciertos tipos de problemas. Estas reglas pueden derivarse de la experiencia o el conocimiento experto. Las búsquedas heurísticas son más simples y directas en comparación con las metaheurísticas, y a menudo son efectivas para problemas relativamente pequeños o donde existe un conocimiento previo específico

Ventajas

- Más rápidas de implementar (AÑADIDO: depende)
- Efectivas en problemas con estructuras y restricciones conocidas
- Son más eficientes en problemas pequeños y bien definidos

Desventajas

- Pueden quedar atrapadas en óptimos locales (AÑADIDO: no si son completas)
- No son tan eficaces en problemas complejos o de alta dimensionalidad
- Pueden no encontrar soluciones óptimas en problemas difíciles (AÑADIDO: por un problema de tiempo o incompletitud)

Búsqueda metaheurística

Las metaheurísticas son enfoques más generales y flexibles para la optimización. Se diseñan para explorar soluciones en un espacio de búsqueda más amplio, lo que las hace adecuadas para problemas complejos y difíciles. Las metaheurísticas son más capaces de escapar de óptimos locales (AÑADIDO: ¡no siempre!) y pueden utilizarse para una variedad de problemas, independientemente de la estructura específica del problema

Ventajas

- Más efectivas en problemas difíciles y complejos
- Pueden evitar óptimos locales y explorar soluciones más ampliamente (AÑADIDO: exploración vs. explotación)
- Adecuadas para problemas donde la estructura no es bien conocida (AÑADIDO: y no es fácil encontrar una buena heurística)

Desventajas

- Pueden requerir más tiempo de implementación y ajuste (AÑADIDO: sobre todo de ajuste)
- No garantizan encontrar soluciones óptimas
- Pueden ser más intensivas en términos computacionales (AÑADIDO: pero pueden encontrar buenas soluciones en tiempos más razonables)