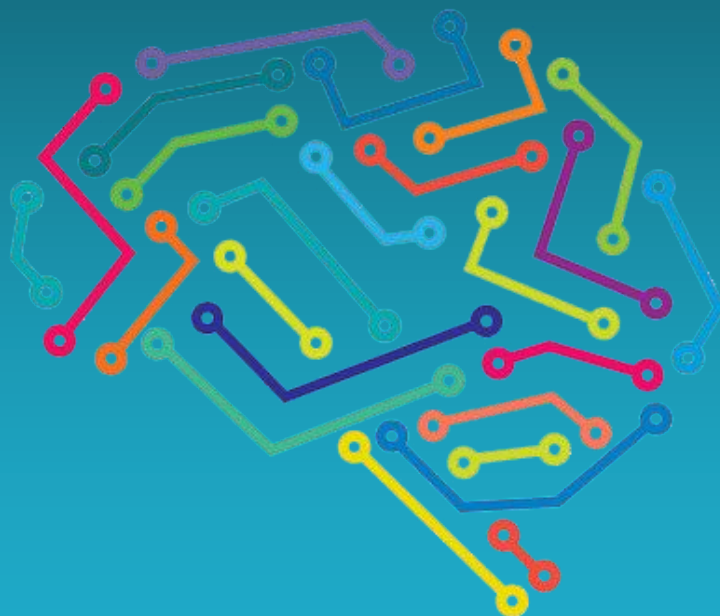




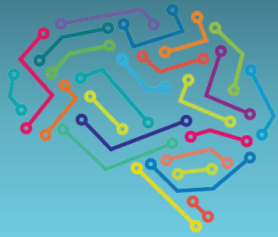
دانشگاه صنعتی شاهرود

ورای جستجوی کلاسیک



درس هوش مصنوعی

نیم سال اول تحصیلی ۹۸-۱۳۹۷



ورای جستجوی کلاسیک

□ الگوریتمهای جستجویی که تا کنون بررسی کردیم در مسائلی کاربرد داشتند که دارای سه ویژگی زیر باشند:

Observable ✓

Deterministic ✓

Known ✓

□ اگر این فرضهای ساده‌ساز را کنار بگذاریم:

✓ الگوریتمهای جستجوی محلی (local search)

✓ در مسائلی که برایشان، فقط هدف مهم است و نه مسیر رسیدن به آن



الگوریتمهای جست و جوی محلی و مسائل بهینه سازی

- ☐ معمولاً در هر بار فقط یک گره گسترش می یابد.
- ☐ معمولاً مسیر نگهداری نمی شود.
- ☐ حافظه خیلی اندکی مصرف می کنند.
- ☐ در فضاهای حالت خیلی بزرگ یا بینهایت (پیوسته) جواب خوبی را می یابند.
- ☐ می توانند در مسائل بهینه سازی نیز مفید باشند. در این مسئله ها هدف یافتن بهترین حالت بر اساس تابع هدف است.
- ☐ وقتی کاملند که هدف را بیابند.
- ☐ وقتی بهیته هستند که مینیمم یا ماکسیمم سراسری را بیالند.



جست و جوی تپه نوردی (جست و جوی محلی حریصانه)

□ این الگوریتم حلقه ای است که در جهت افزایش مقدار حرکت می کند وقتی به قله ای رسید که هیچ همسایه ای از آن بلند تر نیست خاتمه می یابد.

□ سعی در یافتن قله اورست در مه شدید در حالی که دچار فراموشی هستید.

function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

current \leftarrow MAKE-NODE(*problem*.INITIAL-STATE)

loop do

neighbor \leftarrow a highest-valued successor of *current*

if *neighbor*.VALUE \leq *current*.VALUE **then return** *current*.STATE

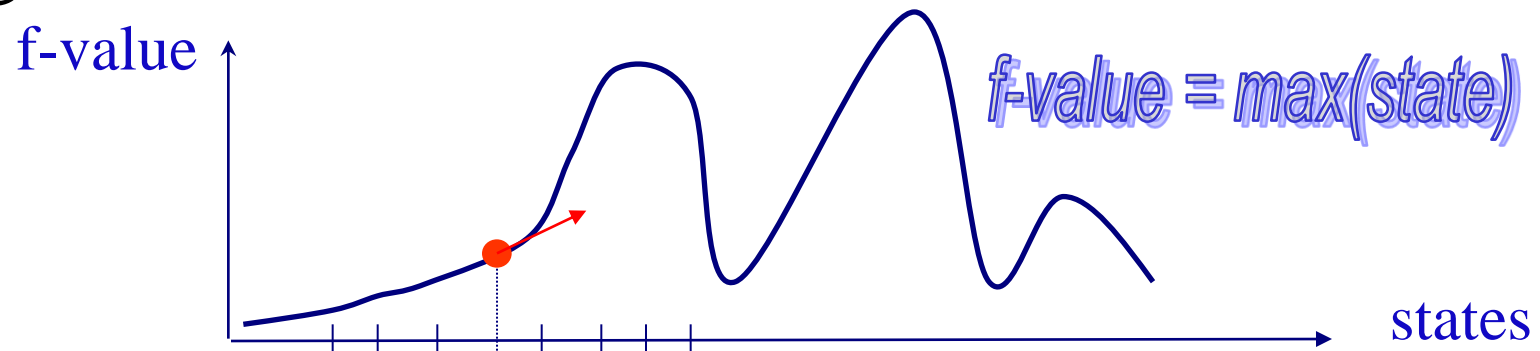
current \leftarrow *neighbor*

Figure 4.2 The hill-climbing search algorithm, which is the most basic local search technique. At each step the current node is replaced by the best neighbor; in this version, that means the neighbor with the highest VALUE, but if a heuristic cost estimate h is used, we would find the neighbor with the lowest h .

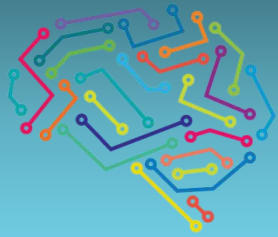


جست و جوی تپه نوردی (جست و جوی محلی حریصانه)

- Apply the rule that increases the most the current state value
- Move in the direction of the greatest gradient



```
while f-value(state) > f-value(best-next(state))  
    state := next-best(state)
```



دلایل شکست تپهنوردی

□ این سیاست ساده، سه زیان عمده دارد:

□ مینیمم یا ماکسیمم محلی: یک ماکزیمم محلی، برخلاف ماکزیمم عمومی، قله‌ای است که پائین‌تر از بلندترین قله در فضای حالت است. زمانی که روی ماکزیمم محلی هستیم، الگوریتم توقف خواهد نمود. اگرچه راه حل نیز ممکن است دور از انتظار باشد.



دلایل شکست تپه‌نوردی

❑ Ridges: تیغه، دارای لبه‌های سراشیب و ترکیبی از تعداد زیادی مینیمم محلی در مجاورت یکدیگر است. مگر اینکه عملگرهایی موجود باشند که مستقیماً به سمت بالای نوک کوه حرکت کنند، جستجو ممکن است از لبه‌ای به لبه دیگر نوسان داشته باشد و پیشرفت کمی را حاصل شود.





دلایل شکست تپهنوردی

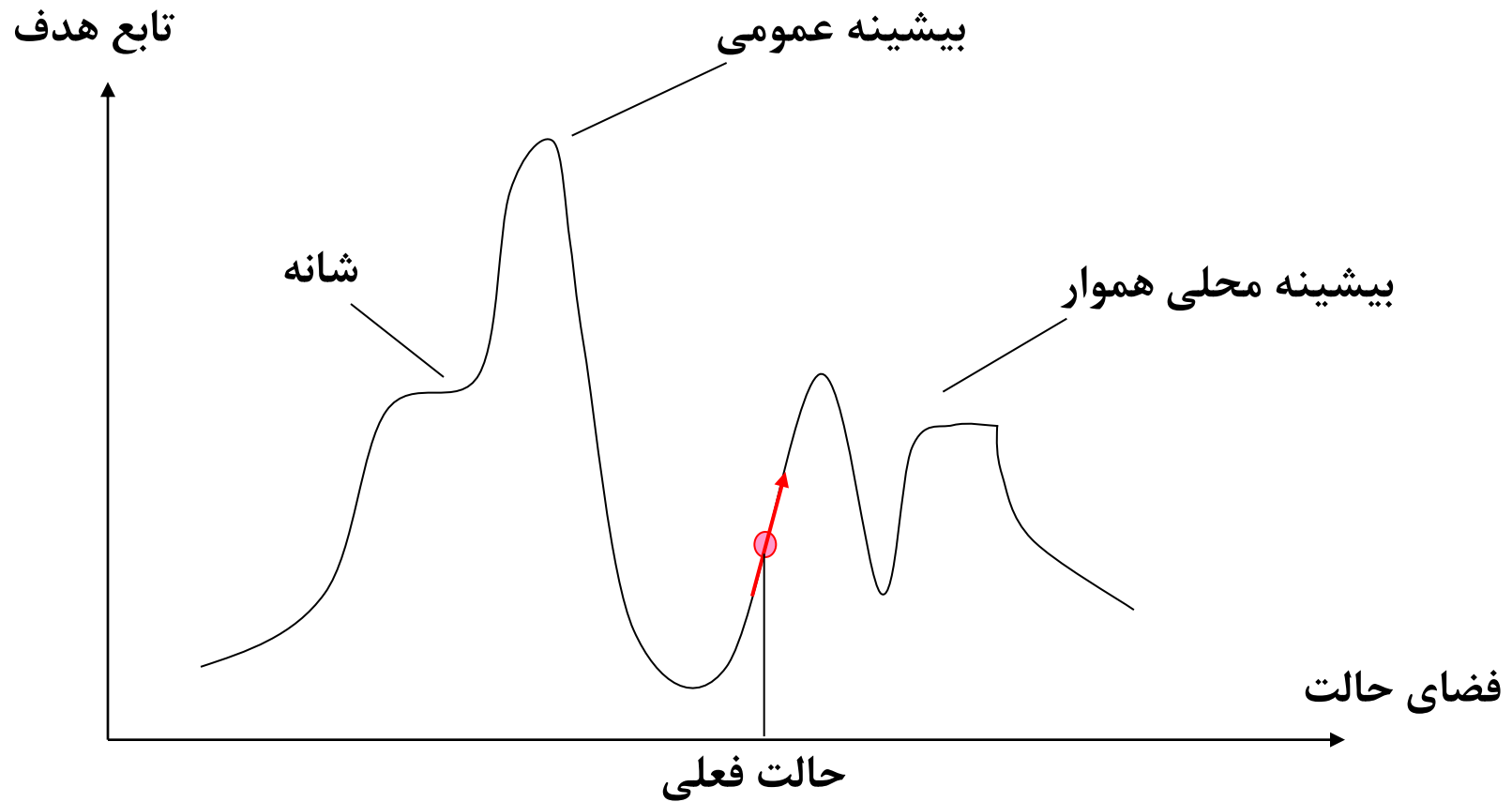
□ یک فلات محوطه‌ای از فضای حالت است که تابع ارزیاب
یکنواخت باشد.

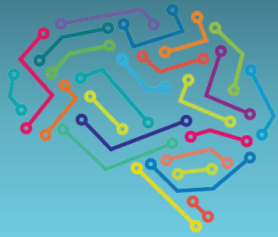
□ راه حل: Sideways Move





جست و جوی تپه نوردی (جست و جوی محلی حریمانه)



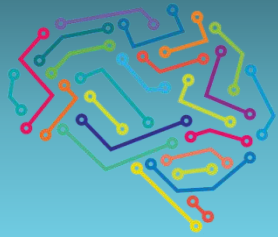


مساله ۸ وزیر با استفاده از جست و جوی تپه نوردی

استفاده از فرمول بندی کامل

هر حالت ۵۶ جانشین دارد

تابع اکتشافی h تعداد جفت هایی از وزیران است که به طور مستقیم و غیر مستقیم یکدیگر را می زنند.



مساله ۸ وزیر با استفاده از جست و جوی تپه نوردی

۱۸	۱۲	۱۴	۱۳	۱۳	۱۲	۱۴	۱۴
۱۴	۱۶	۱۳	۱۵	۱۲	۱۴	۱۲	۱۶
۱۴	۱۲	۱۸	۱۳	۱۵	۱۲	۱۴	۱۴
۱۵	۱۴	۱۴	W	۱۳	۱۶	۱۳	۱۶
W	۱۴	۱۷	۱۵	W	۱۴	۱۶	۱۶
۱۷	W	۱۶	۱۸	۱۵	W	۱۵	W
۱۸	۱۴	W	۱۵	۱۵	۱۴	W	۱۶
۱۴	۱۴	۱۳	۱۷	۱۲	۱۴	۱۲	۱۸

$h = 17$

						W	
				W			
	W						
			W				
					W		
							W
		W					
W							

$h = 1$



Escaping Shoulders: Sideways Move

- ❑ If no downhill (uphill) moves, allow sideways moves in hope that algorithm can escape
 - ✓ Must limit the number of possible sideways moves to avoid infinite loops
- ❑ For 8-queens
 - ✓ Allow sideways moves with limit of 100
 - ✓ Raises percentage of problems solved from 14 to 94%
 - ✓ However...
 - 21 steps for every successful solution
 - 64 for each failure



Hill Climbing Properties

- ❑ Not complete.
- ❑ Terrible worst case running time.
- ❑ Simple, $O(1)$ space, and often very fast.



Hill Climbing: Stochastic Variations

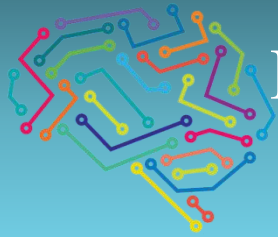
- ❑ When the state-space landscape has local minima, any search that moves only in the greedy direction cannot be complete
- ❑ Random walk, on the other hand, *is* asymptotically complete
- ❑ *Idea:* Combine random walk & greedy hill-climbing
- ❑ At each step do one of the following:
 - ✓ *Greedy*: With prob. p move to the neighbor with largest value
 - ✓ *Random*: With prob. $1-p$ move to a random neighbor



Hill-climbing with random restarts

- ❑ If at first you don't succeed, try, try again!
- ❑ Different variations
 - ✓ For each restart: run until termination vs. run for a fixed time
 - ✓ Run a fixed number of restarts or run indefinitely
- ❑ Analysis
 - ✓ Say each search has probability p of success
 - ✓ e.g., for 8-queens, $p = 0.14$ with no sideways moves
- ❑ Expected number of restarts?
- ❑ Expected number of steps taken?





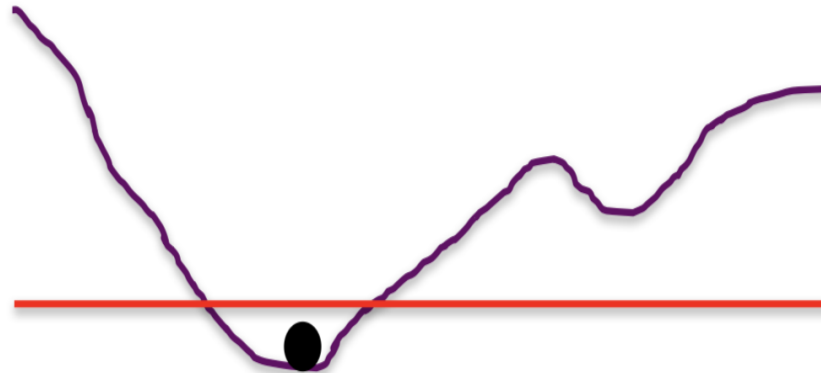
Hill-Climbing with Both Random Walk & Random Sampling

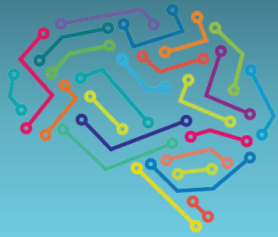
- At each step do one of the three
 - ✓ **Greedy**: move to the neighbor with largest value
 - ✓ **Random Walk**: move to a random neighbor
 - ✓ **Random Restart**: Start over from a new, random state



Simulated Annealing

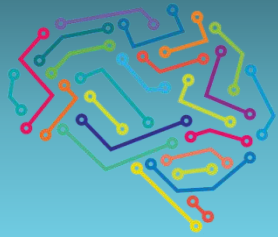
- ❑ Idea: escape local maxima by allowing some “bad” moves
 - ✓ **but gradually decrease their size and frequency**
 - ✓ method proposed in 1983 by IBM researchers for solving VLSI layout problems
- ❑ A Physical Analogy:
 - ✓ Imagine letting a ball roll downhill on the function surface
 - ✓ Now shake the surface, while the ball rolls,
 - ✓ Gradually reducing the amount of **shaking**





Simulated Annealing (cont.)

- ❑ Annealing = physical process of cooling a liquid → frozen
- ❑ simulated annealing:
 - ✓ free variables are like particles
 - ✓ seek “low energy” (high quality) configuration
 - ✓ slowly reducing temp. T with particles moving around randomly
- ❑ high T : probability of “locally bad” move is higher
- ❑ low T : probability of “locally bad” move is lower
- ❑ typically, T is decreased as the algorithm runs longer
 - ✓ i.e., there is a “temperature schedule”



Simulated Annealing (cont.)

function **SIMULATED-ANNEALING**(*problem*, *schedule*) **returns** a solution state

inputs: *problem*, a problem

schedule, a mapping from time to “temperature”

local variables: *current*, a node

next, a node

T, a “temperature” controlling prob. of downward steps

current \leftarrow MAKE-NODE(INITIAL-STATE[*problem*])

for *t* \leftarrow 1 **to** ∞ **do**

T \leftarrow *schedule*[*t*]

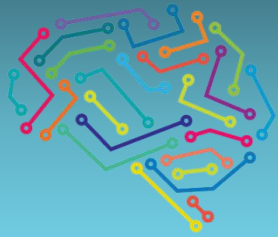
if *T* = 0 **then return** *current*

next \leftarrow a randomly selected successor of *current*

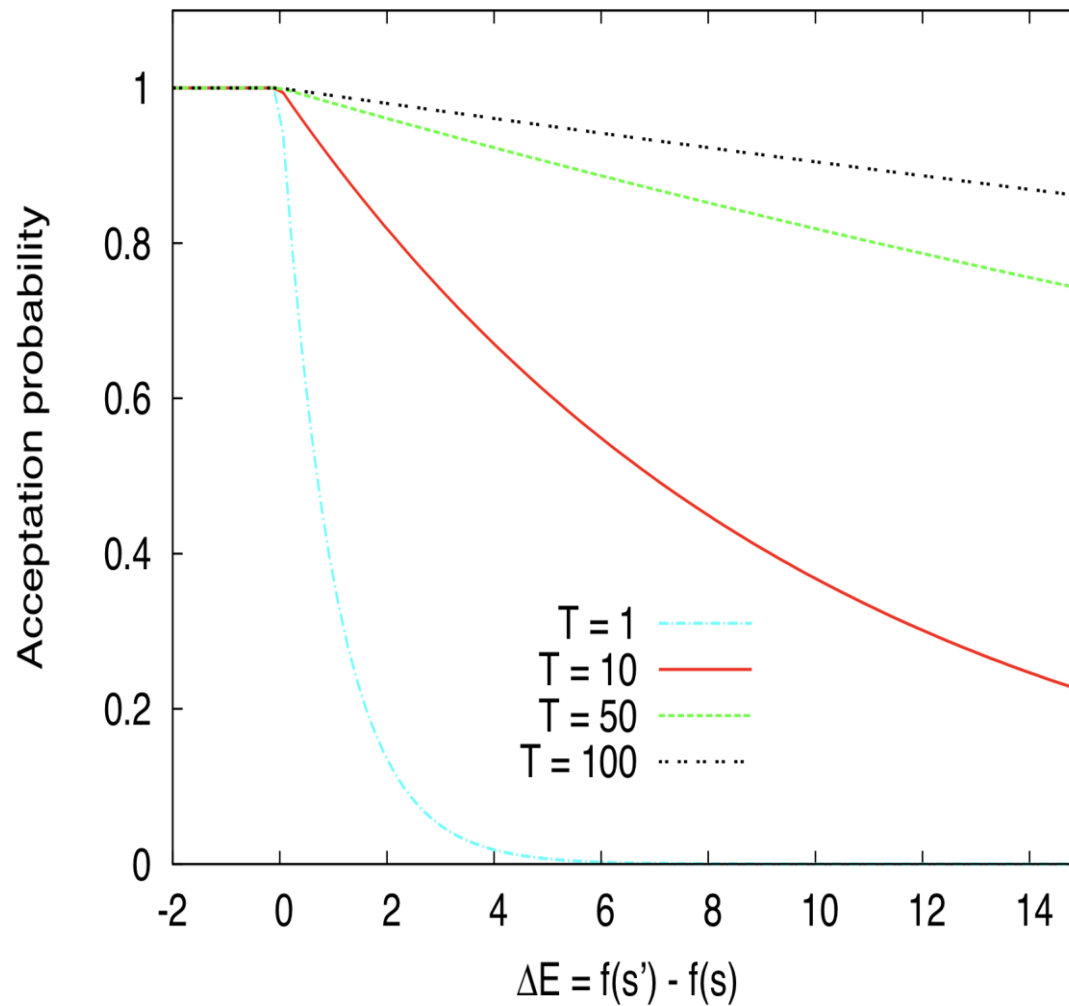
$\Delta E \leftarrow$ VALUE[*next*] – VALUE[*current*]

if $\Delta E > 0$ **then** *current* \leftarrow *next*

else *current* \leftarrow *next* only with probability $e^{\Delta E/T}$

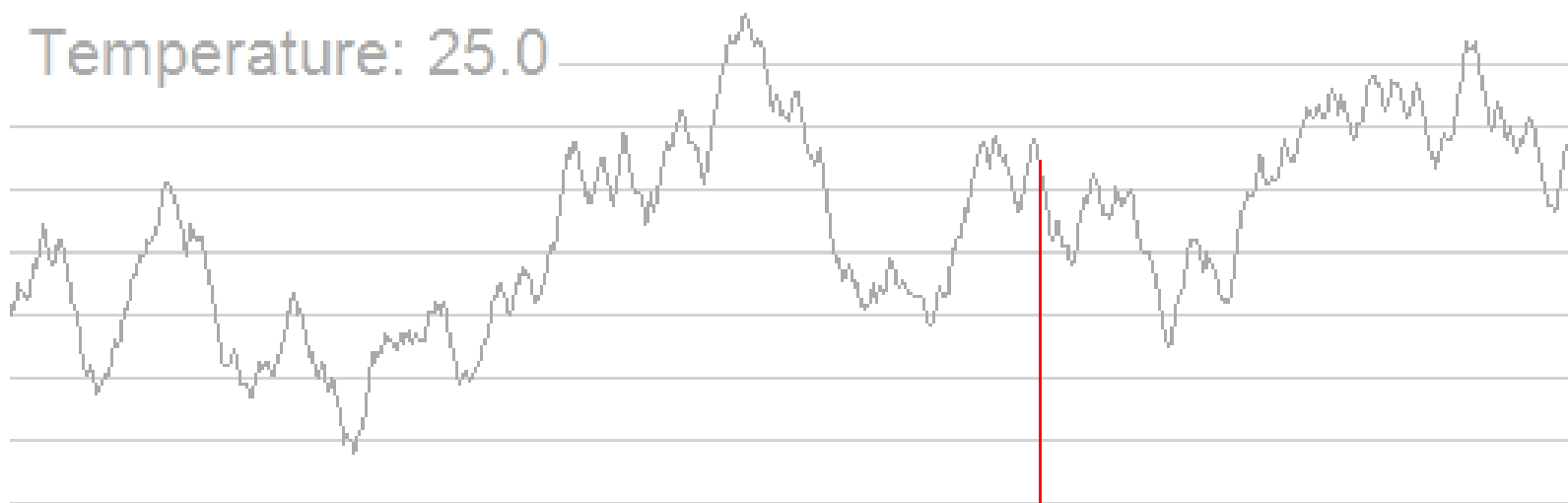


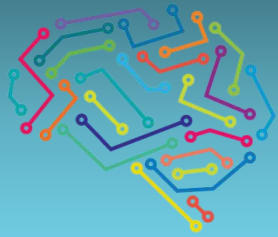
Effect of temperature





Temperature: 25.0





Simulated Annealing in practice

❑ Other applications:

✓ Traveling salesman, Graph partitioning, Graph coloring, Scheduling, Facility Layout, Image Processing, ...

❑ Optimal, given that T is decreased **sufficiently slow**.

❑ Convergence can be guaranteed if at each step, T drops no more quickly than $C/\log n$, C =constant, n = # of steps so far.



Local beam search

- ❑ Idea: Keeping only one node in memory is an extreme reaction to memory problems.
- ❑ Keep track of k states instead of one
 - ✓ Initially: k randomly selected states
 - ✓ Next: determine all successors of k states
 - ✓ If any of successors is goal \rightarrow finished
 - ✓ Else select k best from successors and repeat



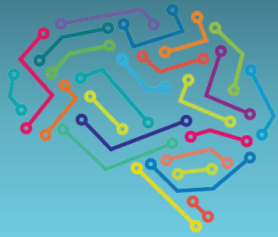
Local Beam Search

- ❑ Not the same as *k random-start searches that run in parallel!*
 - ✓ Searches that find good states recruit other searches to join them
- ❑ Problem: quite often, all *k states end up on same local hill*
- ❑ Idea: Stochastic beam search
 - ✓ Choose *k successors randomly, biased towards good ones*
- ❑ Observe the close analogy to **natural selection!**



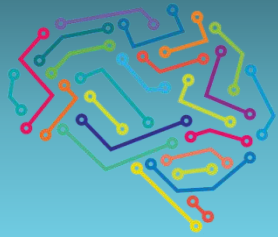
Genetic Algorithm (GA)

- ❑ Local beam search, but...
 - ✓ A successor state is generated by *combining two parent states*
- ❑ Start with k randomly generated states (**population**)
- ❑ A state is represented as a ***string*** over a finite alphabet (often a string of 0s and 1s)
- ❑ Evaluation function (**fitness function**). Higher = better
- ❑ Produce the next generation of states by **selection**, **crossover**, and **mutation**

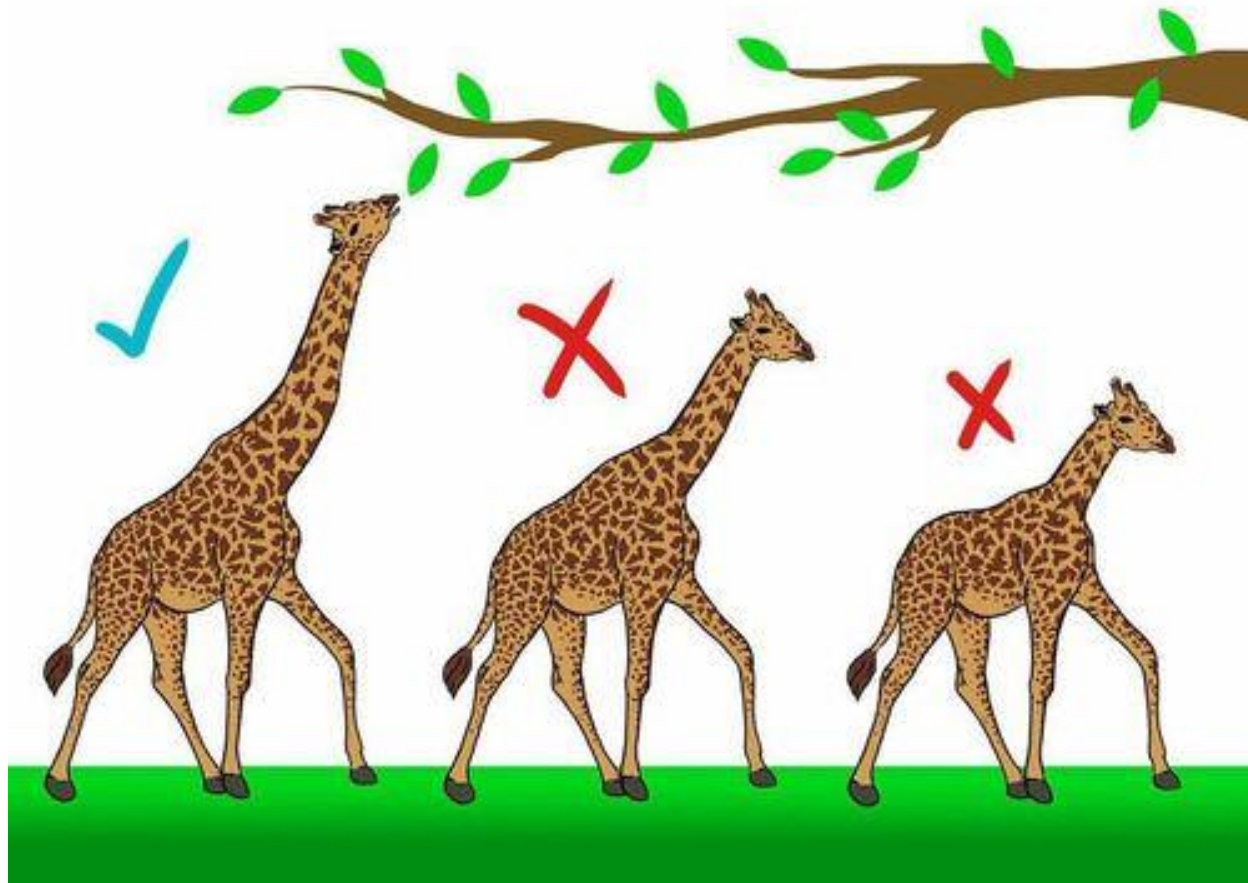


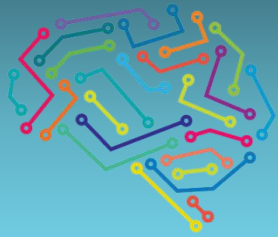
Natural selection





Natural selection





Natural selection

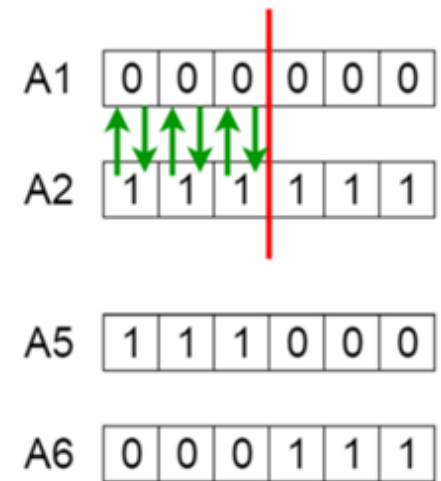
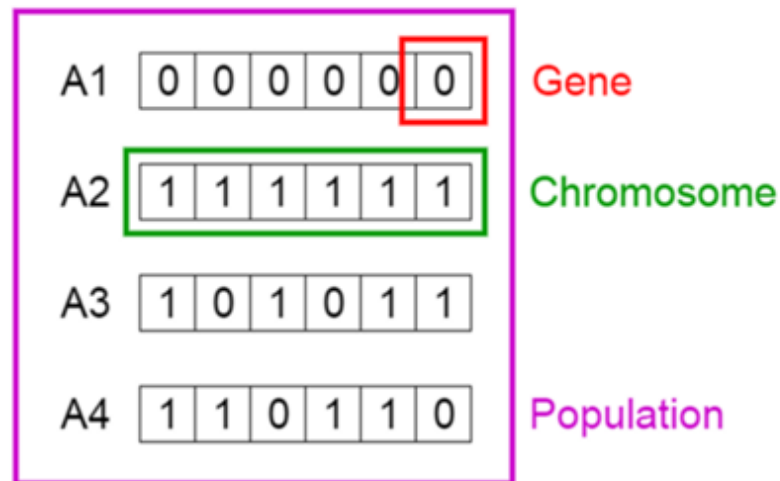
- ❑ The process of natural selection starts with the selection of fittest individuals from a population.
- ❑ They produce offspring which probably inherit the superior characteristics of the parents and will be added to the next generation.
- ❑ Their offspring who inherit the superior characteristics have a better chance at surviving.
- ❑ This process keeps on iterating and at the end, a generation with the fittest individuals will be found.

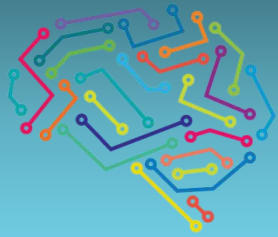


GA

❑ Five phases are considered in a genetic algorithm.

- ✓ Initial population
- ✓ Fitness function
- ✓ Selection
- ✓ Crossover
- ✓ Mutation



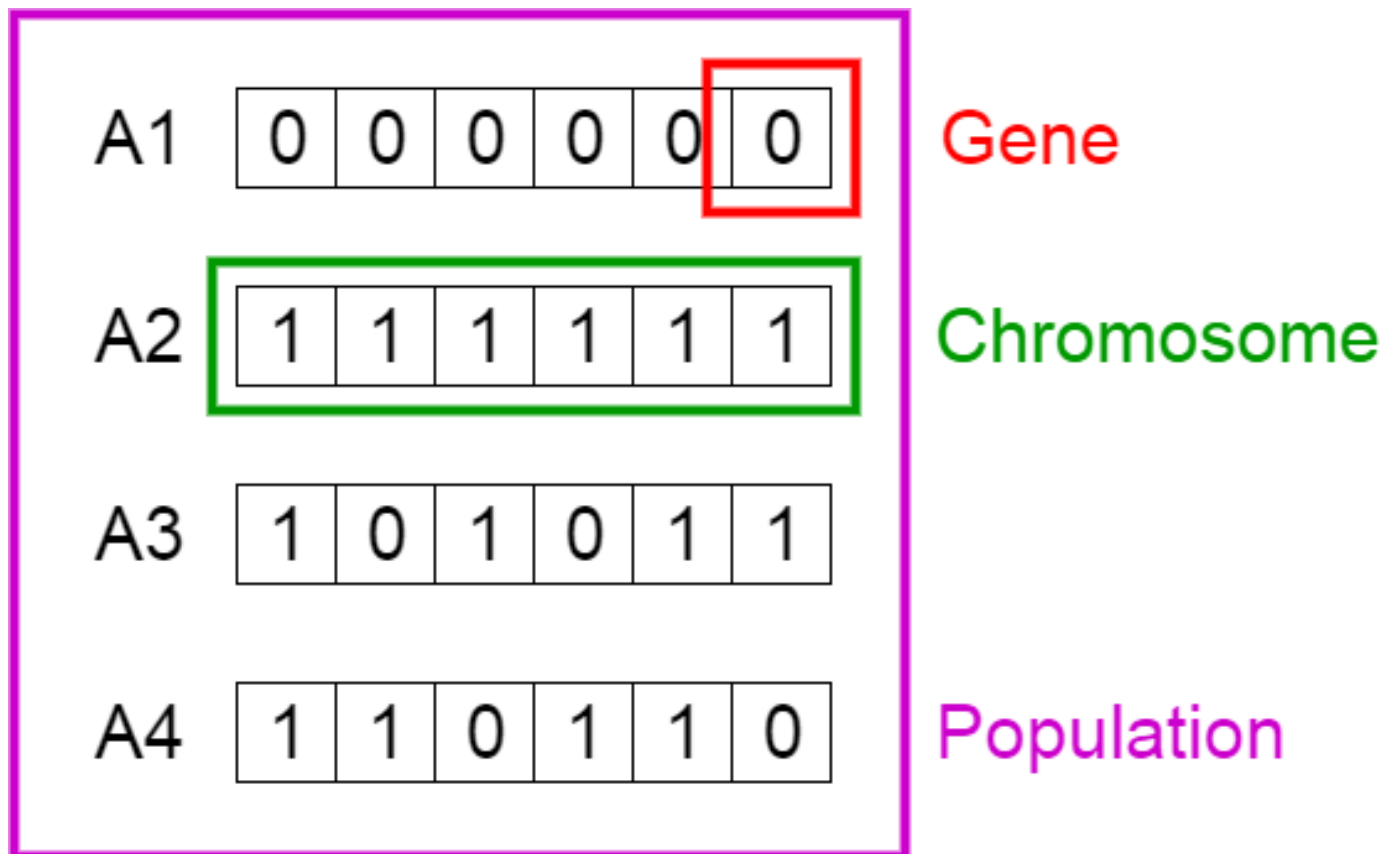


GA

- ❑ The process begins with a set of individuals which is called a **Population**. Each individual is a solution to the problem you want to solve.
- ❑ An individual is characterized by a set of parameters (variables) known as **Genes**. Genes are joined into a string to form a **Chromosome** (solution).
- ❑ In a genetic algorithm, the set of genes of an individual is represented using a string, in terms of an alphabet. Usually, binary values are used (string of 1s and 0s). We say that we encode the genes in a chromosome.



GA





GA

❑ Fitness Function

- ✓ The **fitness function** determines how fit an individual is (the ability of an individual to compete with other individuals). It gives a **fitness score** to each individual. The probability that an individual will be selected for reproduction is based on its fitness score.

❑ Selection

- ✓ The idea of **selection** phase is to select the fittest individuals and let them pass their genes to the next generation.
- ✓ Two pairs of individuals (**parents**) are selected based on their fitness scores. Individuals with high fitness have more chance to be selected for reproduction.

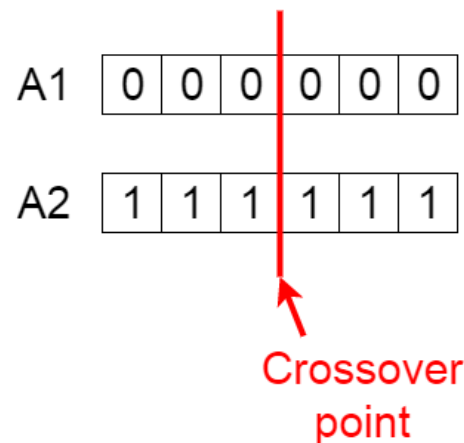


GA - crossover

□ Crossover

✓ **Crossover** is the most significant phase in a genetic algorithm. For each pair of parents to be mated, a **crossover point** is chosen at random from within the genes.

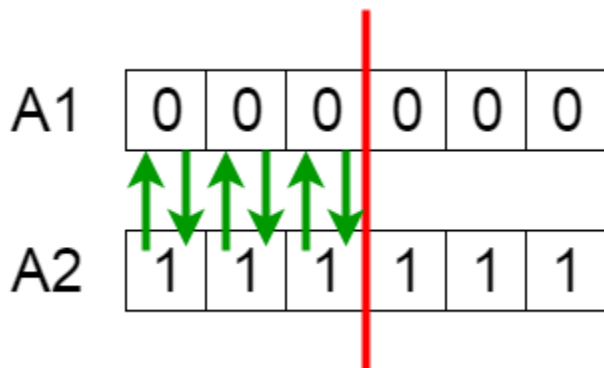
□ For example, consider the crossover point to be 3 as shown below.





GA - crossover

- ❑ **Offspring** are created by exchanging the genes of parents among themselves until the crossover point is reached.



- ❑ The new offspring are added to the population.

A5:

1	1	1	0	0	0
---	---	---	---	---	---

A6:

0	0	0	1	1	1
---	---	---	---	---	---



GA - Mutation

□ Mutation

- ✓ Sometimes, new offspring can be subjected to a **mutation** with a low random probability. This implies that some of the bits in the bit string can be flipped.
- Mutation occurs to maintain diversity within the population and prevent premature convergence.

Before Mutation

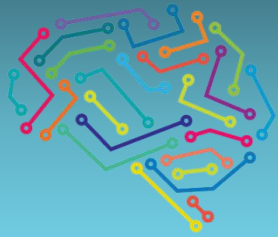
A5

1	1	1	0	0	0
---	---	---	---	---	---

After Mutation

A5

1	1	0	1	1	0
---	---	---	---	---	---



GA- other factors

❑ Heuristics

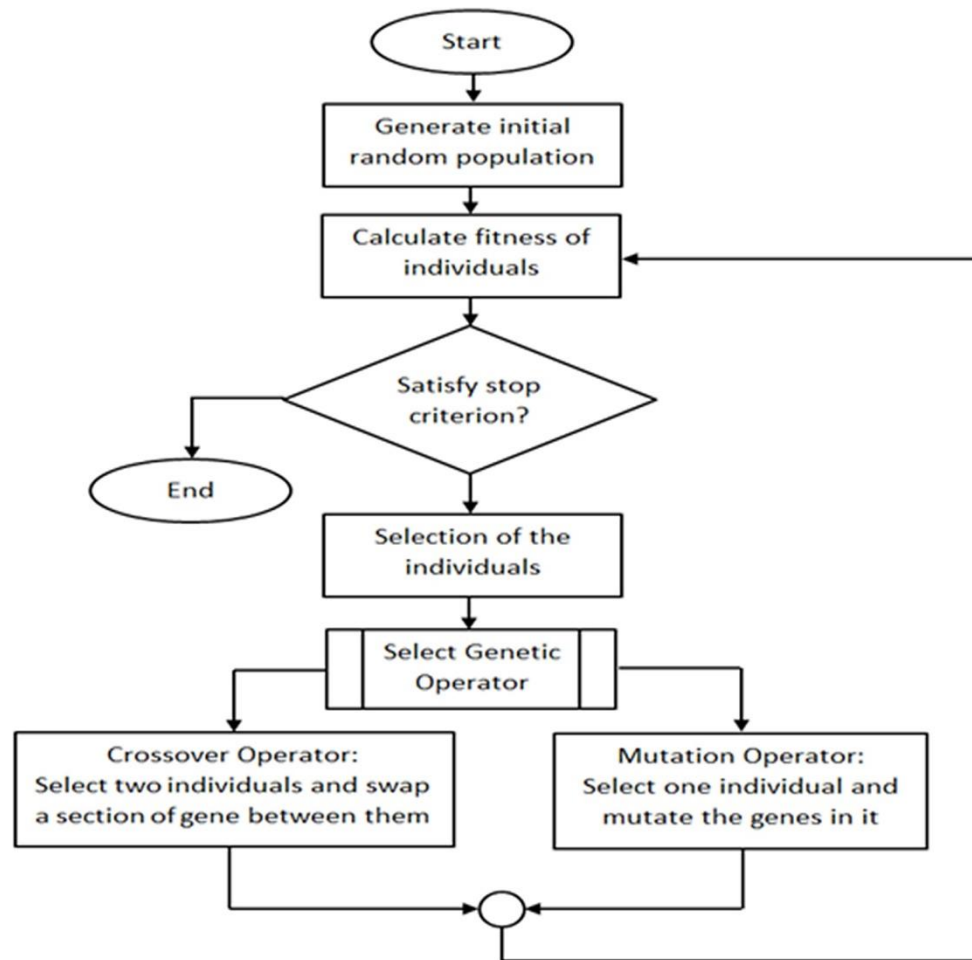
- ✓ In addition to the main operators above, other heuristics may be employed to make the calculation faster or more robust. The *speciation* heuristic penalizes crossover between candidate solutions that are too similar.

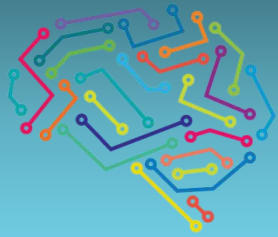
❑ Termination

- ✓ This generational process is repeated until a termination condition has been reached. Common terminating conditions are:
- ✓ A solution is found that satisfies minimum criteria
- ✓ Fixed number of generations reached
- ✓ Allocated budget (computation time/money) reached
- ✓ The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
- ✓ Manual inspection
- ✓ Combinations of the above

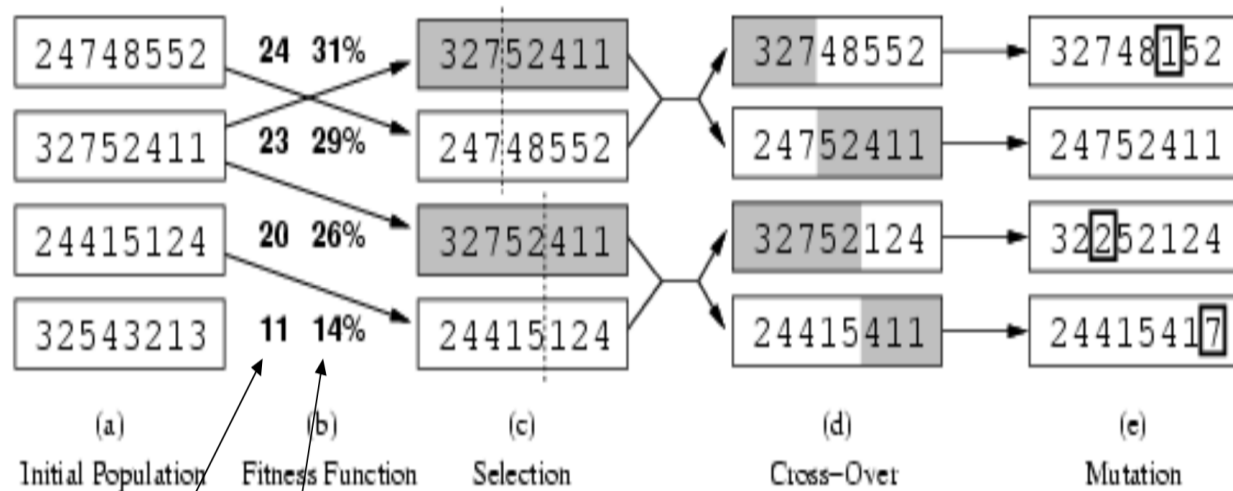


GA - Flowchart



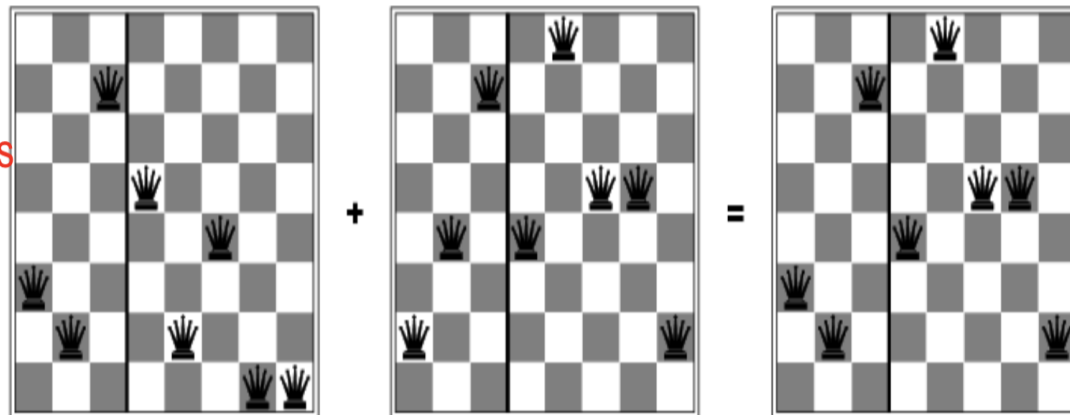


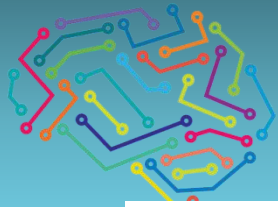
n-queens example



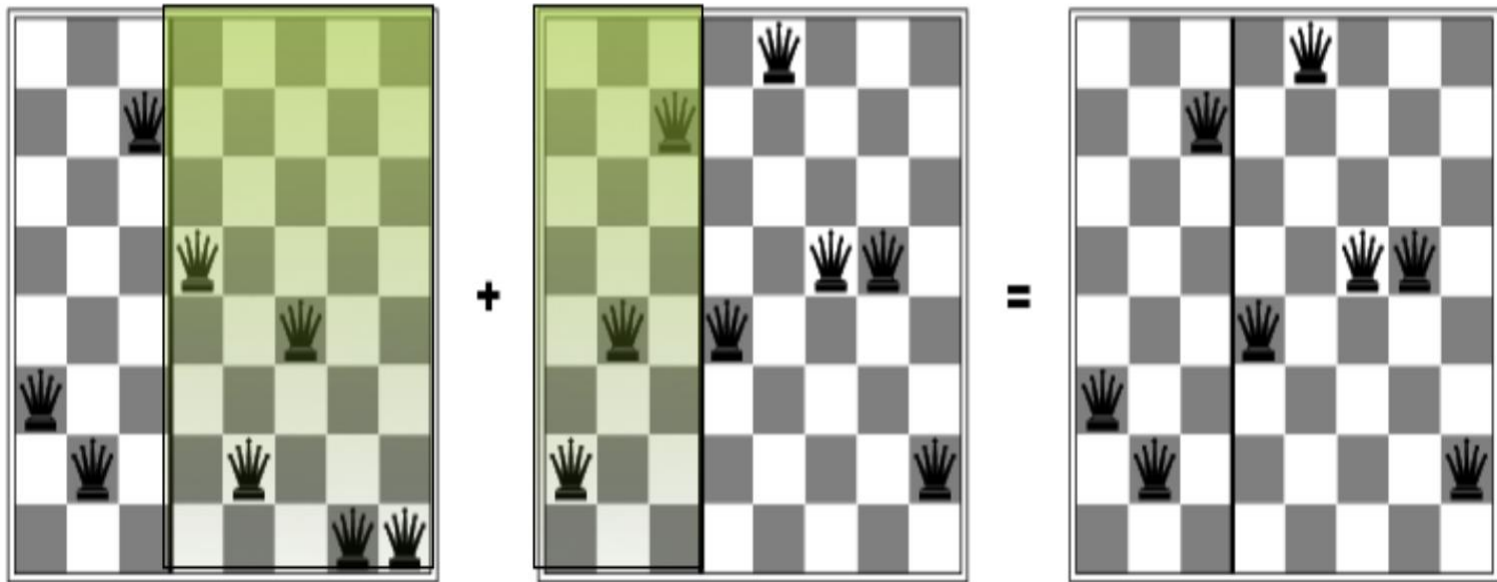
fitness:
#non-attacking queens

probability of being
regenerated
in next generation





n-queens example (cont.)



Has the effect of “jumping” to a completely different new part of the search space (quite non-local)



Comments on Genetic Algorithms

❑ Genetic algorithm is a variant of “stochastic beam search”

❑ Positive points

- ✓ Random exploration can find solutions that local search can't
 - (via crossover primarily)
- ✓ Appealing connection to human evolution
 - “neural” networks, and “genetic” algorithms are **metaphors!**

❑ Negative points

- ✓ Large number of “tunable” parameters
 - Difficult to replicate performance from one problem to another
- ✓ Lack of good empirical studies comparing to simpler methods
- ✓ Useful on some (small?) set of problems but no convincing evidence that GAs are better than hill-climbing w/random restarts in general



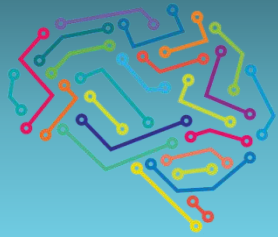
Genetic Programming

- ❑ In artificial intelligence, **genetic programming (GP)** is a technique whereby computer programs are encoded as a set of genes that are then modified (evolved) using an evolutionary algorithm (often a genetic algorithm, "GA").
- ❑ It is an application of (for example) genetic algorithms where the space of solutions consists of computer programs.
- ❑ The results are computer programs that are able to perform well in a predefined task..



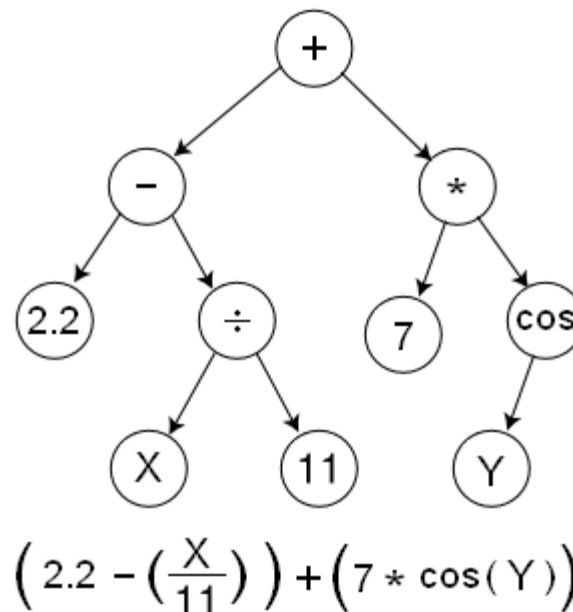
Evolutionary Algorithms

- ❑ In artificial intelligence, an **evolutionary algorithm (EA)** is a generic population-based metaheuristic optimization algorithm.
- ❑ An EA uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection.
- ❑ Candidate solutions to the optimization problem play the role of individuals in a population, and the fitness function determines the quality of the solutions (see also loss function).
- ❑ Evolution of the population then takes place after the repeated application of the above operators.



GP

- GP evolves computer programs, traditionally represented in memory as tree structures. Trees can be easily evaluated in a recursive manner. Every tree node has an operator function and every terminal node has an operand, making mathematical expressions easy to evolve and evaluate.





Ants Colony Optimization

- ❑ In the natural world, ants of some species (initially) wander randomly, and upon finding food return to their colony while laying down pheromone trails. If other ants find such a path, they are likely not to keep travelling at random, but instead to follow the trail, returning and reinforcing it if they eventually find food.
- ❑ Over time, however, the pheromone trail starts to evaporate, thus reducing its attractive strength. The more time it takes for an ant to travel down the path and back again, the more time the pheromones have to evaporate. A short path, by comparison, gets marched over more frequently, and thus the pheromone density becomes higher on shorter paths than longer ones.



ACP

- ❑ Pheromone evaporation also has the advantage of avoiding the convergence to a locally optimal solution. If there were no evaporation at all, the paths chosen by the first ants would tend to be excessively attractive to the following ones. In that case, the exploration of the solution space would be constrained.
- ❑ The influence of pheromone evaporation in real ant systems is unclear, but it is very important in artificial systems.
- ❑ The overall result is that when one ant finds a good (i.e., short) path from the colony to a food source, other ants are more likely to follow that path, and positive feedback eventually leads to many ants following a single path. The idea of the ant colony algorithm is to mimic this behavior with "simulated ants" walking around the graph representing the problem to solve.



Other EAs

- ❑ **Artificial bee colony algorithm (ABC)** is an optimization algorithm based on the intelligent foraging behaviour of honey bee swarm, proposed by Karaboga in 2005.
- ❑ **Particle swarm optimization (PSO)** was first intended for simulating social behaviour, as a stylized representation of the movement of organisms in a bird flock or fish school. The algorithm was simplified and it was observed to be performing optimization.



جستجو در محیط‌های پیوسته

- در فضاهای پیوسته، در هر حالت، تعداد حالات بعدی نامحدود است.
- تابع هدف به صورت تابعی از متغیرهای پیوسته تعریف می‌شود.
- نزول (صعود) در امتداد گرادیان شناخته شده ترین تکنیک جستجو در این محیط‌ها است.
- در هر گام به اندازه آلفا در جهت گرادیان حرکت می‌کنیم :

$$X \leftarrow X + \alpha \sum \nabla f(X)$$

- روش‌های مختلفی برای تنظیم مقدار آلفا (اندازه گام) پیشنهاد شده است.
- آنقدر در جهت گرادیان پیش می‌رویم تا به یک نقطه بهینه برسیم. این نقطه به عنوان حالت بعد در نظر گرفته می‌شود. نقاط بهینه محلی نقاطی هستند که گرادیان تابع در آنها صفر است : روش نیوتن-رافسون برای یافتن ریشه گرادیان



جستجو با اعمال غیرقطعی

❑ غیرقطعی بودن عمل‌ها:

- ✓ محیط غیرقطعی
- ✓ محیط نیمه‌قابل مشاهده
- ✓ محیط ناشناخته

❑ تفاوت با جستجو در محیط‌های قطعی:

✓ راه حل مسأله دنباله ثابتی از عمل‌ها نیست بلکه یک استراتژی از دنباله‌های مشروط به مشاهدات عامل از محیط است.

✓ خروجی هر عمل مجموعه‌ای از حالات است نه و یک حالت خاص.

❑ درخت جستجو به شکل درخت جستجوی AND-OR خواهد بود.

❑ راه حل یک درخت AND-OR زیردرختی است که

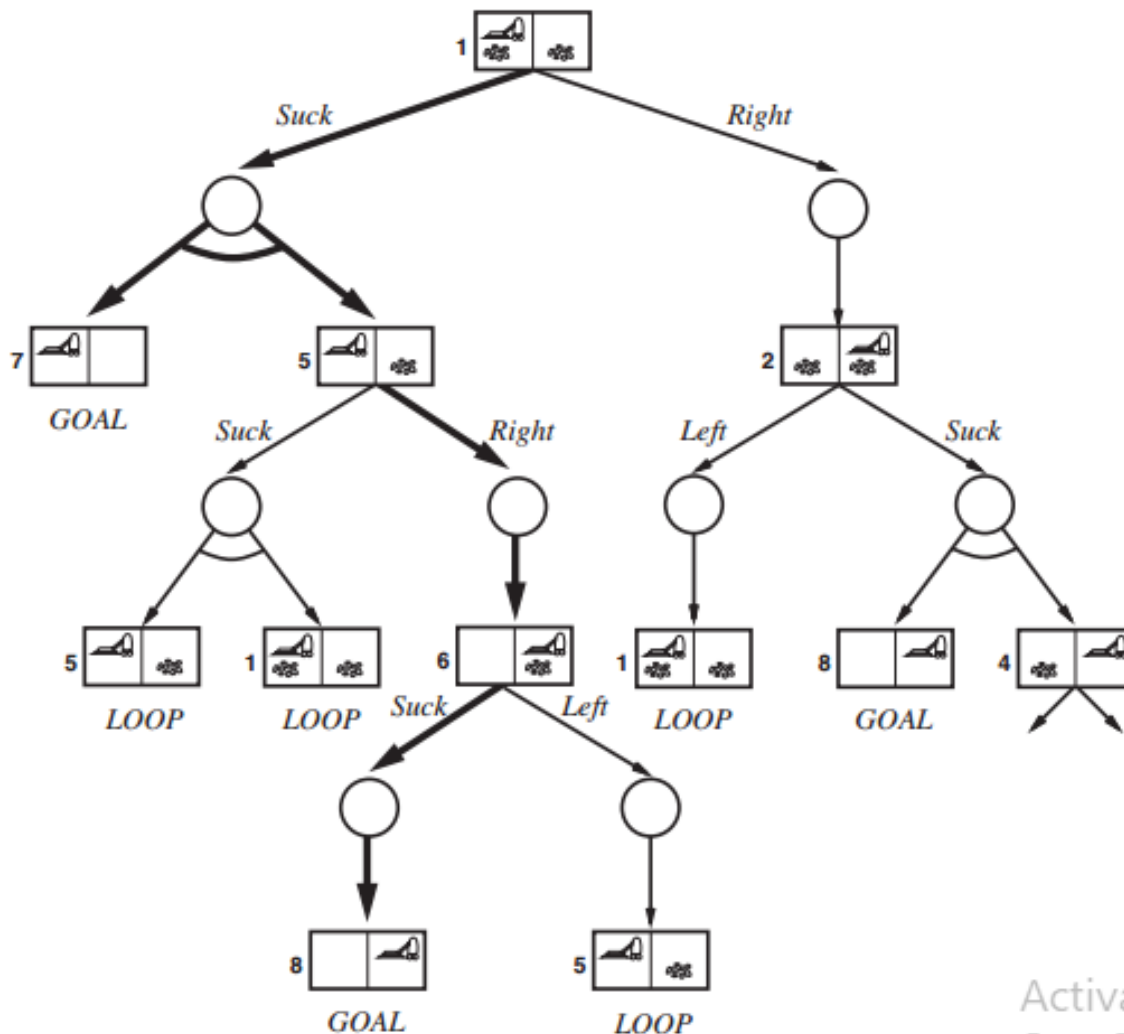
✓ همه برگ‌های آن حالت هدف باشد.

✓ در هر گره OR عمل مشخص را انتخاب کرده باشد.

✓ در هر گره AND کلیه خروجی‌های ممکن را شامل شود



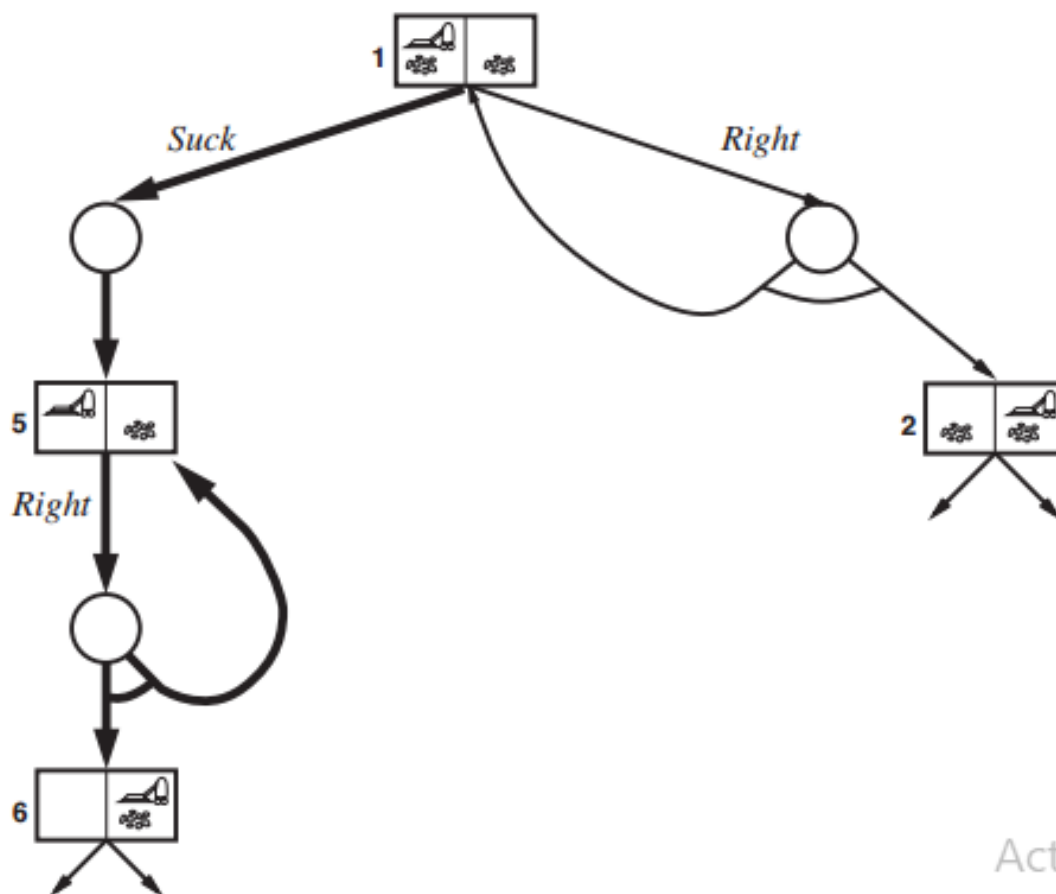
AND-OR درخت



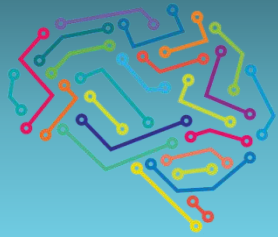
Activate Wi
Go to Settings



درخت AND-OR

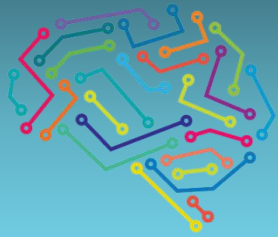


Activate
Call Center

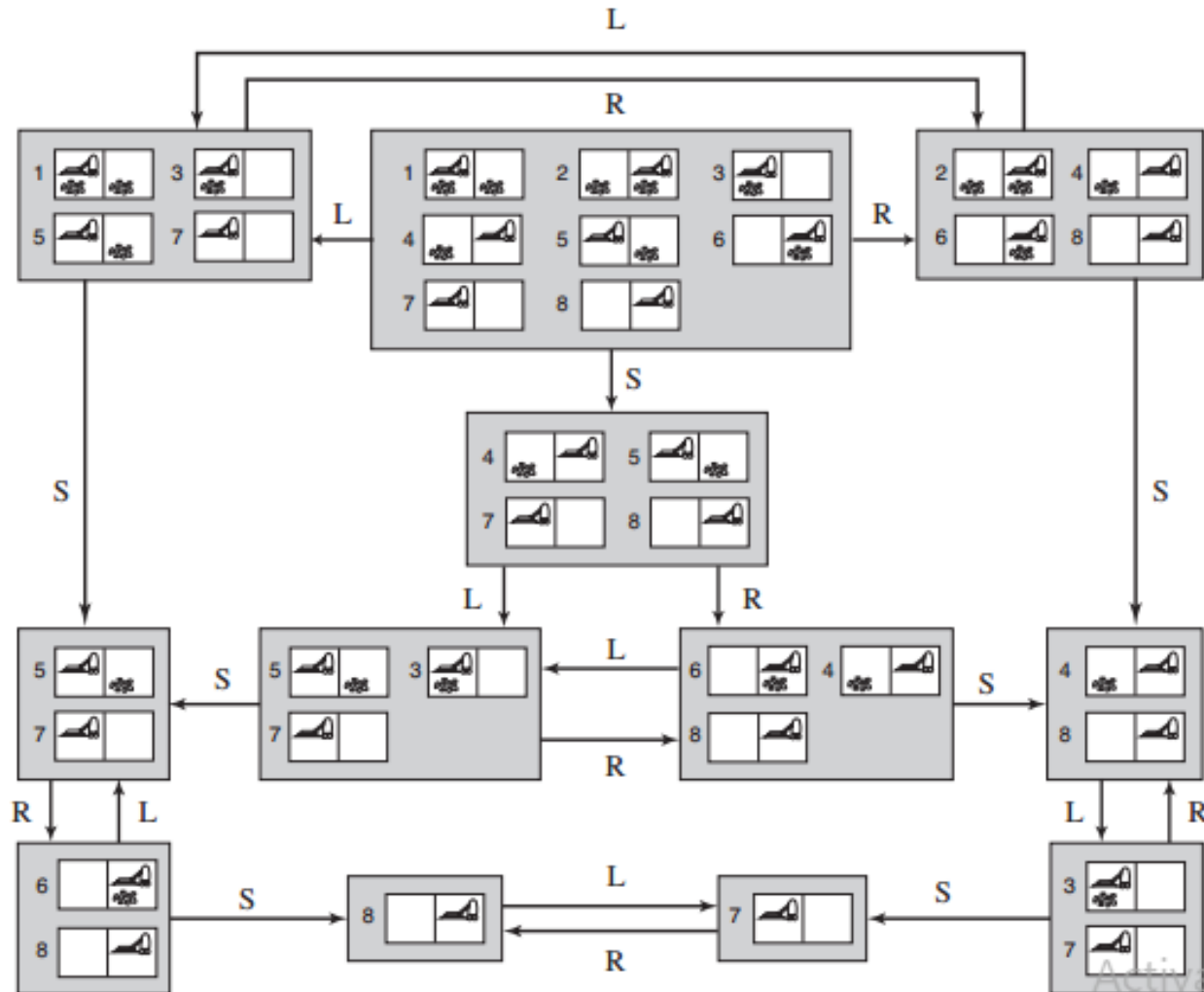


جستجو در محیط‌های نیمه‌مشاهده‌پذیر

- حتی مسائل بدون حسگر (sensorless) نیز قابل حل هستند:
 - ✓ محیطه‌هایی که در آنها عامل هیچ حسگری برای درک محیط اطراف ندارد.
 - ✓ علت: امکان حل مسأله بدون نیاز به حسگر، هزینه بالایی حسگرها و ...
- در محیط‌های نیمه‌مشاهده‌پذیر:
 - ✓ حالت محیط به صورت مجموعه‌ای از حالات ممکن یا حالات باور (belief states) بیان می‌شود.
- اندازه مجموعه حالات باور در این محیط‌ها بسیار بزرگ است.
- جستجو در فضای باور به جای جستجو در فضای حالت:
 - ✓ حالت شروع، مجموعه تمام حالات فیزیکی ممکن است.
 - ✓ اعمال ممکن در هر حالت باور، اشتراک تمام اعمال ممکن در حالات فیزیکی موجود در آن حالت باور است.
 - ✓ مجموعه حالات فیزیکی ممکن در اثر انجام هر یک از اعمال مذکور، حالت باور بعدی را می‌سازند.
 - ✓ حالت باوری، حالت هدف محسوب می‌شود که تمام حالات فیزیکی موجود در آن، حالت هدف باشد.



Belief search





جستجوی آنلاین

□ جستجوی آفلاین (برون خط):

✓ اجرای کامل فرآیند جستجو و یافتن راه حل کامل پیش از شروع اجرای آن در دنیای واقعی.

□ جستجوی آنلاین (برخط):

✓ جستجو و اجرای همزمان راه حل

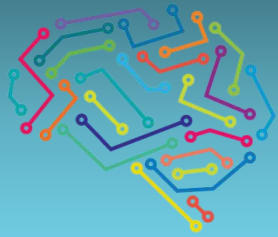
✓ تا زمانی که در موقعیت جاری قرار نگرفته‌ایم، حالات بعدی را نمی‌دانیم

✓ مناسب برای :

▪ محیط‌های پویا یا نیمه پویا : مواردی که گذشت زمان هزینه (جریمه) دارد

▪ محیط‌های نامعین

▪ محیط‌های ناشناخته (مسأله اکتشاف : exploration problem)



جستجوی آنلاین

□ جستجوی آنلاین:

✓ در هر لحظه، عامل در نقطه‌ای از محیط واقعی قرار دارد و تنها می‌تواند جستجوی خود را به صورت محلی از این نقطه دنبال کند.

✓ برای ادامه جستجو از بخش دیگری از محیط لازم است عامل به صورت فیزیکی به آن حالت برگردد.

□ جستجو باید ماهیت محلی داشته باشد، چون نمی‌توان جستجو را در لحظه جاری به موقعیت غیرشناخته شده‌ای از فضای حالت منتقل کرد.

□ الگوریتم‌های قابل استفاده:

✓ جستجوی اول عمق آنلاین

✓ تپه نوردی با گردش تصادفی یا حافظه

✓ جستجوی A^* یادگیرنده real-time ($LRTA^*$)



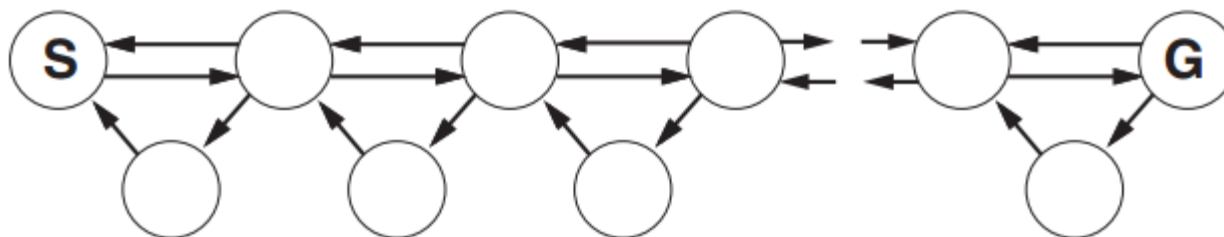
جستجوی آنلاین

□ تپه نوردی آنلاین:

✓ الگوریتم تپه نوردی به دلیل نگهداری تنها، حالت فعلی، فی نفسه يك الگوریتم جستجوی آنلاین است.

✓ ولی ناکارآمد است، چون حالت شروع مجدد تصادفی به دلیل عدم اطلاع از نقشه محیط، قابل پیاده سازی نیست.

✓ می توان از random walk برای اکتشاف محیط استفاده کرد ولی ممکن است خیلی زمان بر باشد.





جستجوی آنالین

□ صرف حافظه بیشتر، جواب بهتری می‌دهد.

□ در توسعه مبتنی بر حافظه این الگوریتم، مقدار $H(s)$ یعنی بهترین تخمین الگوریتم برای رسیدن به هدف از حالت s ، بر اساس تجربیات عامل در محیط واقعی بروز می‌شود.

□ مقدار اولیه $H(s)$ برابر با مقدار حاصل از یک تابع هیوریستیک قرار داده می‌شود.

LRTA*



LRTA*

