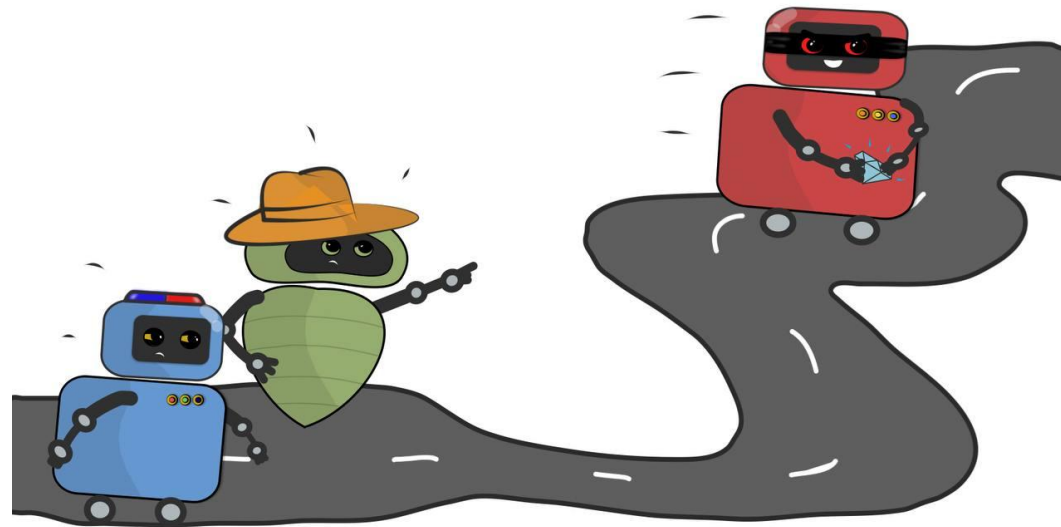


مبانی و کاربردهای هوش مصنوعی

مسائل ارضای محدودیت 2 (فصل 6.1 الی 6.5)



مدرس: مهدی جوانمردی

دانشکده مهندسی کامپیوتر دانشگاه صنعتی امیرکبیر

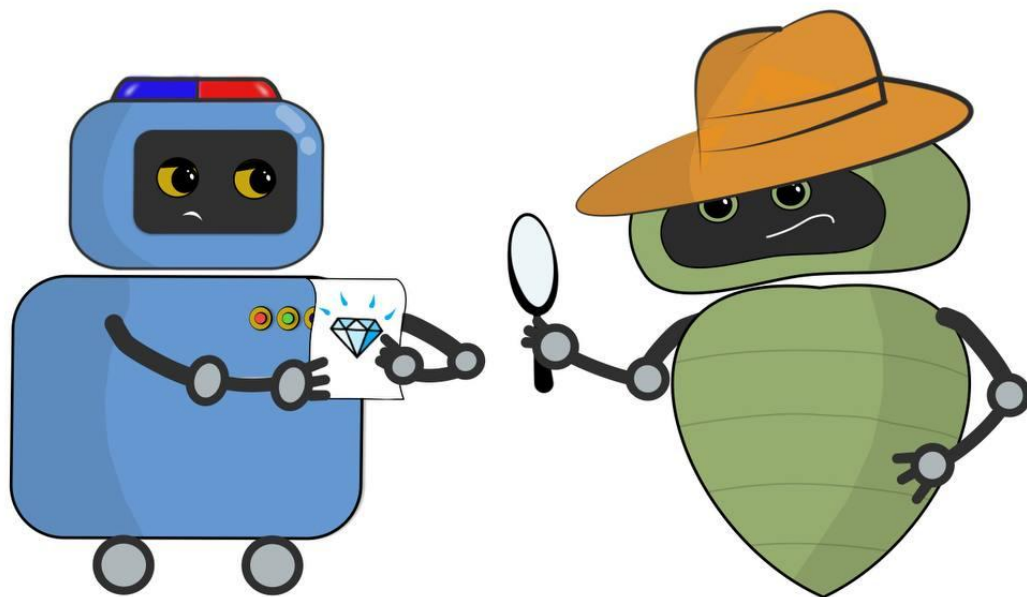


(الهام گرفته از محتوای درس هوش مصنوعی دانشگاه برکلی)

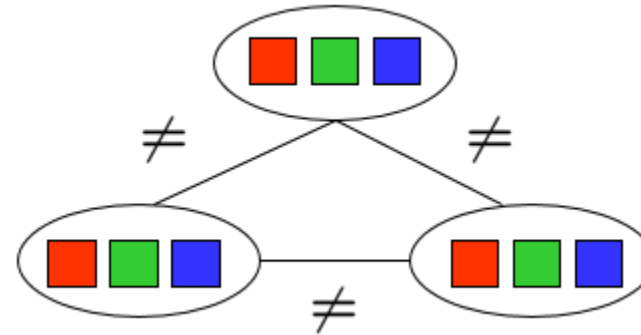
امروز

- راه حل کارآمد برای CSP ها

- جستجوی محلی



یادآوری: CSP ها



• CSP ها:

- متغیرها
- دامنه‌ها
- محدودیت‌ها

• ضمنی (ارائه کد برای محاسبه)

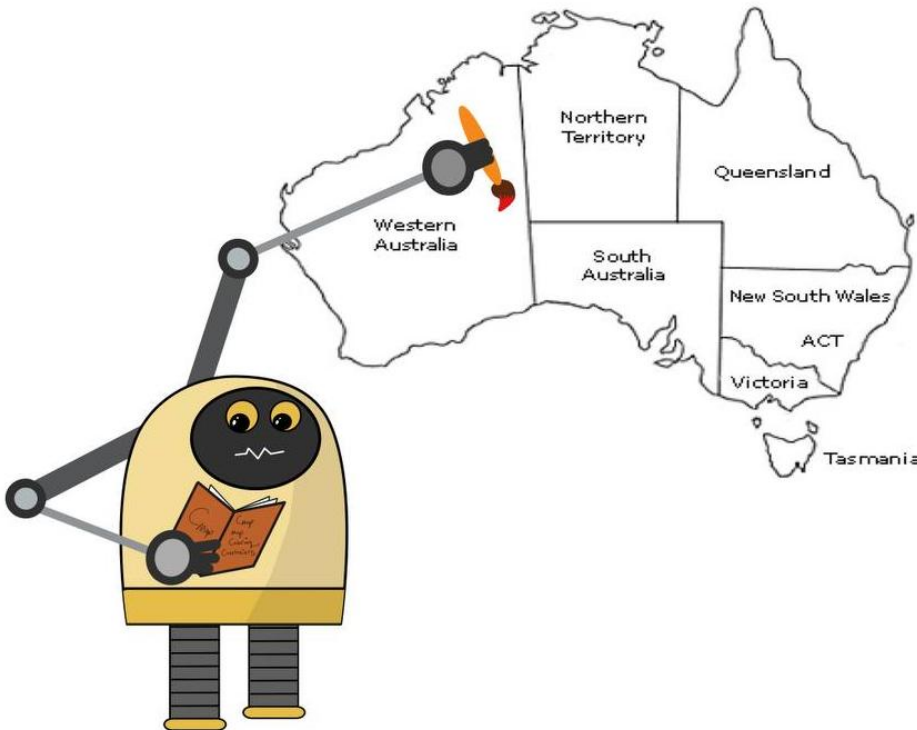
• صریح (لیستی از تخصیص‌های مجاز ارائه کنید)

• Unary / Binary / N-ary

• اهداف:

• اینجا: هر راه‌حلی پیدا کنید.

• همچنین: همه راه‌حل‌ها را پیدا کنید، بهترین را پیدا کنید و غیره.

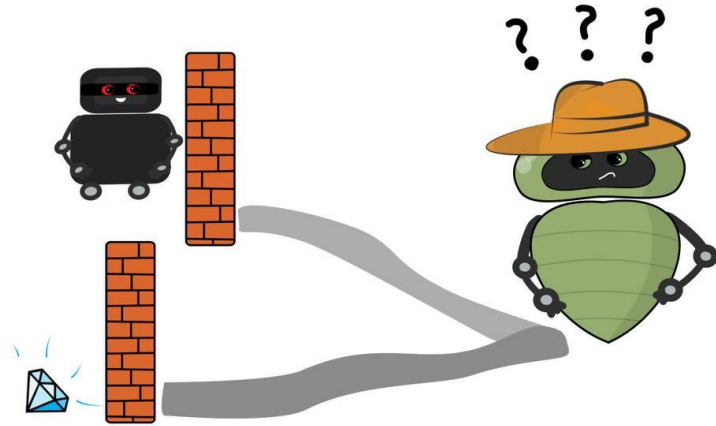


جستجوی عقبگرد

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
  return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
    if value is consistent with assignment given CONSTRAINTS[csp] then
      add {var = value} to assignment
      result ← RECURSIVE-BACKTRACKING(assignment, csp)
      if result ≠ failure then return result
      remove {var = value} from assignment
  return failure
```

بهبود عقبگرد

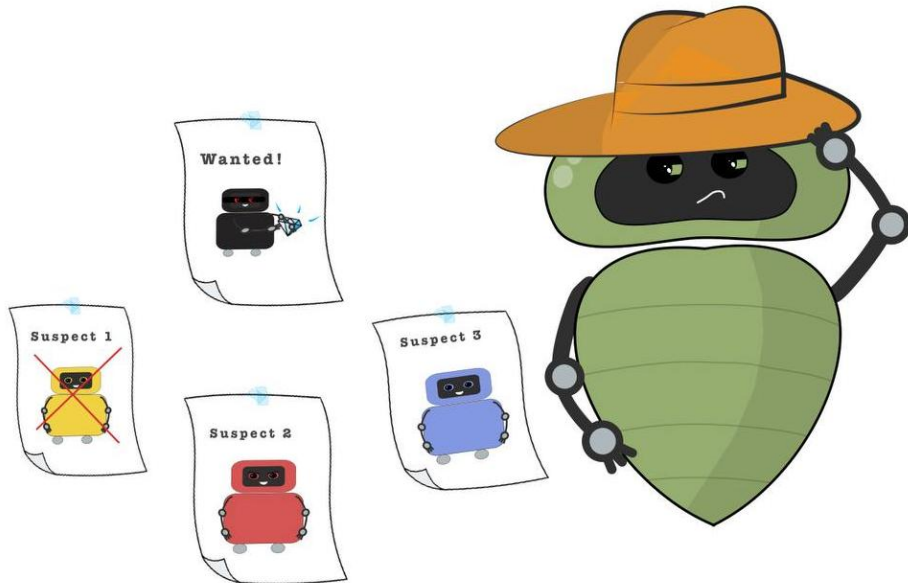


- ایده‌های همه منظوره بهبود قابل توجهی در سرعت به همراه دارند
- ... اما همه چیز هنوز NP-hard است

- فیلتر کردن: آیا می‌توانیم شکست اجتناب ناپذیر را زود تشخیص دهیم؟

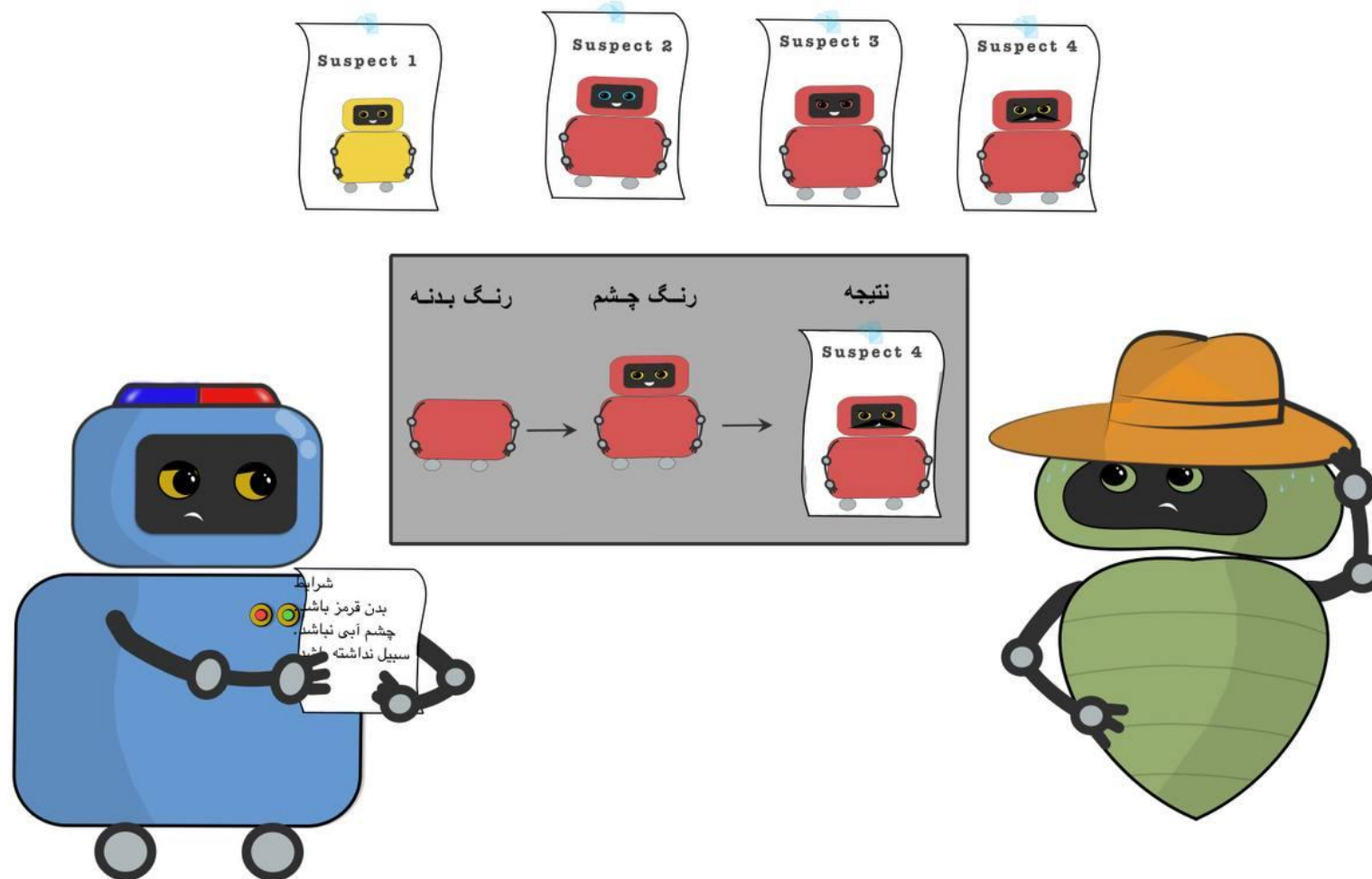
- مرتب‌سازی:

- کدام متغیر باید در مرحله بعد مقداردهی شود؟ (MRV)
- مقادیر ممکن آن را به چه ترتیبی باید امتحان کرد؟ (LCV)



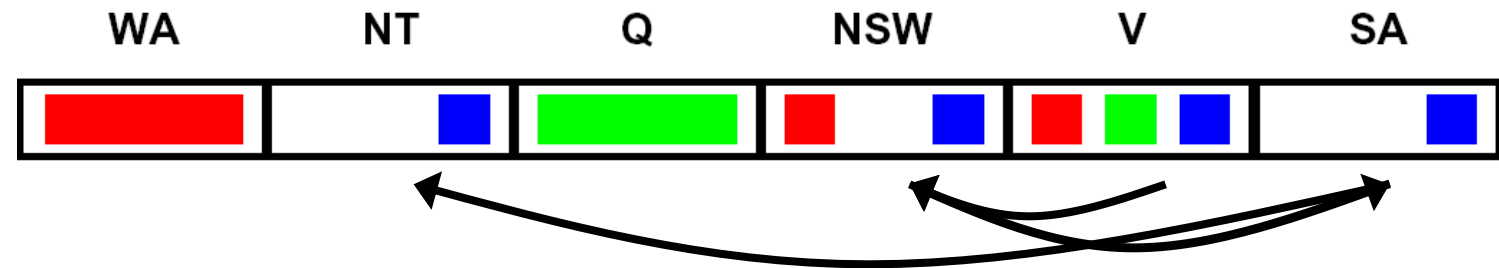
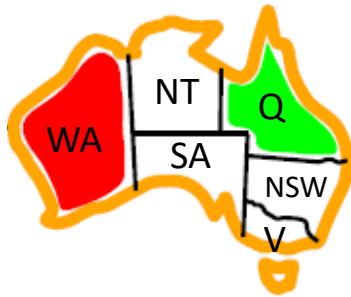
- ساختار: آیا می‌توانیم از ساختار مسئله بهره برداری کنیم؟

سازگاری یال و فراتر از آن



سازگاری یال یک CSP کامل

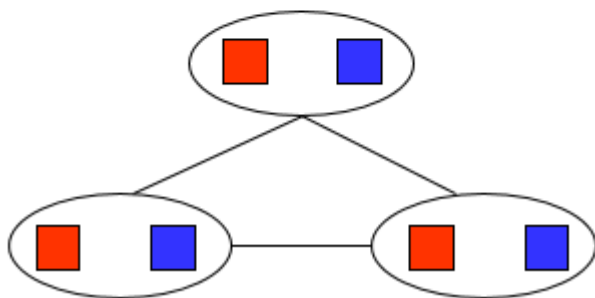
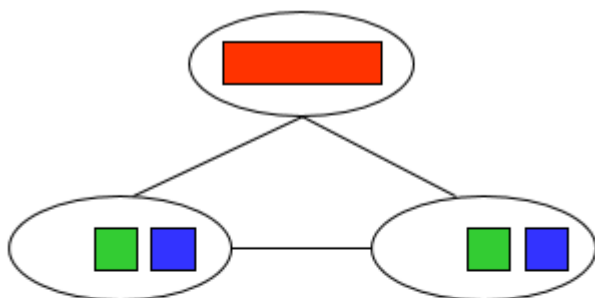
- یک طریقه ساده از انتشار، اطمینان حاصل می‌کند که تمام یال‌ها سازگار هستند:



به یاد داشته
باشید: حذف از دم!

- سازگاری یال، شکست را زودتر از بررسی رو به جلو تشخیص می‌دهد.
- مهم: اگر X مقداری را از دست بدهد، همسایگان X باید دوباره بررسی شوند!
- باید بعد از هر انتساب دوباره اجرا شود!

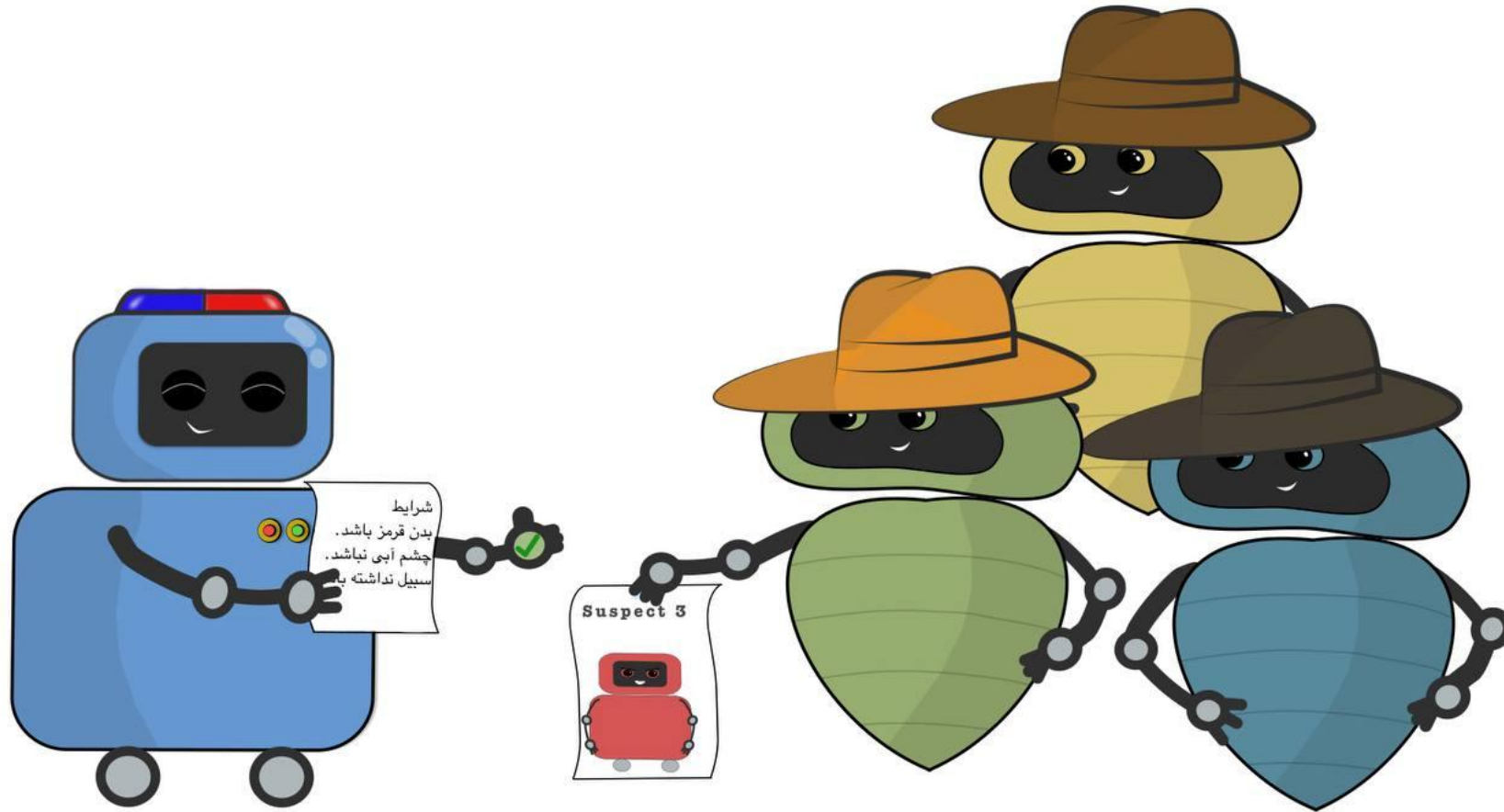
محدودیت‌های سازگاری یال



اینجا چه اشتباهی
رخ داده است؟

- پس از اعمال سازگاری یال:
 - می‌تواند یک راه‌حل باقی بماند.
 - می‌تواند چندین راه‌حل باقی‌مانده باشد.
 - می‌تواند هیچ راه‌حلی باقی نمانده باشد (و الگوریتم جستجو نداند)
- سازگاری یال همچنان در داخل جستجوی عقبگرد اجرا می‌شود!

سازگاری K تایی (K-Consistency)



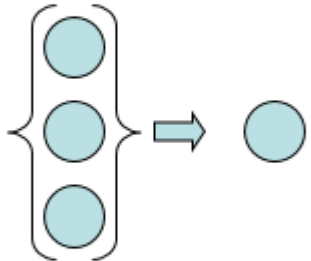
سازگاری K تایی

- افزایش درجات سازگاری:

- سازگاری 1 تایی (سازگاری گره): دامنه هر گره منفرد دارای مقداری است که با محدودیت‌های یگانی (Unary Constraints) آن گره مطابقت دارد



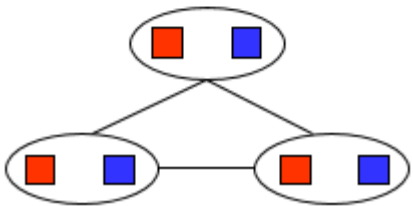
- سازگاری 2 تایی (سازگاری یال): برای هر جفت گره، هر تخصیص سازگار به یکی از گره‌ها را می‌توان به دیگری تعمیم داد



- سازگاری K تایی: برای هر k گره، هر تخصیص سازگار به $k-1$ گره را می‌توان به k ام تعمیم داد

- هر چقدر k بیشتر باشد، هزینه محاسبات بیشتر می‌شود

- (شما حالت $k=2$ را یاد گرفتید: سازگاری یال)



سازگاری K تایی قوی (Strong K-Consistency)

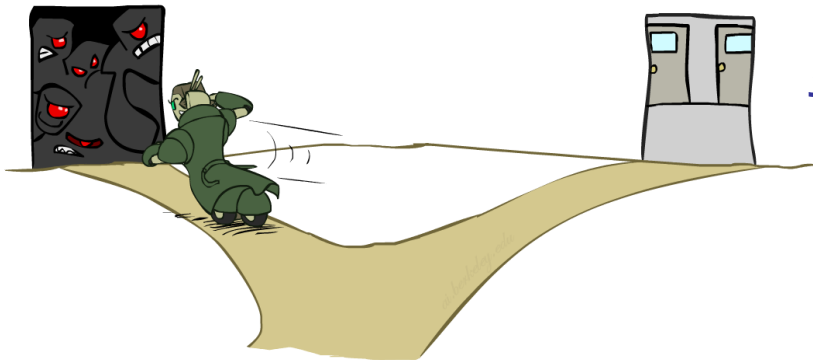
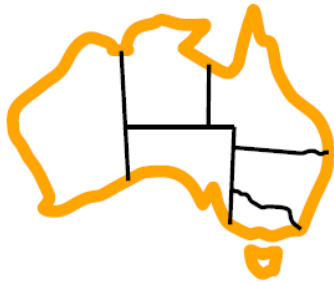
- سازگاری k تایی قوی: علاوه بر k، برای 1, ..., k-2, k-1 سازگاری وجود داشته باشد
- ادعا: سازگاری n تایی قوی به این معنی است که ما می‌توانیم بدون عقبگرد (Backtracking) مسئله را حل کنیم!
- چرا؟
 - هر انتساب به هر متغیری را انتخاب کنید
 - یک متغیر جدید انتخاب کنید
 - با سازگاری 2 تایی، حداقل یک انتخاب سازگار با متغیر اولی وجود دارد
 - یک متغیر جدید انتخاب کنید
 - با سازگاری 3 تایی، حداقل یک انتخاب سازگار با دو متغیر قبلی وجود دارد
 - ...
- روش‌های میانی زیادی بین سازگاری یال و سازگاری n تایی وجود دارد!
(به عنوان مثال $k=3$ ، سازگاری مسیر نامیده می‌شود)

مرتب‌سازی (Ordering)



مرتب‌سازی: حداقل مقادیر باقی‌مانده (Minimum Remaining Values)

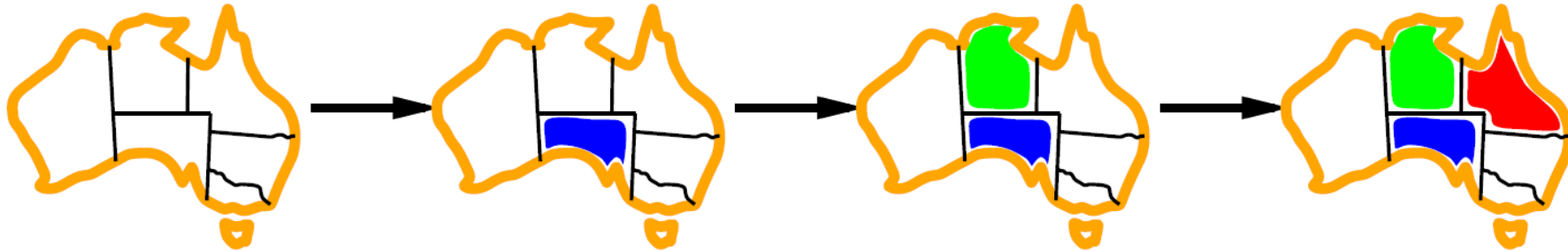
- ترتیب متغیر: هیوریستیک حداقل مقادیر باقی‌مانده (MRV)
- متغیری را با کمترین مقادیر ممکن باقی‌مانده در دامنه خود انتخاب کنید



- چرا حداقل به جای حداکثر؟
- همه متغیرها حتما باید مقداردهی شوند: در نتیجه از متغیر سخت‌تر شروع کن
- همچنین "محدود ترین متغیر" (Most Constrained Variable) نامیده می‌شود
- مرتب‌سازی "شکست-به سرعت" (Fail-Fast Ordering)
- کمکی: هیوریستیک درجه (degree heuristic)
- در هنگام برابری MRV انتخاب متغیر دارای بیشترین محدودیت با دیگر متغیرها

هیوریستیک درجه‌ای (Degree Heuristic)

- مورد استفاده در صورت وجود چند متغیر MRV
- هیوریستیک درجه‌ای:
- متغیری را انتخاب کنید که در بیشترین محدودیت‌ها بر متغیرهای باقی‌مانده شرکت دارد



- چرا بیشتر به جای کمترین محدودیت‌ها؟

مقدار اعمال کننده کمترین محدودیت (Least Constraining Value)

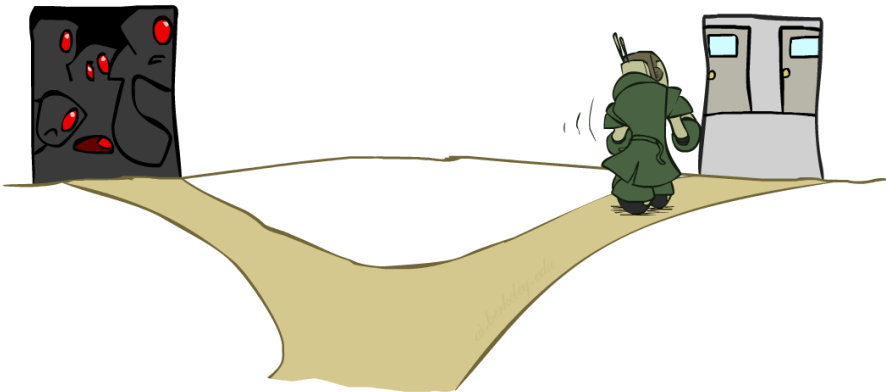
- ترتیب مقدار: مقدار اعمال کننده کمترین محدودیت

- با توجه به انتخاب متغیر، مقداری را انتخاب کنید که حداقل محدودیت را به دیگر متغیرها اعمال کند
- یعنی مقداری که با تخصیص آن، کمترین تعداد مقادیر از بین مقادیر ممکن برای متغیرهای باقی مانده خط می خورند
- توجه داشته باشید که برای تعیین این مقدار ممکن است مقداری محاسبات لازم باشد!
(به عنوان مثال، اجرای مجدد فیلترینگ)

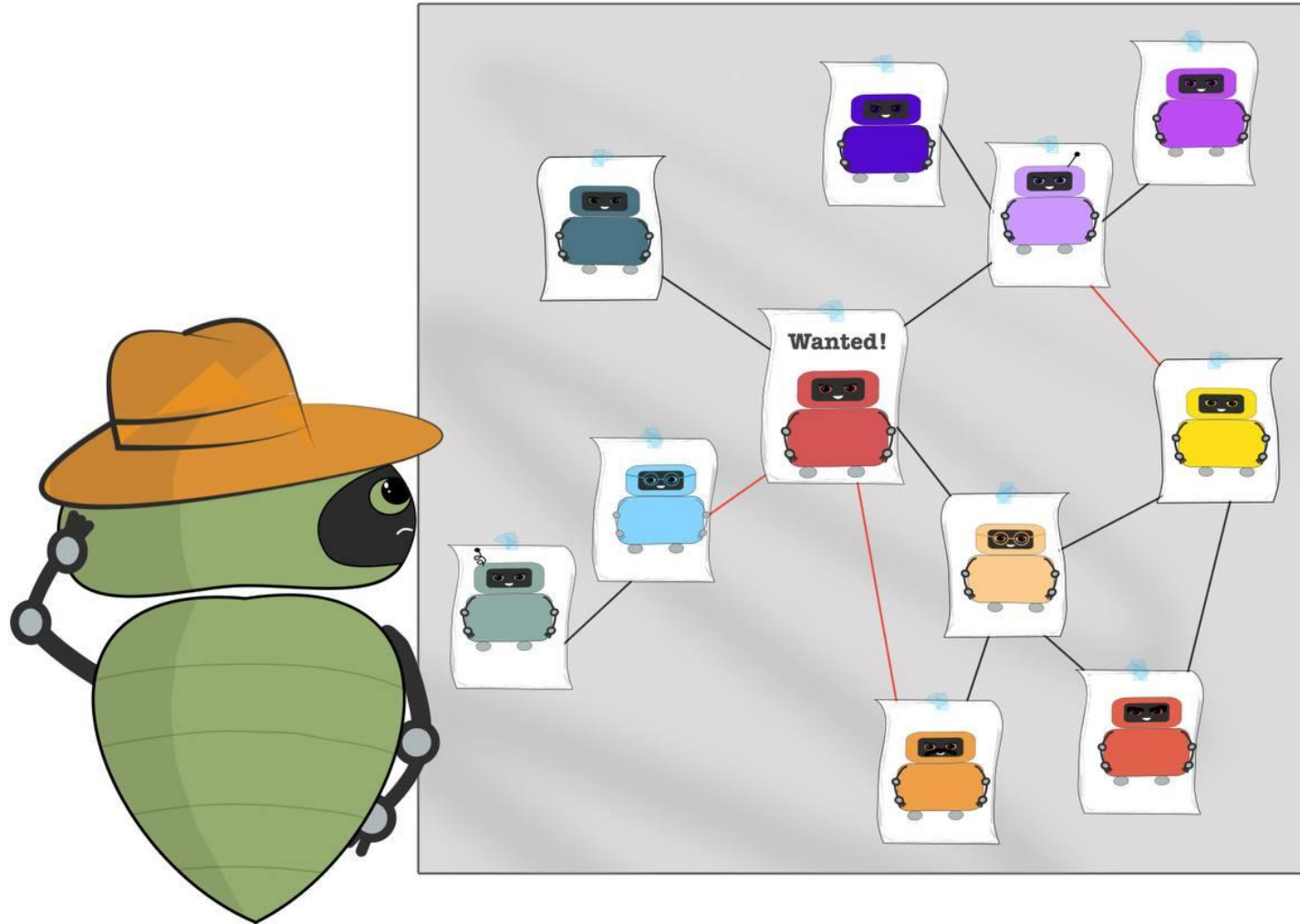
- چرا حداقل به جای حداکثر؟

- همه مقادیر یک متغیر ممکن است لازم نباشد بررسی شود
- در نتیجه از مقادیر آسان تر شروع کن

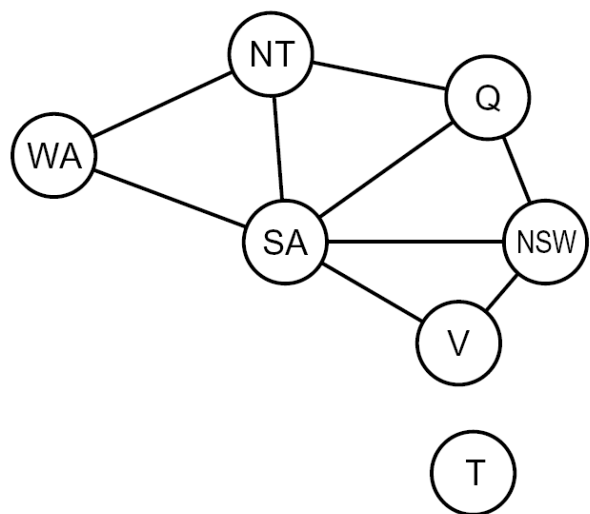
- ترکیب این ایده های مرتب سازی، حل 1000 وزیر را امکان پذیر می کند



ساختار (Structure)

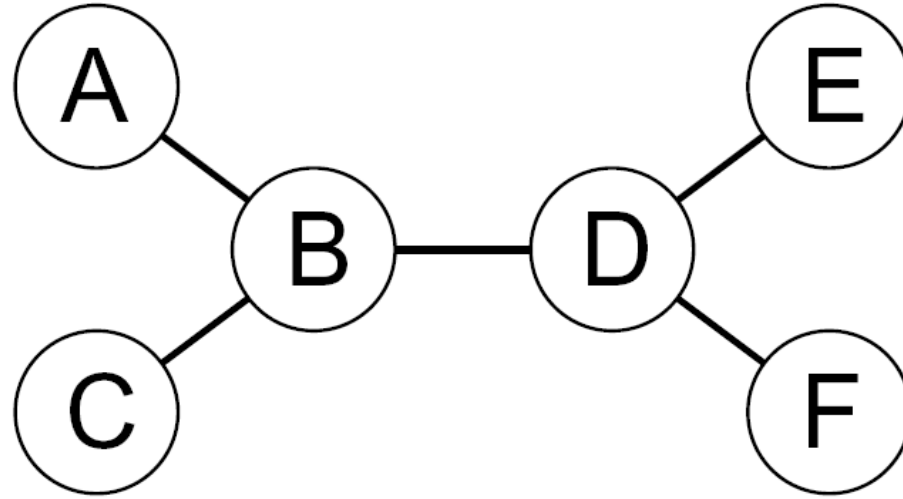


ساختار مسئله



- حالت ویژه: زیرمسائل مستقل
 - مثال: تاسمانی (T) و سرزمین اصلی تعامل ندارند
- زیرمسائل مستقل به عنوان اجزای متصل گراف محدودیت قابل شناسایی هستند
- فرض کنید نموداری از n متغیر را می‌توان به زیرمسئله‌های تنها c متغیره تقسیم کرد:
 - هزینه راه‌حل در بدترین حالت $O(\frac{n}{c} d^c)$ است، که نسبت به n خطی است
 - مقایسه با هزینه CSPهای عمومی، که در آن بدترین زمان $O(d^n)$ است
 - به عنوان مثال، $c = 20$ ، $d = 2$ ، $n = 80$
 - $2^{80} = 4$ میلیارد سال با سرعت محاسبه 10 میلیون گره در ثانیه
 - $0.4 = 4 \times 2^{20}$ ثانیه با سرعت محاسبه 10 میلیون گره در ثانیه

CSPهای با ساختار درختی (Tree-Structured CSPs)

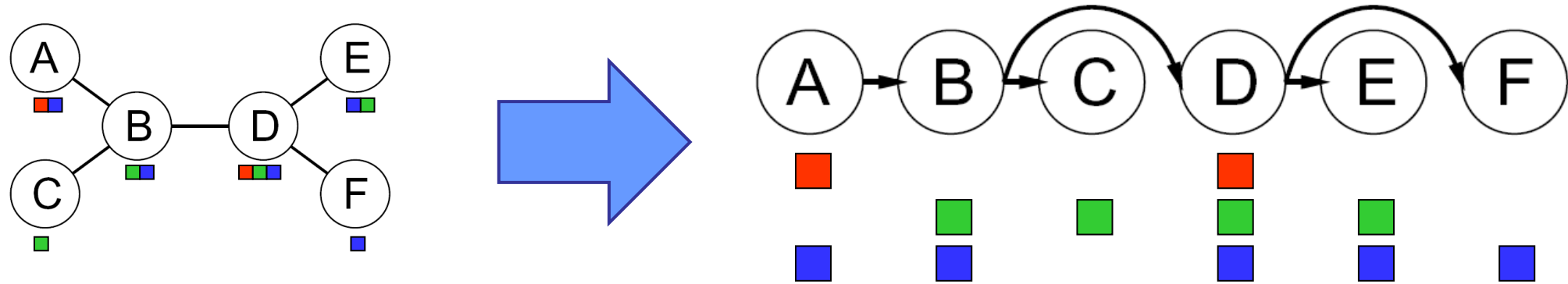


- قضیه: اگر گراف محدودیت فاقد حلقه باشد، CSP را می‌توان در زمان $O(n.d^2)$ حل کرد
 - مقایسه با هزینه CSPهای عمومی، که در آن بدترین زمان $O(d^n)$ است
- این ویژگی در مورد استنتاج احتمالی نیز صدق می‌کند (مباحث بعدی):
نمونه‌ای از رابطه بین محدودیت‌های نحوی (syntactic restrictions) و پیچیدگی استدلال

CSP های با ساختار درختی

- الگوریتم برای CSP های دارای ساختار درختی:

- ترتیب: یک متغیر ریشه را انتخاب کنید، متغیرها را طوری مرتب کنید که والدین بر فرزندان مقدم باشند (topological sort)



- حذف کردن رو به عقب:

For $i = n : 2$, apply $\text{RemoveInconsistent}(\text{Parent}(X_i), X_i)$

- تخصیص رو به جلو:

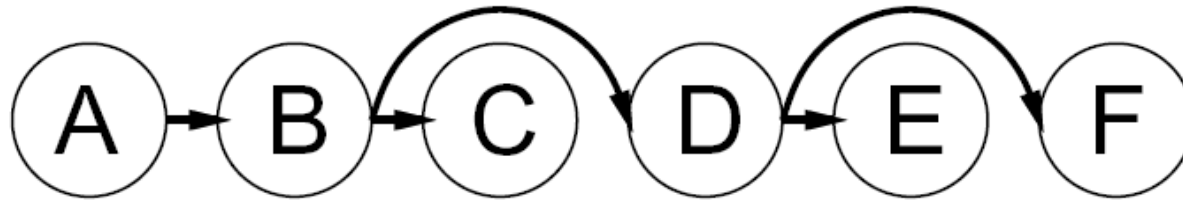
For $i = 1 : n$, assign X_i consistently with $\text{Parent}(X_i)$

- زمان اجرا: $O(n.d^2)$ (چرا؟)

- چرا این الگوریتم درست کار میکند؟

CSP های با ساختار درختی

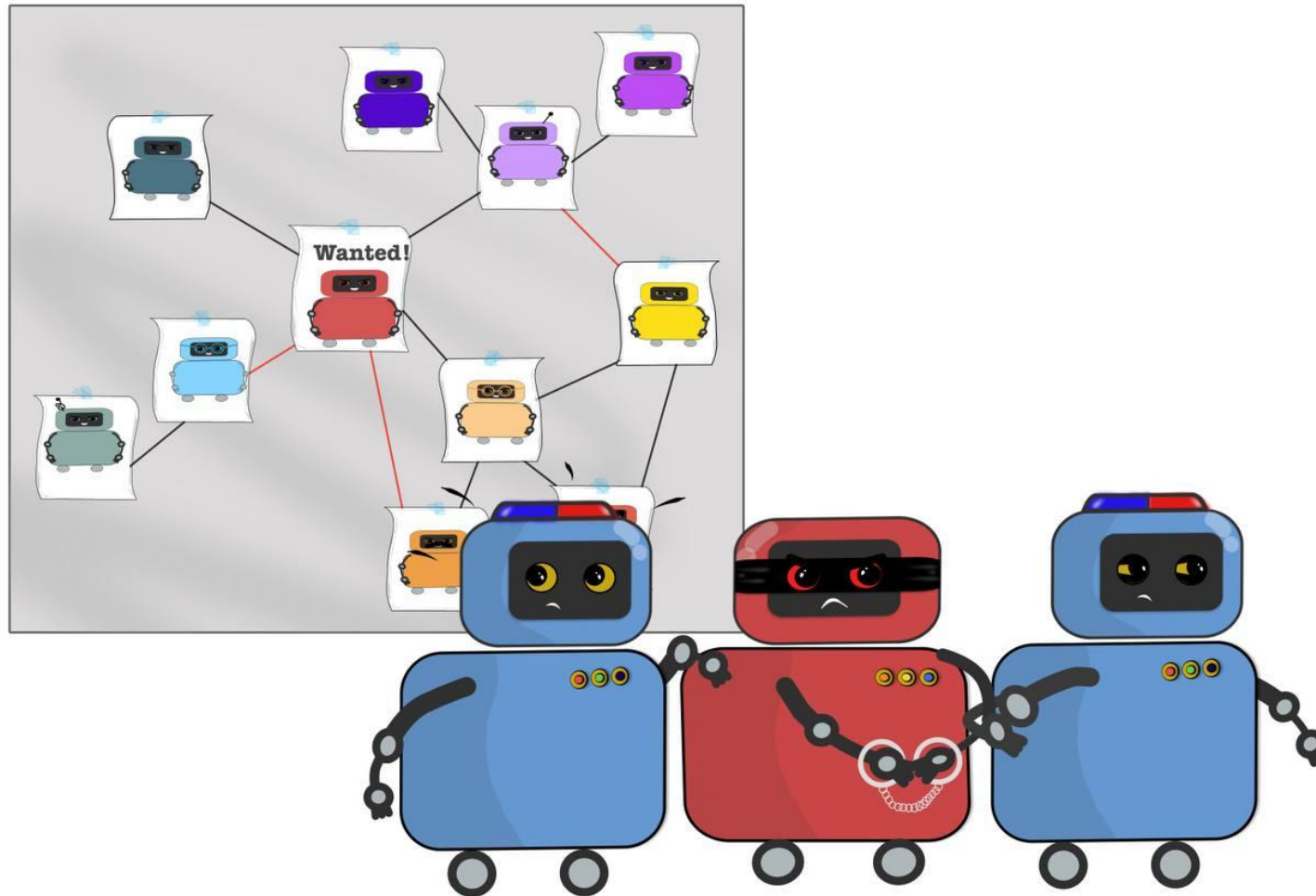
- ادعای 1: پس از پیمایش رو به عقب، تمام یال‌های ریشه به برگ سازگار هستند
- اثبات: هر $X \rightarrow Y$ در یک نقطه از پیمایش سازگار شده بود و دامنه Y نمی‌توانست پس از آن کاهش یابد (زیرا فرزندان Y قبل از Y پردازش شده بودند)



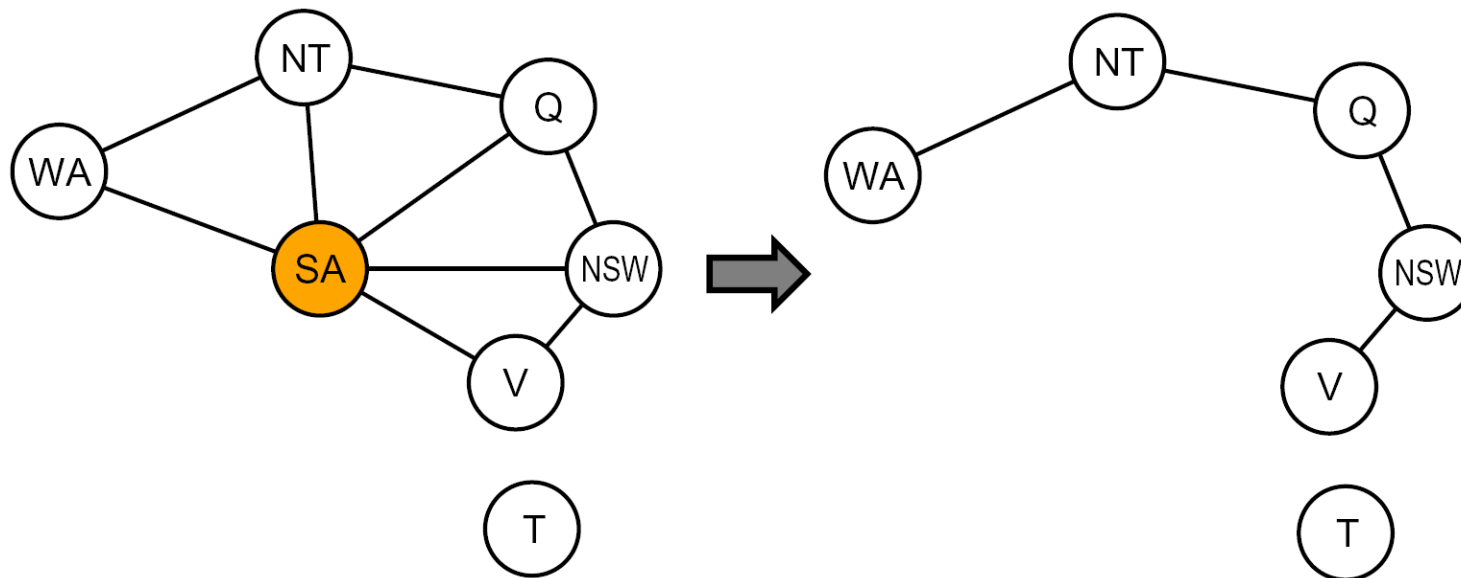
- ادعای 2: اگر یال‌های ریشه به برگ سازگار باشند، تخصیص رو به جلو عقبگرد (Backtrack) نخواهد داشت.

- چرا این الگوریتم با دوره‌های گراف محدودیت کار نمی‌کند؟
- توجه: این ایده اصلی را دوباره با شبکه‌های بیز خواهیم دید.

بهبود ساختار



CSP های تقریباً با ساختار درختی



- شرطی‌سازی (Conditioning): یک متغیر را مقداردهی کنید، دامنه‌ی همسایه‌های آن را هرس کنید
- شرطی‌سازی Cycle Cutset : مقداردهی مجموعه‌ای از متغیرها به گونه‌ای که گراف محدودیت باقی‌مانده یک درخت باشد (امتحان کردن همه‌ی حالات)
- Cutset به اندازه c ، زمان اجرای $O(d^c(n - c)d^2)$ را دارد، برای c های کوچک بسیار سریع است

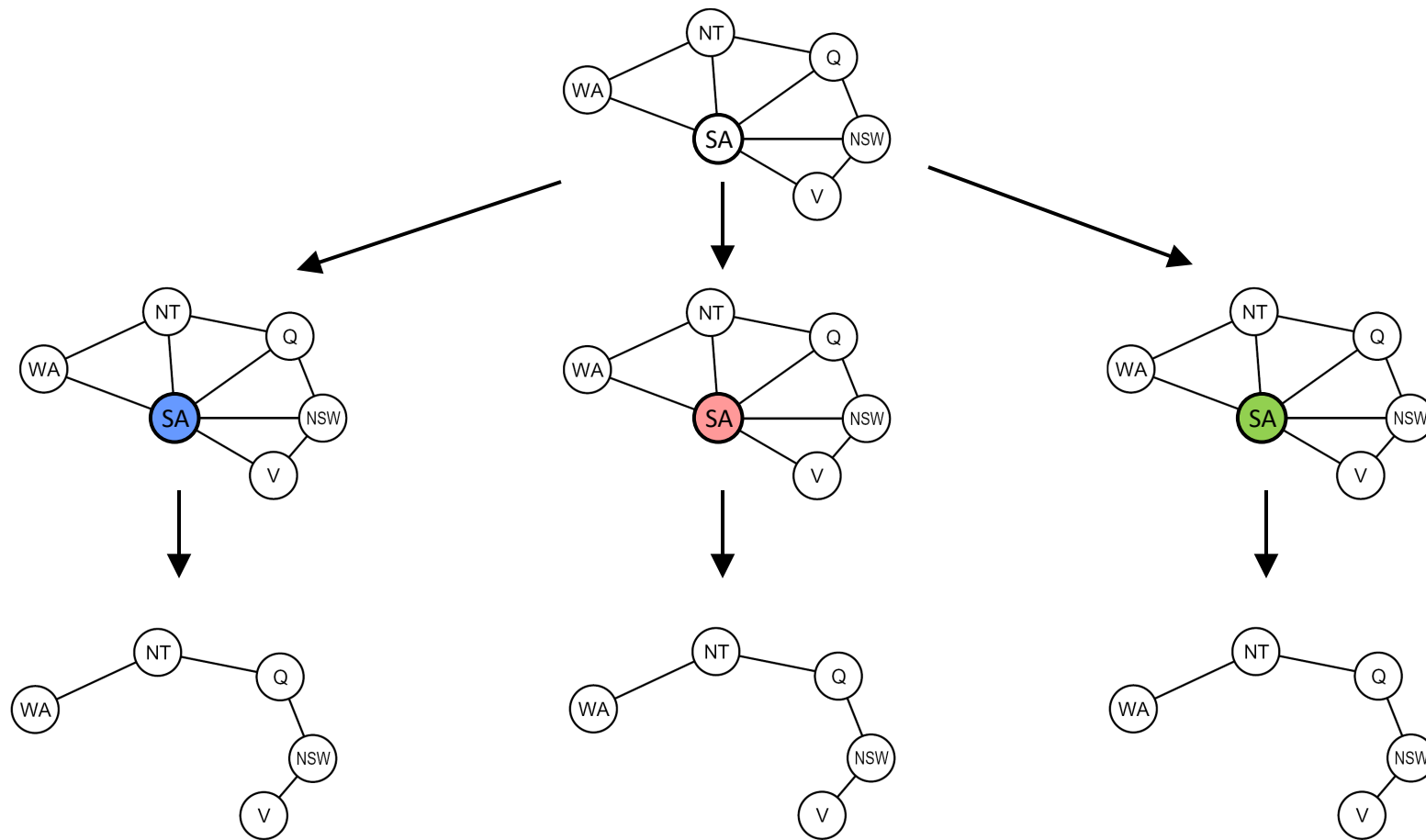
شرطی سازی Cutset (Cutset Conditioning)

یک cycle cutset انتخاب کنید

مقداردهی cutset
(امتحان کردن تمام حالات)

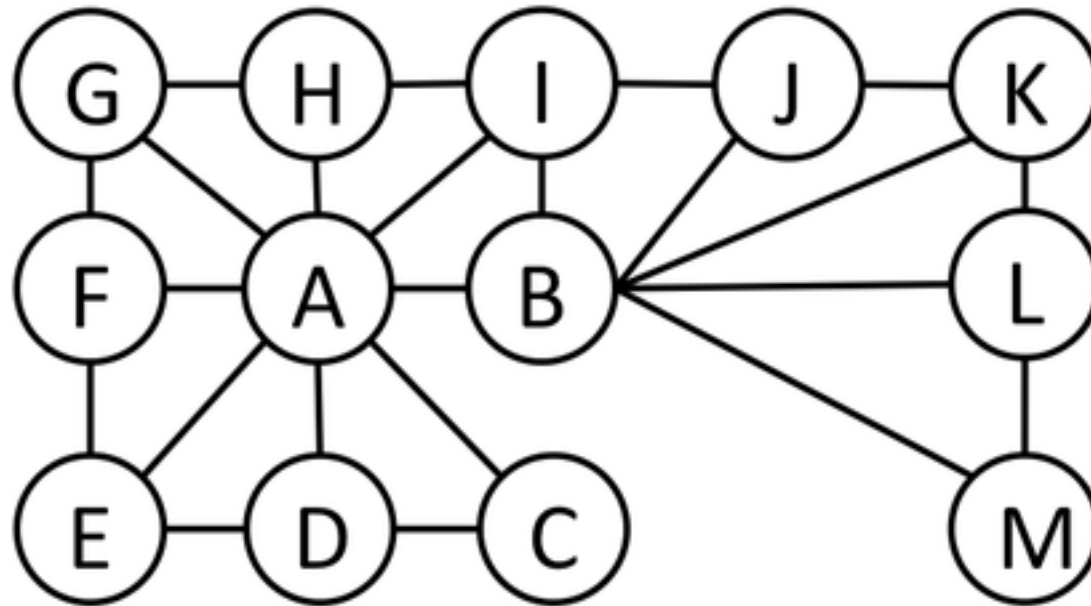
CSP باقی مانده را برای هر
انتساب محاسبه کنید

CSP های باقی مانده را حل کنید
(ساختار درختی)

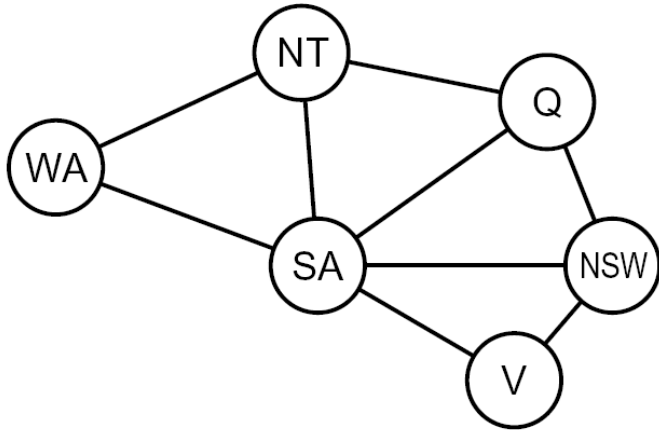


آزمونک Cutset

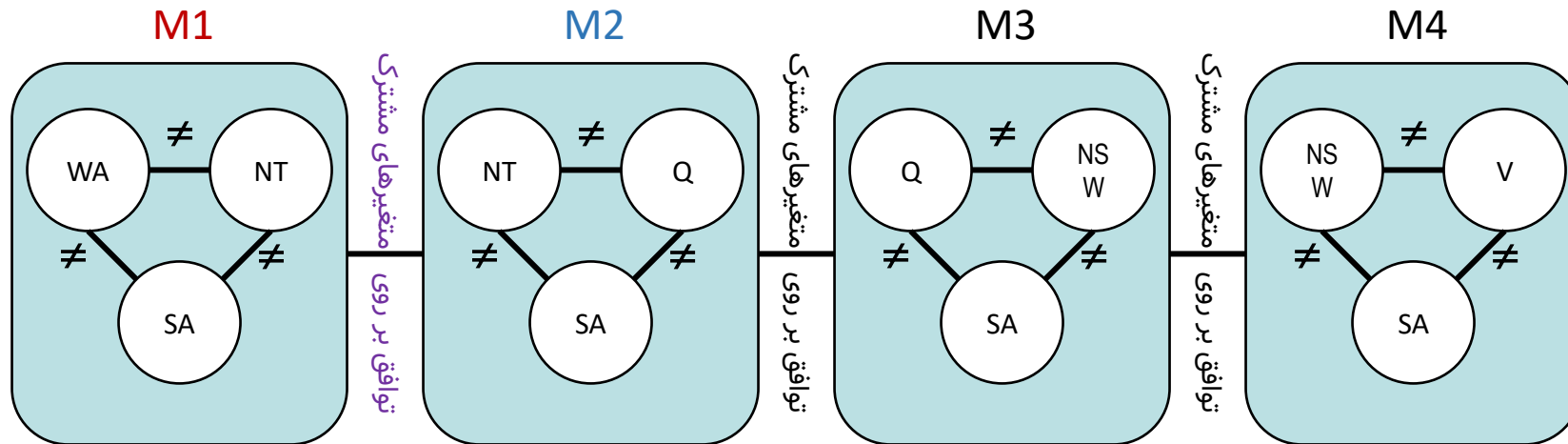
- کوچکترین cycle cutset را برای گراف زیر پیدا کنید •



تجزیه درخت* (Tree Decomposition)



- ایده: یک گراف با ساختار درختی از متغیرهای بزرگ ایجاد کنید
- هر متغیر بزرگ بخشی از CSP اصلی را رمزگذاری می‌کند
- زیرمسئله‌ها برای تضمین راه‌حل‌های سازگار، هم‌پوشانی دارند

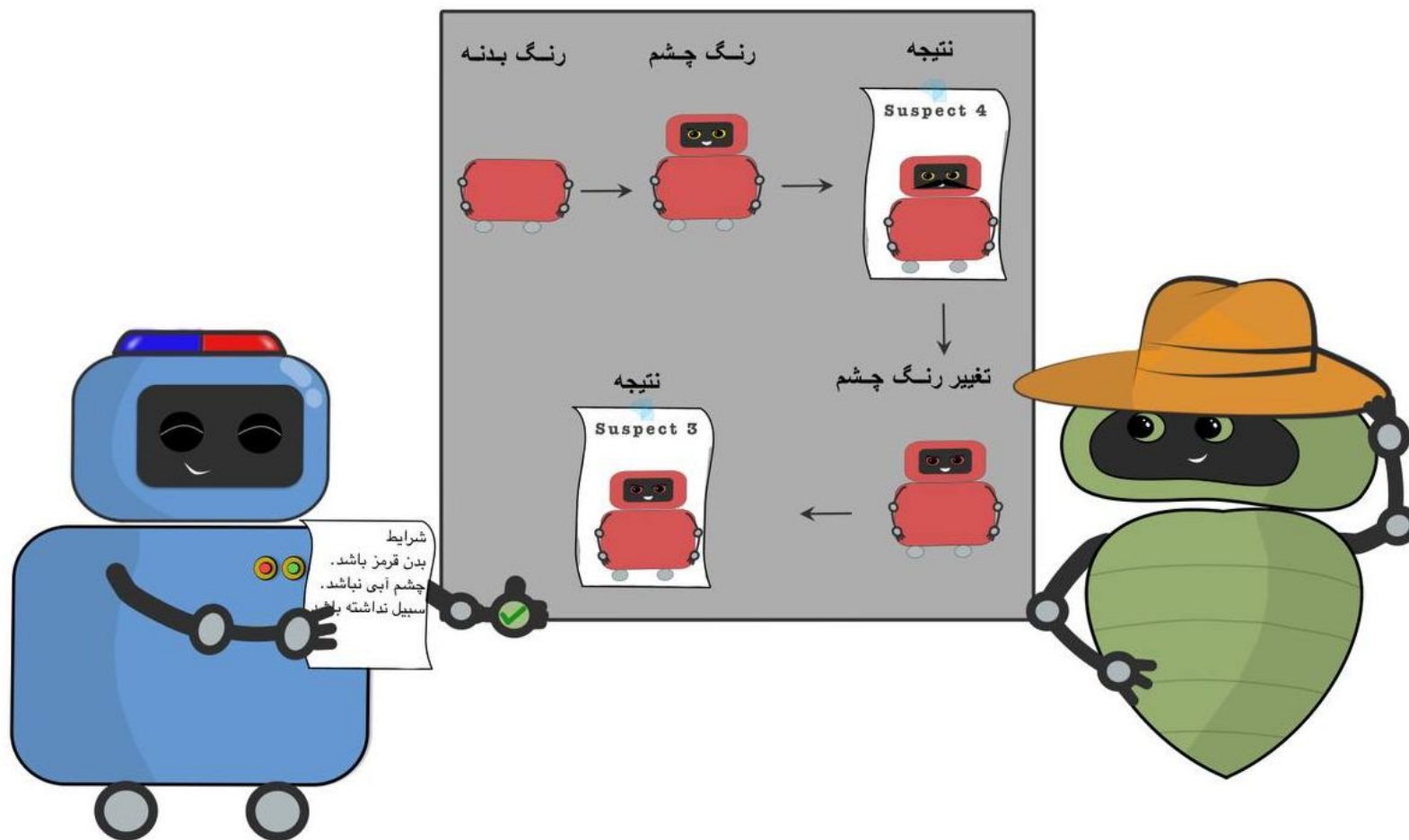


$\{(WA = r, SA = g, NT = b),$
 $(WA = b, SA = r, NT = g),$
 $\dots\}$

$\{(NT = r, SA = g, Q = b),$
 $(NT = b, SA = g, Q = r),$
 $\dots\}$

Agree (توافق): $(M1, M2) \in$
 $\{((WA = g, SA = g, NT = g), (NT = g, SA = g, Q = g)),$
 $\dots\}$

بهبود تکرارشونده (Iterative Improvement)



Local Search for CSP

الگوریتم‌های تکرارشونده برای CSPها

- روش‌های جستجوی محلی معمولاً با حالات «کامل» کار می‌کنند،

یعنی به تمام متغیرها مقداری اختصاص داده شده باشد

- برای اعمال به CSPها:

- یک انتساب با محدودیت‌های ارضا نشده را انتخاب کنید

- عملگرها مقادیر متغیر را دوباره تخصیص می‌دهند

- بدون لبه (Fringe)!

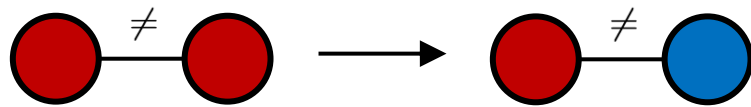
- الگوریتم: تا زمانی که حل نشده است،

- انتخاب متغیر: به طور تصادفی هر متغیری که محدودیتی را نقض کرده است را انتخاب کنید

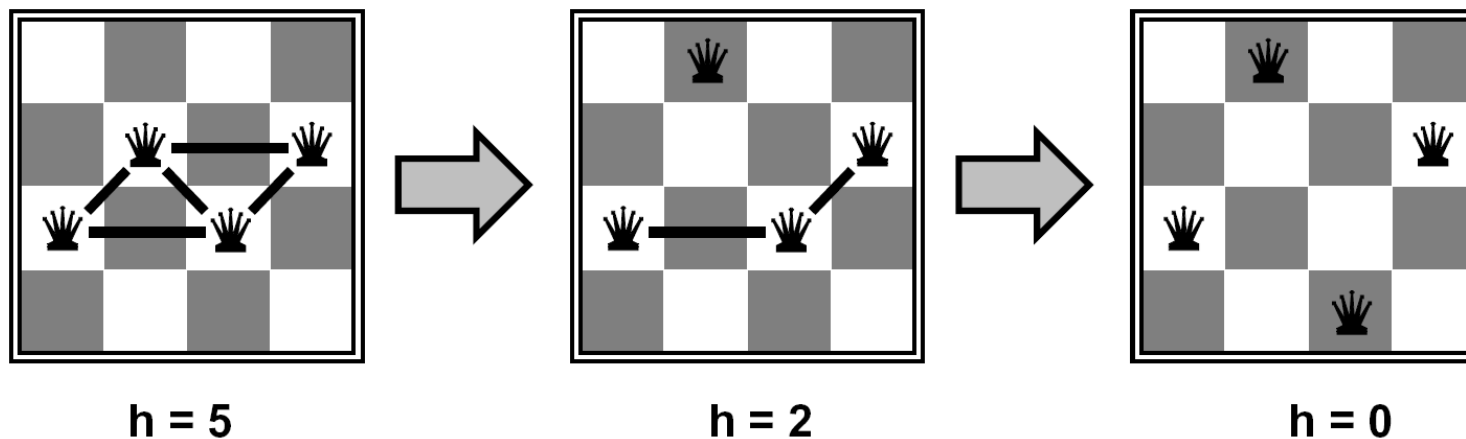
- انتخاب مقدار: هیوریستیک حداقل مغایرت‌ها (min-conflicts):

- مقداری را انتخاب کنید که کمترین محدودیت‌ها را نقض کند

- یعنی، الگوریتم تپه نوردی (hill climb) با $h(n)$ = تعداد کل محدودیت‌های نقض شده



مثال: 4- وزیر



- حالات: 4 وزیر در 4 ستون ($4^4 = 256$ حالت)
- عملگرها: حرکت ملکه در ستون
- آزمون هدف: وزیرها همدیگر را تهدید نکنند
- ارزیابی: $c(n)$ = تعداد تهدیدها

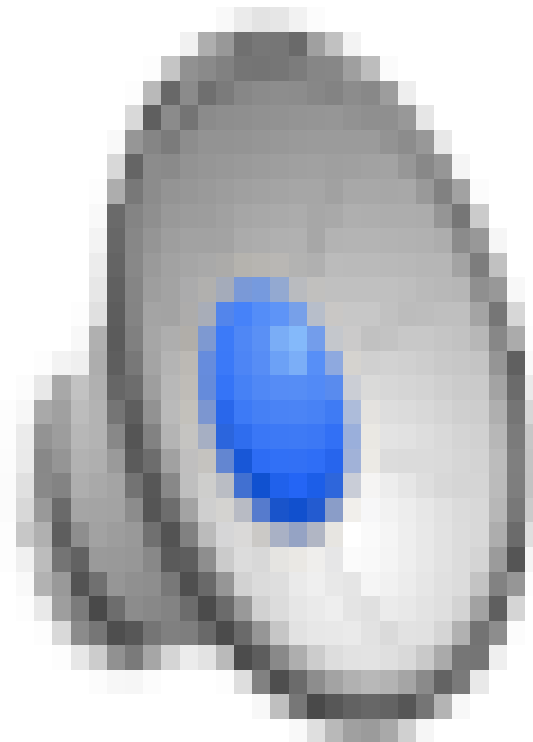
شبه کد جستجوی محلی برای حل مسائل ارضای محدودیت

```
function MIN-CONFLICTS(csp, max_steps) returns a solution or failure
  inputs: csp, a constraint satisfaction problem
           max_steps, the number of steps allowed before giving up

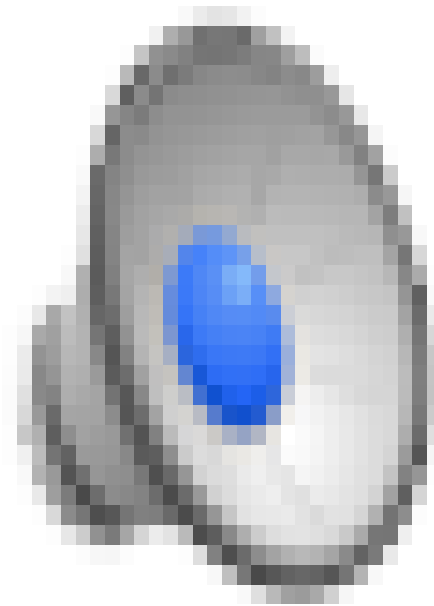
  current  $\leftarrow$  an initial complete assignment for csp
  for i = 1 to max_steps do
    if current is a solution for csp then return current
    var  $\leftarrow$  a randomly chosen conflicted variable from csp.VARIABLES
    value  $\leftarrow$  the value v for var that minimizes CONFLICTS(csp, var, v, current)
    set var = value in current
  return failure
```

Figure 5.9 The MIN-CONFLICTS local search algorithm for CSPs. The initial state may be chosen randomly or by a greedy assignment process that chooses a minimal-conflict value for each variable in turn. The CONFLICTS function counts the number of constraints violated by a particular value, given the rest of the current assignment.

ویدیوی دموی بهبود تکرارشونده - n-وزیر



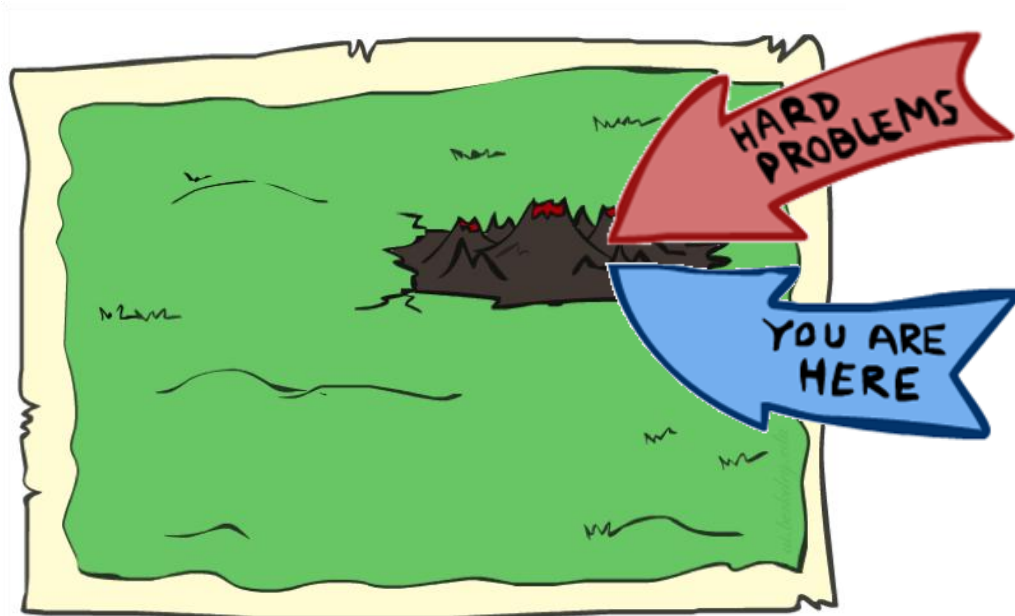
ویدیوی دموی بهبود تکرارشونده - رنگ آمیزی



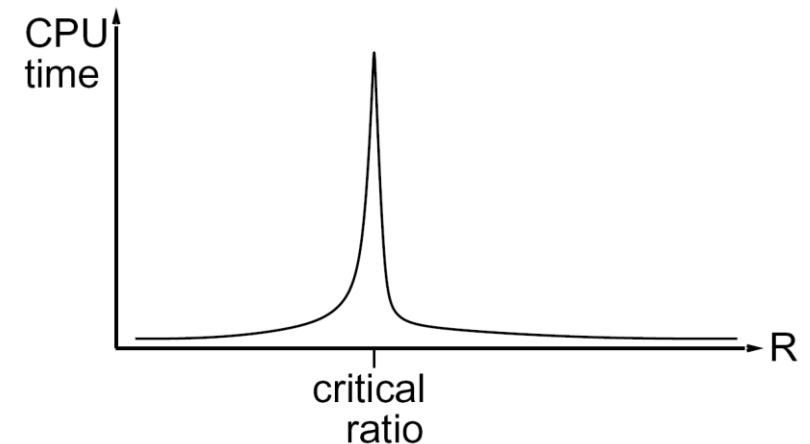
عملکرد Min-Conflict

- با توجه به حالت اولیه تصادفی، می‌توان n وزیر را در زمان تقریباً ثابت برای n دلخواه با احتمال بالا حل کرد (به عنوان مثال، $n = 10,000,000$!)

- به نظر می‌رسد که همین امر برای هر CSP که به طور تصادفی تولید می‌شود صادق باشد، به جز در محدوده باریکی از نسبت زیر:

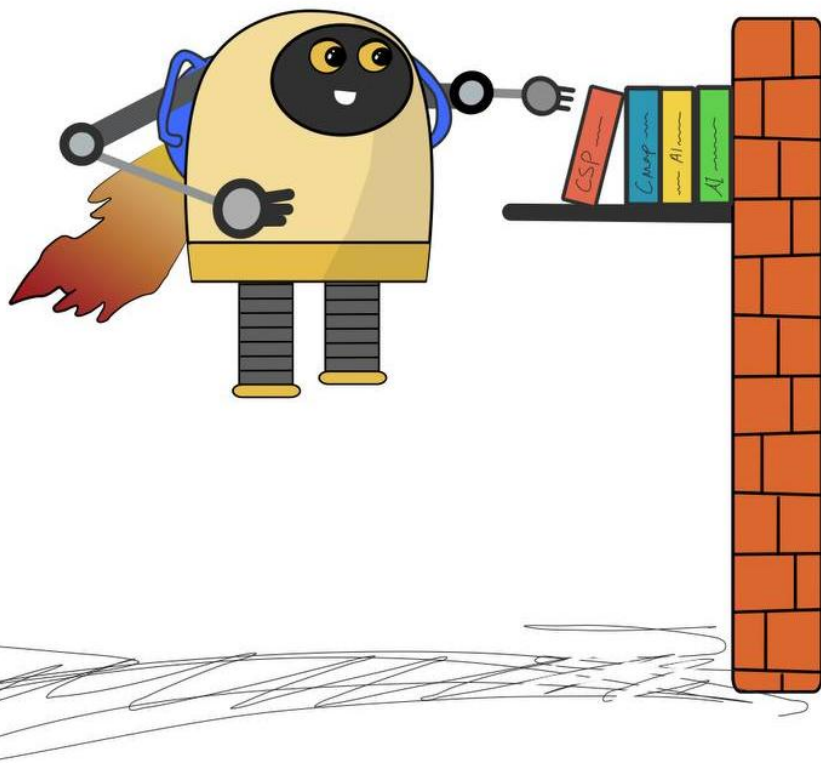


$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



خلاصه: CSPها

- CSPها نوع خاصی از مسائل جستجو هستند:
 - حالات (states)، انتسابات جزئی هستند.
 - آزمون هدف با محدودیت‌ها تعریف می‌شود.
- راه‌حل اساسی: جستجوی عقبگرد (Backtracking)
 - افزایش سرعت جستجو:
 - مرتب‌سازی (Ordering)
 - فیلتر کردن (Filtering)
 - ساختار (Structure)
- min-conflict‌های تکرارشونده اغلب در عمل موثر هستند.

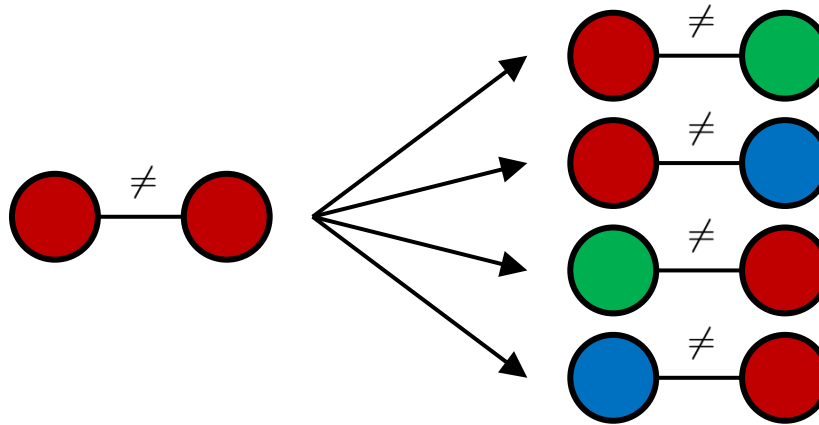


جستجوی محلی



جستجوی محلی

- جستجوی درختی مقادیر جایگزین بررسی نشده (unexplored) را در لیست لبه (fringe) نگه می‌دارد (تضمین کامل بودن)
- جستجوی محلی: یک گزینه را تا زمانی که نتوانید آن را بهتر کنید بهبود دهید (بدون لبه!)
- تابع پسین جدید: تغییرات محلی



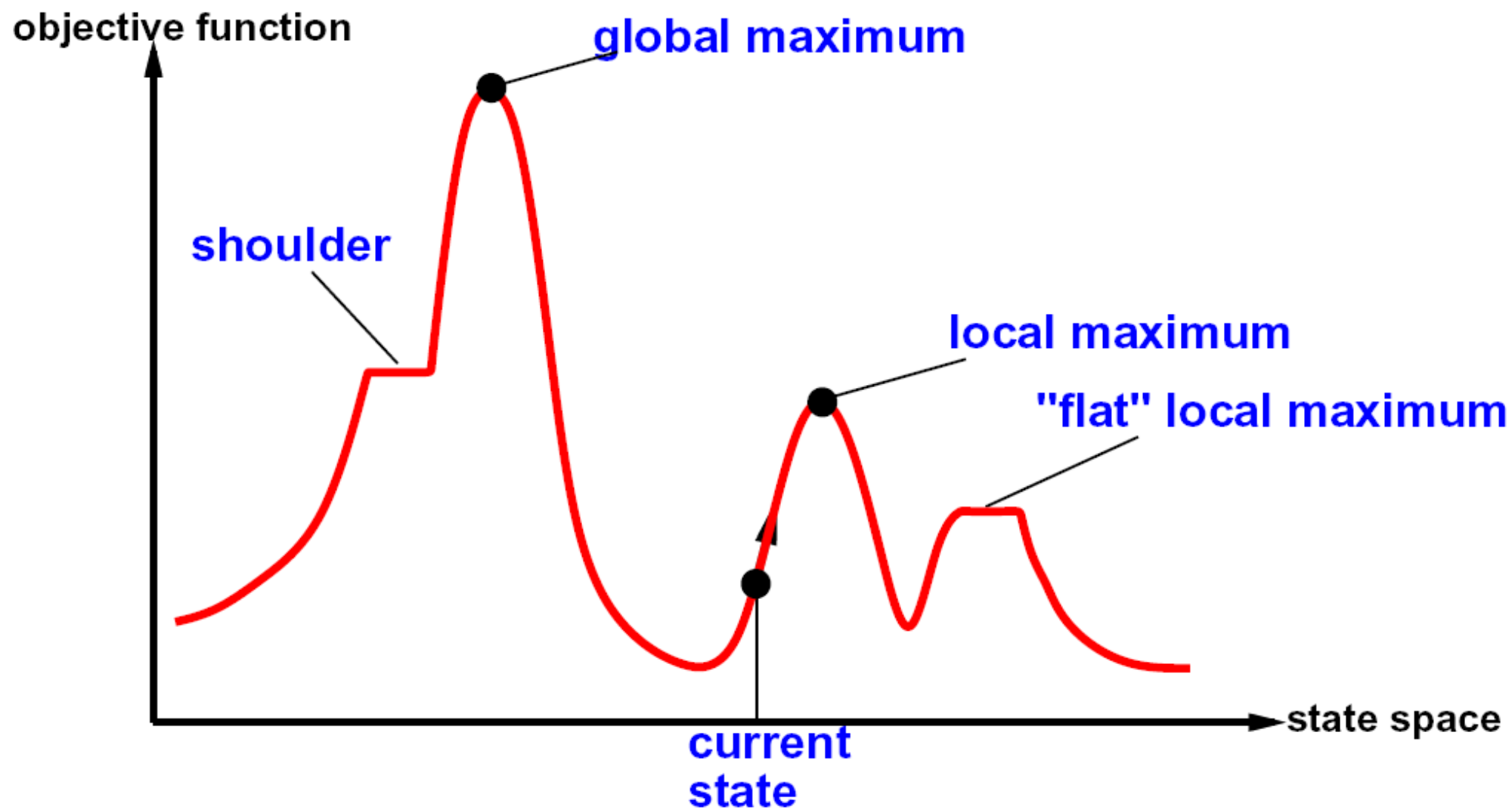
- به طور کلی بسیار سریع‌تر و از نظر حافظه‌ای کم‌هزینه‌تر (اما ناکامل و غیربهبوده)

تپه نوردی (Hill climbing)

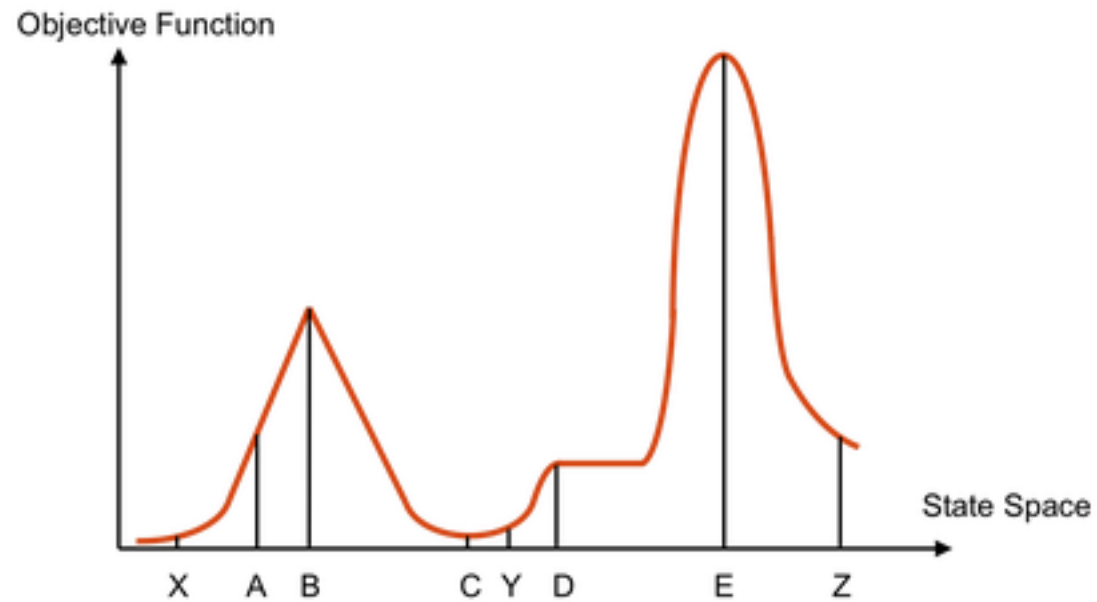


- ایده ساده و کلی:
 - از هر کجا شروع کنید
 - تکرار کنید: به بهترین حالت (state) همسایه بروید
 - اگر همسایه‌ای بهتر از حالت فعلی نیست، دست از کار بکشید
- معایب این رویکرد چیست؟
 - کامل؟
 - بهینه؟
- مزایای آن چیست؟

نمودار تپه نوردی



آزمونک تپه نوردی

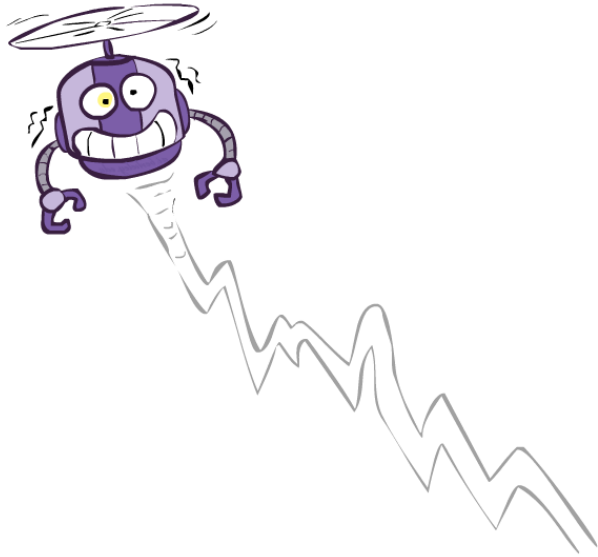


از X شروع کنید، به کجا می‌رسید؟

از Y شروع کنید، به کجا می‌رسید؟

از Z شروع کنید، به کجا می‌رسید؟

تبرید شبیه‌سازی‌شده (Simulated Annealing)



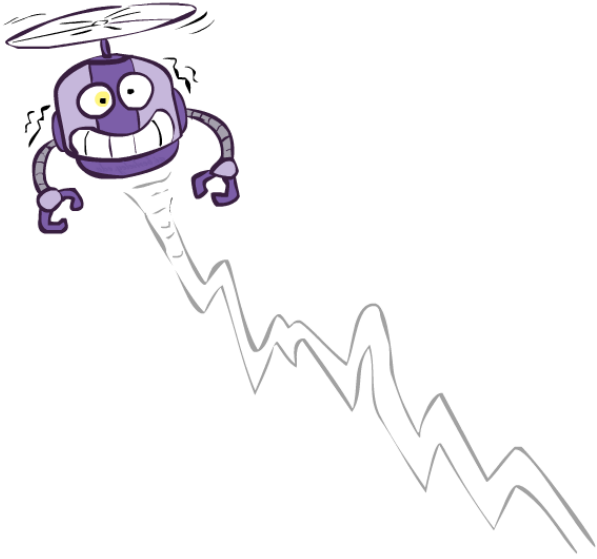
• ایده: با اجازه دادن به حرکت در سراشیبی، از بیشینه‌های محلی فرار کنید

• اما با گذشت زمان آن‌ها را نادرتر کنید

```
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

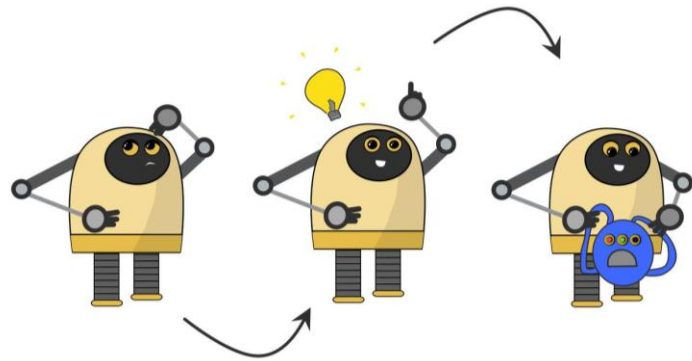
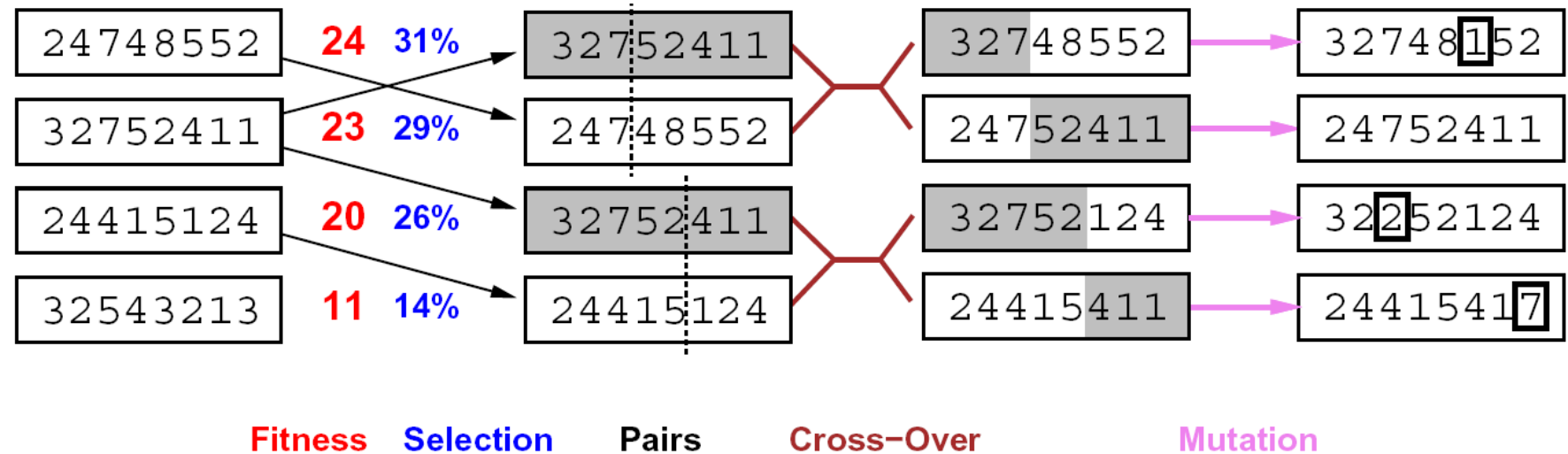
current ← MAKE-NODE(INITIAL-STATE[problem])
for t ← 1 to ∞ do
    T ← schedule[t]
    if T = 0 then return current
    next ← a randomly selected successor of current
     $\Delta E \leftarrow \text{VALUE}[\textit{next}] - \text{VALUE}[\textit{current}]$ 
    if  $\Delta E > 0$  then current ← next
    else current ← next only with probability  $e^{\Delta E/T}$ 
```

تبرید شبیه‌سازی شده



- تضمین تئوری: $p(x) \propto e^{-\frac{E(x)}{kT}}$
- توزیع ثابت:
- اگر T به اندازه کافی به آرامی کاهش یابد، به حالت بهینه همگرا می‌شود!
- آیا این تضمین جالبی است؟
- جادو به نظر می‌رسد، اما واقعیت واقعیت است:
- هرچه برای فرار از بهینه محلی به گام‌های سراشیبی بیشتری نیاز داشته باشید، کمتر احتمال دارد که همه آنها را پشت سر هم انجام دهید.
- مردم در مورد عملگرهای مرزی (*Ridge Operators*) بسیار می‌اندیشند که به شما امکان پرش در فضا به روش‌های بهتری را بدهند.

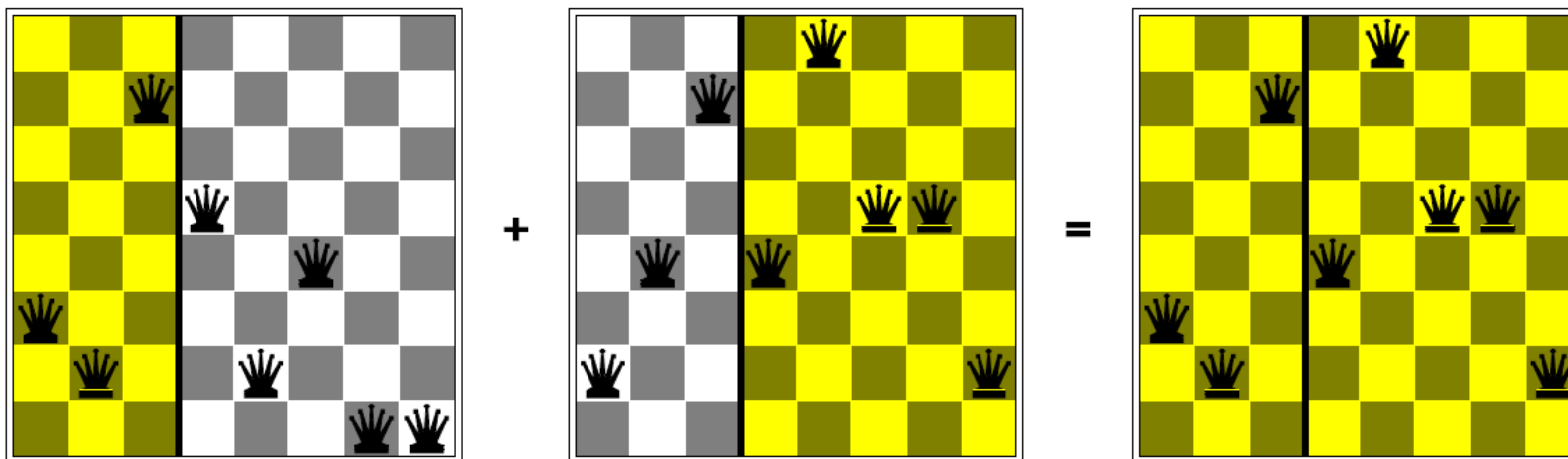
الگوریتم ژنتیک



- الگوریتم‌های ژنتیک از تشبیه انتخاب طبیعی (Natural Selection) استفاده می‌کنند
- بهترین N فرضیه را در هر مرحله (انتخاب) بر اساس تابع شایستگی (Fitness Function) نگه دارید
- همچنین دارای عملگرهای متقاطع جفتی (Cross-Over)، با جهش (Mutation) اختیاری برای ایجاد تنوع

- احتمالاً به اشتباه تعبیر شده‌ترین، به اشتباه به کار گرفته شده‌ترین (و حتی بدنام شده‌ترین) تکنیک موجود

مثال: 4-وزیر



- چرا crossover اینجا معقول است؟
- چه زمانی منطقی نخواهد بود؟
- جهش چه خواهد بود؟
- تابع شایستگی مناسب چه خواهد بود؟

جلسات بعدی: جستجوی خصمانه! (Adversarial Search)
