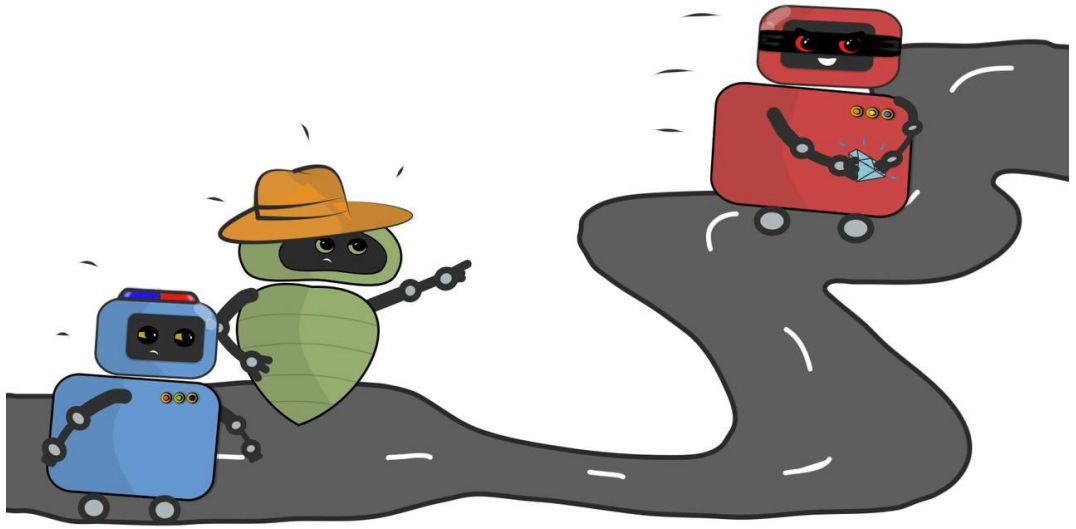
مبانی و کاربردهای هوش مصنوعی

مسائل ارضای محدودیت (فصل 6.1 الی 6.5)



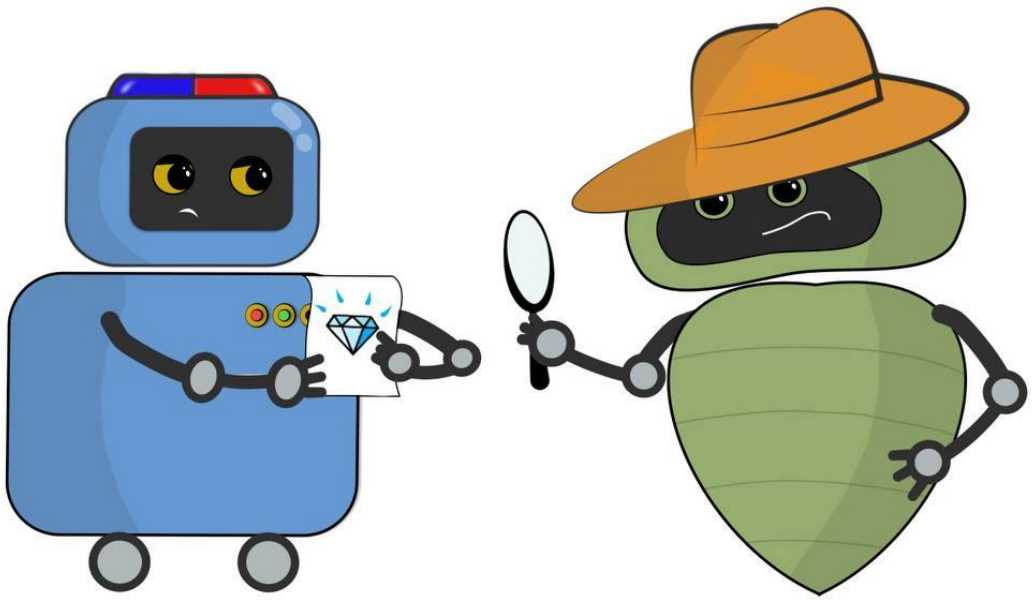
مدرس: مهدی جوانمردی

دانشکده مهندسی کامپیوتر دانشگاه صنعتی امیرکبیر

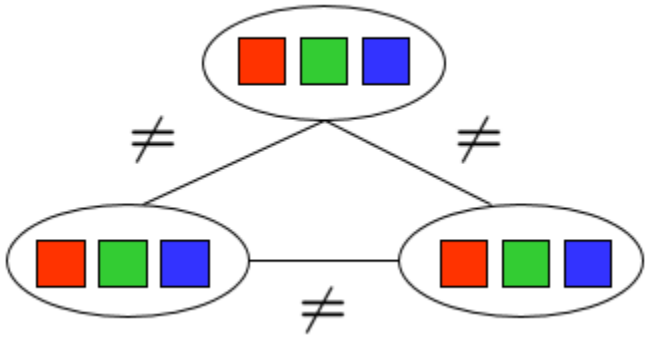
(الهام گرفته از محتوای درس هوش مصنوعی دانشگاه برکلی)

امروز

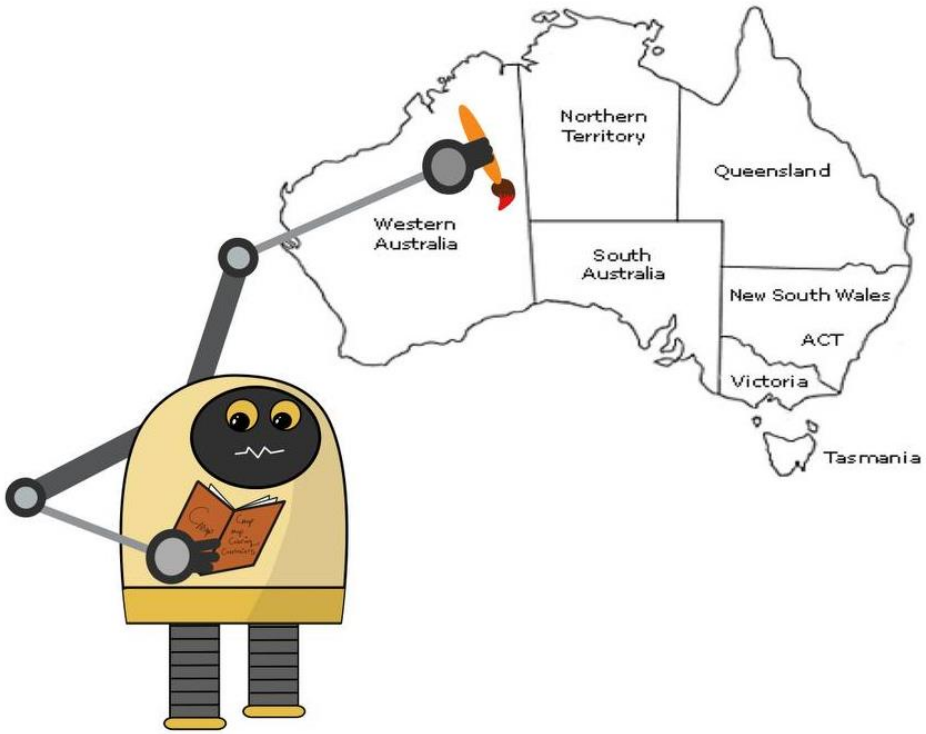
- راه‌حل کارآمد برای CSPها
- جستجوی محلی



یادآوری: CSPها



- CSPها:
  - 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.

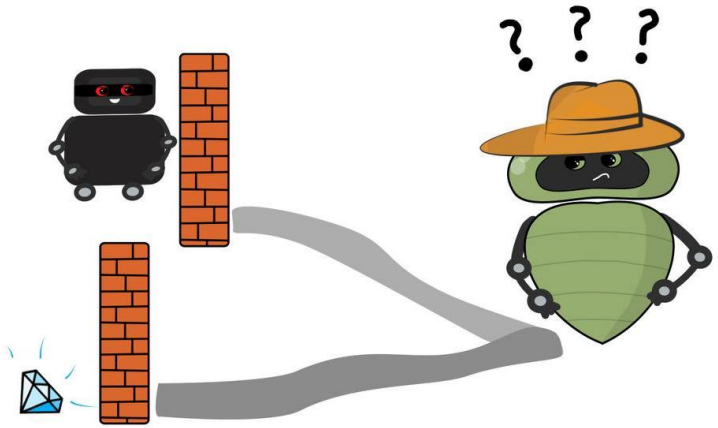


جستجوی عقبگرد

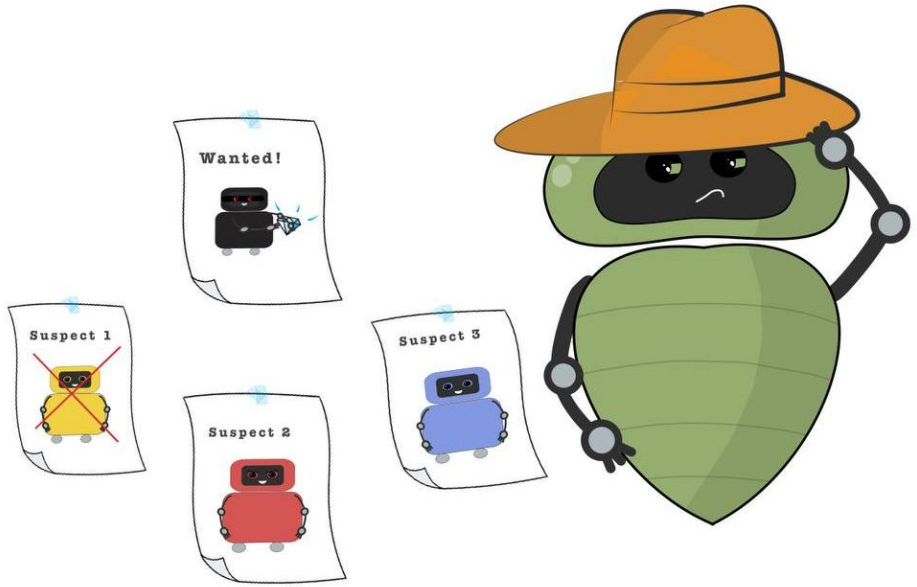
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

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
```

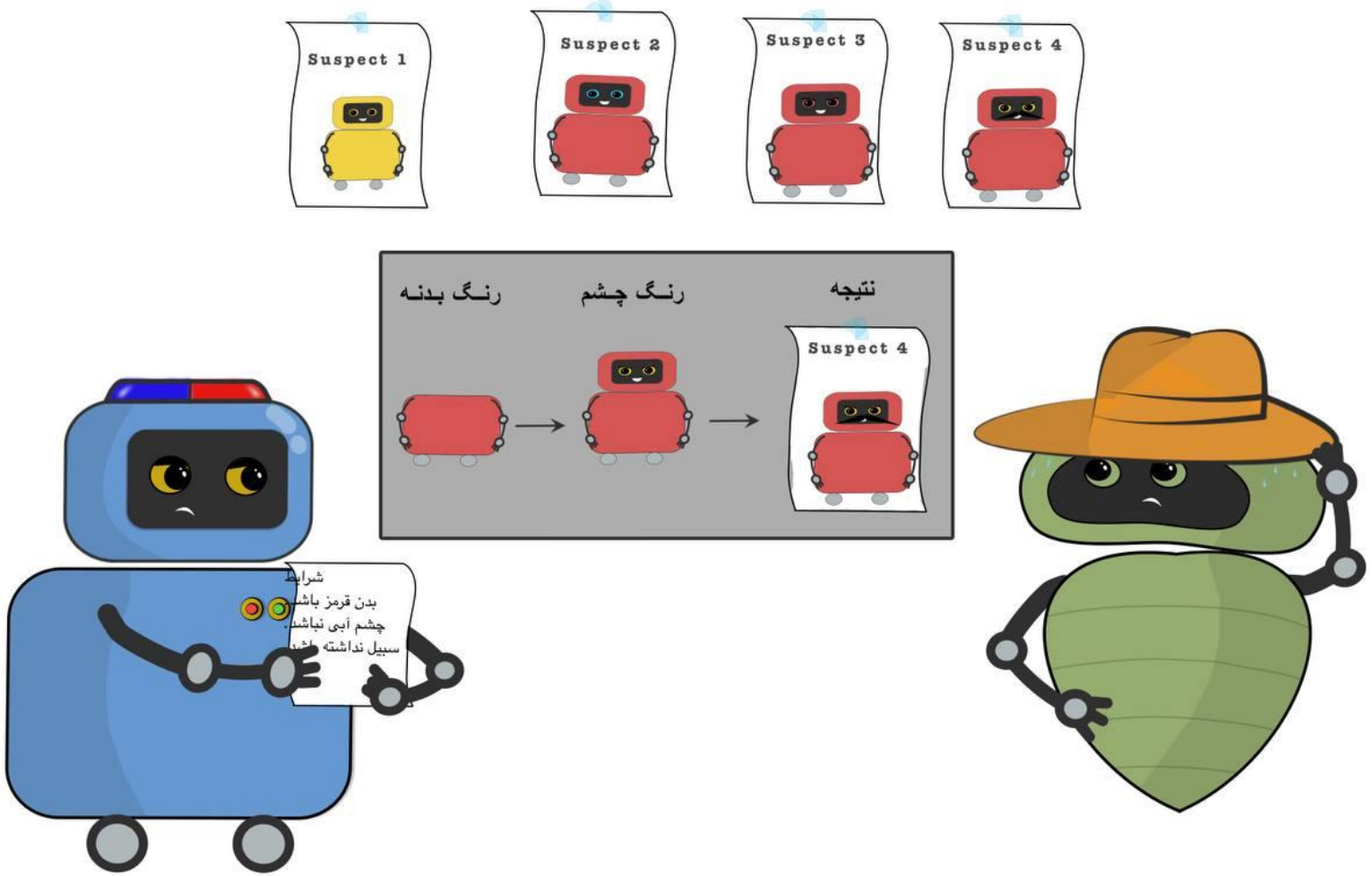
بهبود عقبگرد



- 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؟
- مرتب سازی:
  - کدام متغیر باید در مرحله بعد مقدار دهی شود؟ (MRV)
  - مقادیر ممکن آن را به چه ترتیبی باید امتحان کرد؟ (LCV)
- ساختار: آیا می‌توانیم از ساختار مسئله بهره برداری کنیم؟

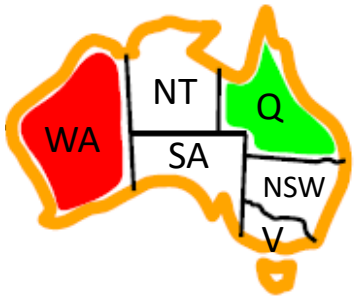


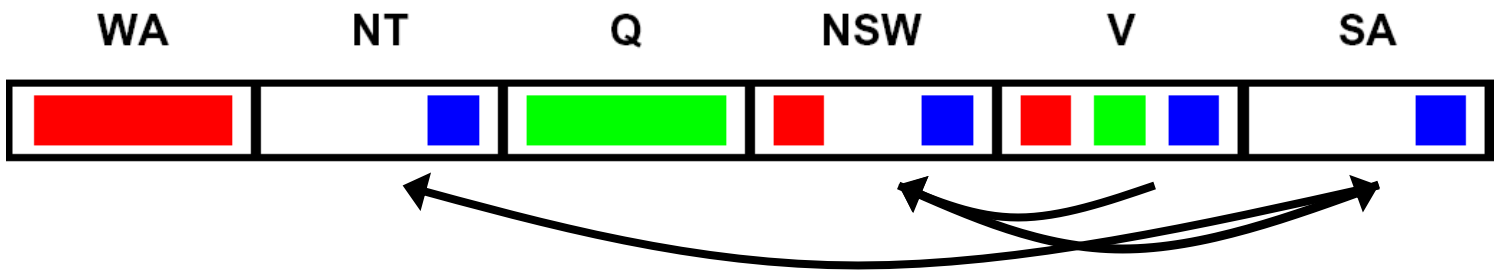
سازگاری یال و فراتر از آن



سازگاری یال یک CSP کامل

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

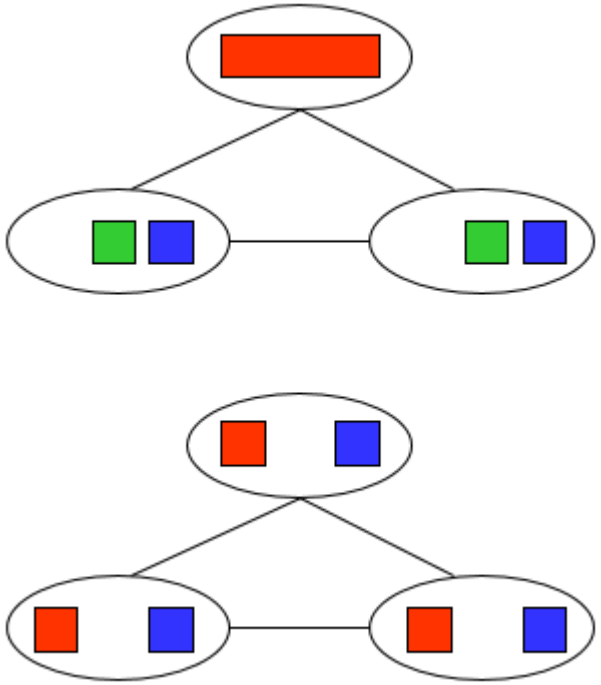




- سازگاری یال، شکست را زودتر از بررسی رو به جلو تشخیص می‌دهد.
- مهم: اگر X مقداری را از دست بدهد، همسایگان X باید دوباره بررسی شوند!
- باید بعد از هر انتساب دوباره اجرا شود!

به یاد داشته باشید: حذف از دم!

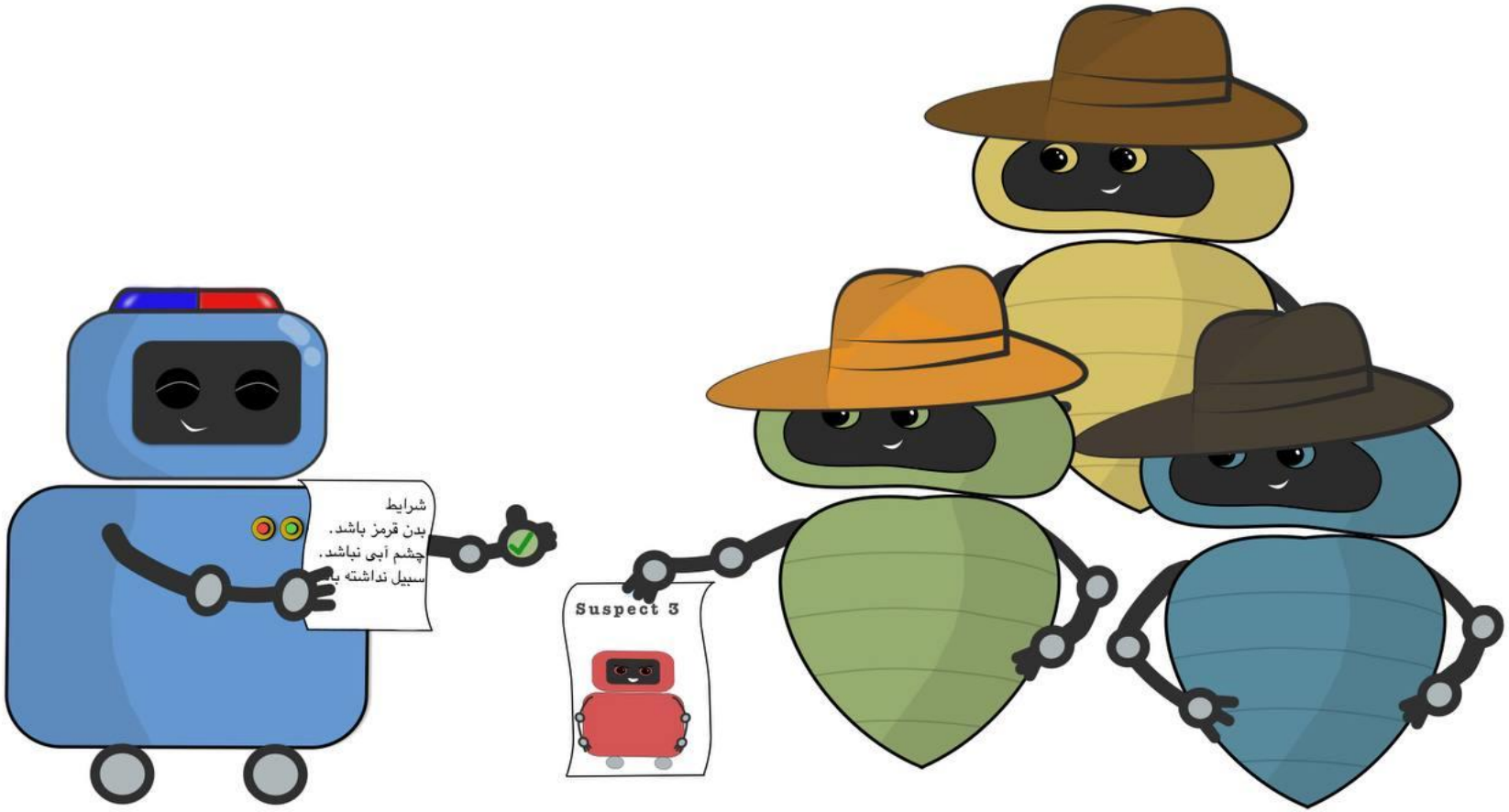
محدودیت‌های سازگاری یال



اینجا چه اشتباهی
رخ داده است؟

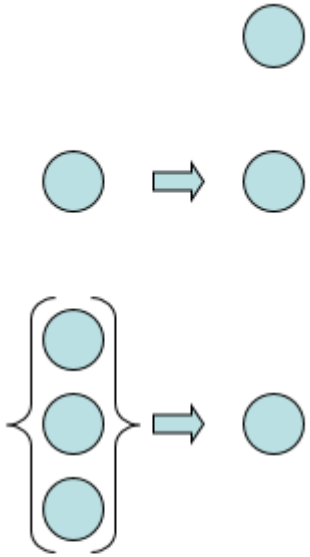
- پس از اعمال سازگاری یال:
  - می‌تواند یک راه حل باقی بماند.
  - می‌تواند چندین راه حل باقی مانده باشد.
  - می‌تواند هیچ راه حلی باقی نمانده باشد (و الگوریتم جستجو نداند)
- سازگاری یال همچنان در داخل جستجوی عقبگرد اجرا می‌شود!

سازگاری Kتایی (K-Consistency)

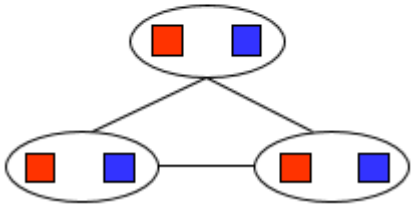


سازگاری Kتایی

- افزایش درجات سازگاری:
  - سازگاری 1تایی (سازگاری گره): دامنه هر گره منفرد دارای مقداری است که با محدودیت‌های یگانی (Unary Constraints) آن گره مطابقت دارد
  - سازگاری 2تایی (سازگاری یال): برای هر جفت گره، هر تخصیص سازگار به یکی از گره‌ها را می توان به دیگری تعمیم داد
  - سازگاری Kتایی: برای هر k گره، هر تخصیص سازگار به k-1 گره را می توان به گره k ام تعمیم داد



- هر چقدر k بیشتر باشد، هزینه محاسبات بیشتر می‌شود
- (شما باید حالت k=2 را بدانید: سازگاری یال)



سازگاری Kتایی قوی
(Strong K-Consistency)

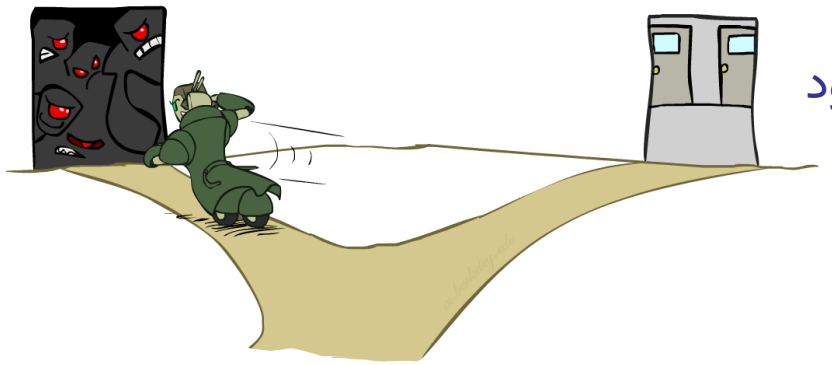
- سازگاری kتایی قوی: علاوه بر k، برای 1 ,..., k-2, k-1 سازگاری وجود داشته باشد
- ادعا: سازگاری nتایی قوی به این معنی است که ما می‌توانیم بدون عقبگرد (Backtracking) مسئله را حل کنیم!
- چرا؟
  - هر انتساب به هر متغیری را انتخاب کنید
  - یک متغیر جدید انتخاب کنید
  - با سازگاری 2تایی، حداقل یک انتخاب سازگار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

مرتب‌سازی (Ordering)

مرتب‌سازی: حداقل مقادیر باقیمانده (Minimum Remaining Values)

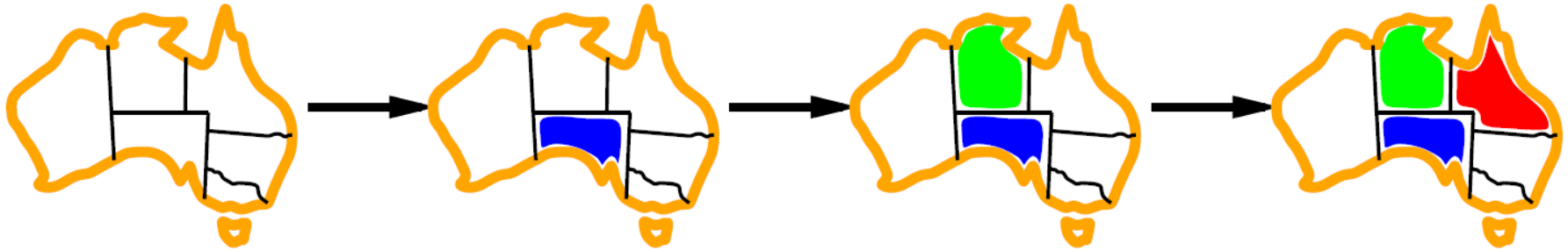
- ترتیب متغیر: هیوریستیک حداقل مقادیر باقیمانده (MRV)
  - متغیری را با کمترین مقادیر ممکن باقی‌مانده در دامنه خود انتخاب کنید

- 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
- مرتب سازی "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
- هیوریتستیک درجه‌ای:
  - متغیری را انتخاب کنید که در بیشترین محدودیت‌ها بر متغیرهای باقیمانده شرکت دارد

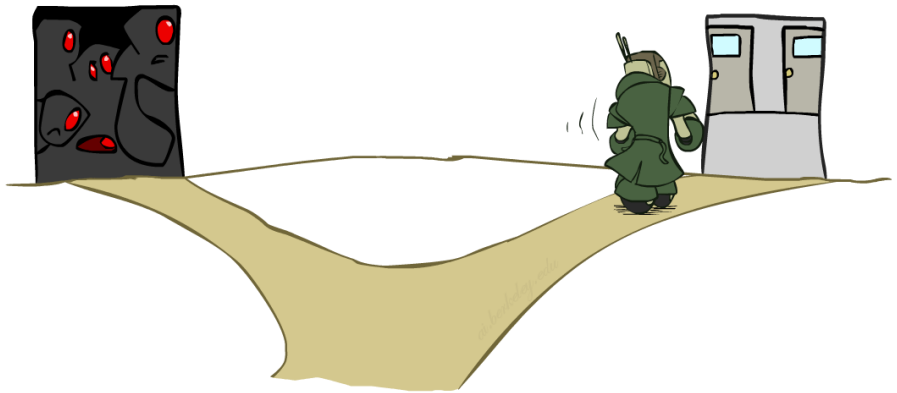


- چرا بیشتر به جای کمترین محدودیت‌ها؟

مقدار اعمال‌کننده کمترین محدودیت
(Least Constraining Value)

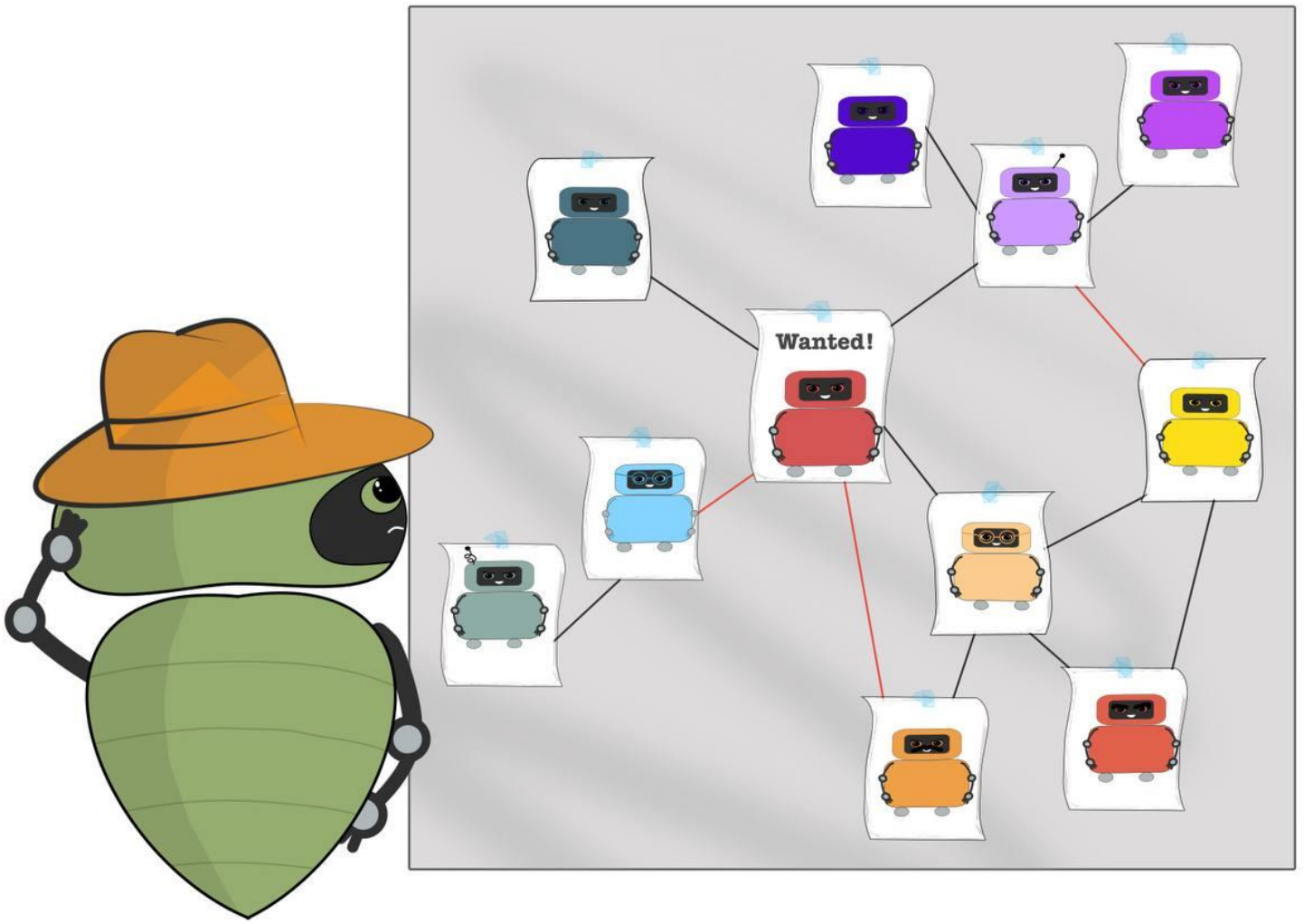
- ترتیب مقدار: مقدار اعمال‌کننده کمترین محدودیت
  - با توجه به انتخاب متغیر، مقداری را انتخاب کنید که حداقل محدودیت را به دیگر متغیرها اعمال کند
  - یعنی مقداری که با تخصیص آن، کمترین تعداد مقادیر از بین مقادیر ممکن برای متغیرهای باقی‌مانده خط می‌خورند
  - توجه داشته باشید که برای تعیین این مقدار ممکن 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 آسان‌تر شروع کن

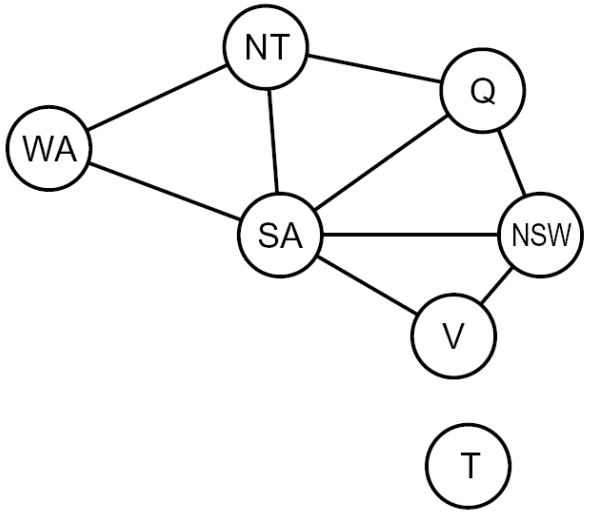


- ترکیب این ایده‌های مرتب‌سازی، حل 1000 وزیر را امکان پذیر می‌کند

ساختار (Structure)

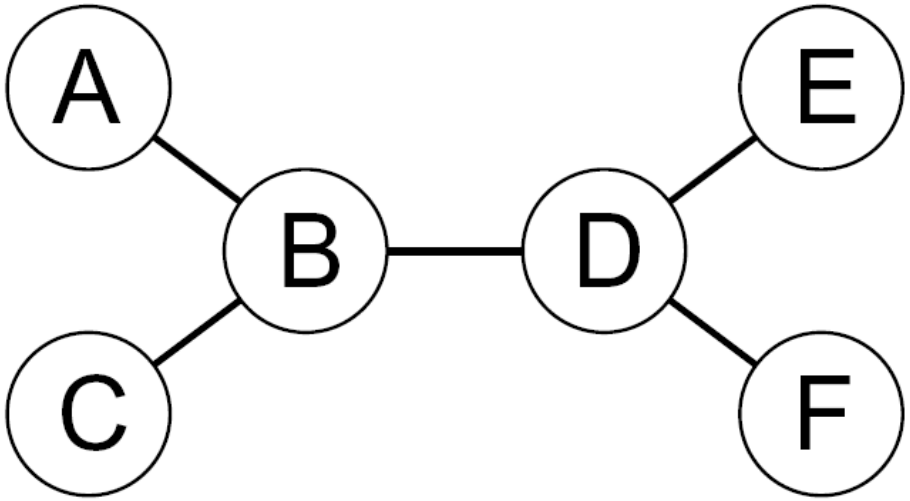


ساختار مسئله



- حالت شدید: 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
  - به عنوان مثال، n = 80، d = 2، c = 20
  - 2^{80}= 4 میلیارد سال با سرعت محاسبه 10 میلیون گره در ثانیه
  - 4×2^{20} = 0.4 ثانیه با سرعت محاسبه 10 میلیون گره در ثانیه

CSPهای با ساختار درختی
(Tree-Structured CSPs)

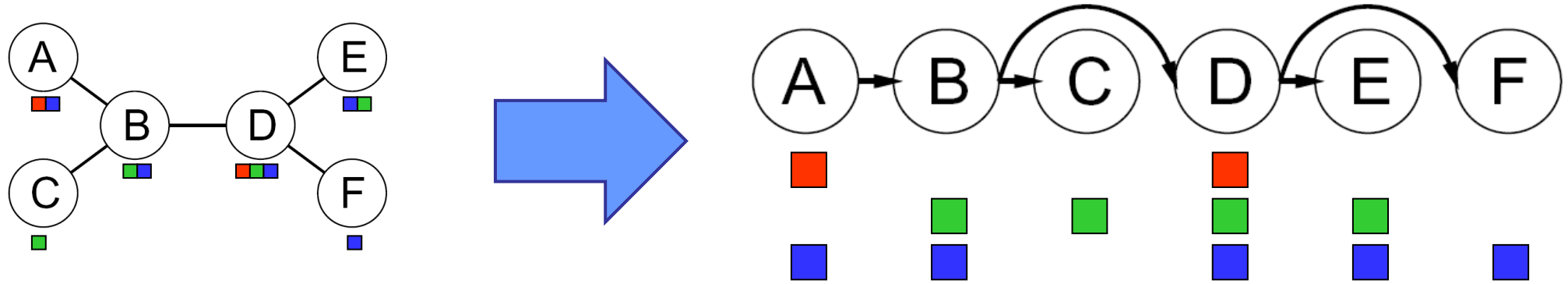


- قضیه: اگر گراف محدودیت فاقد حلقه باشد، CSP را می‌توان در زمان $O(n.d^2)$ حل کرد
  - مقایسه با هزینه CSPهای عمومی، که در آن بدترین زمان $O(d^n)$ است

- این ویژگی در مورد استنتاج احتمالی نیز صدق می‌کند (مباحث بعدی):
  نمونه‌ای از رابطه بین محدودیت‌های نحوی (syntactic restrictions) و پیچیدگی استدلال

CSPهای با ساختار درختی

- الگوریتم برای CSPهای 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 RemoveInconsistent(Parent(X_i),X_i)

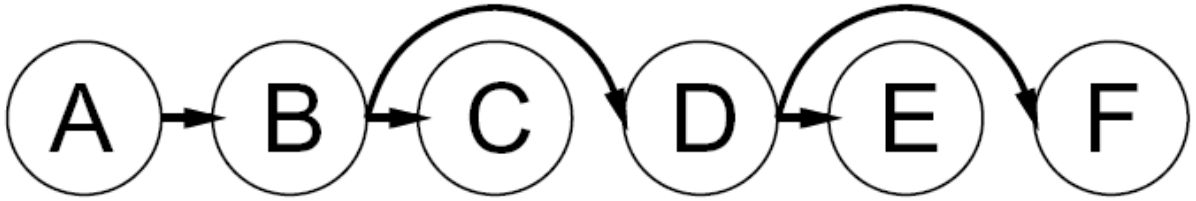
  - تخصیص رو به جلو:

For i = 1 : n, assign X_i consistently with Parent(X_i)

- زمان اجرا: $O(n.d^2)$ (چرا؟)

CSPهای با ساختار درختی

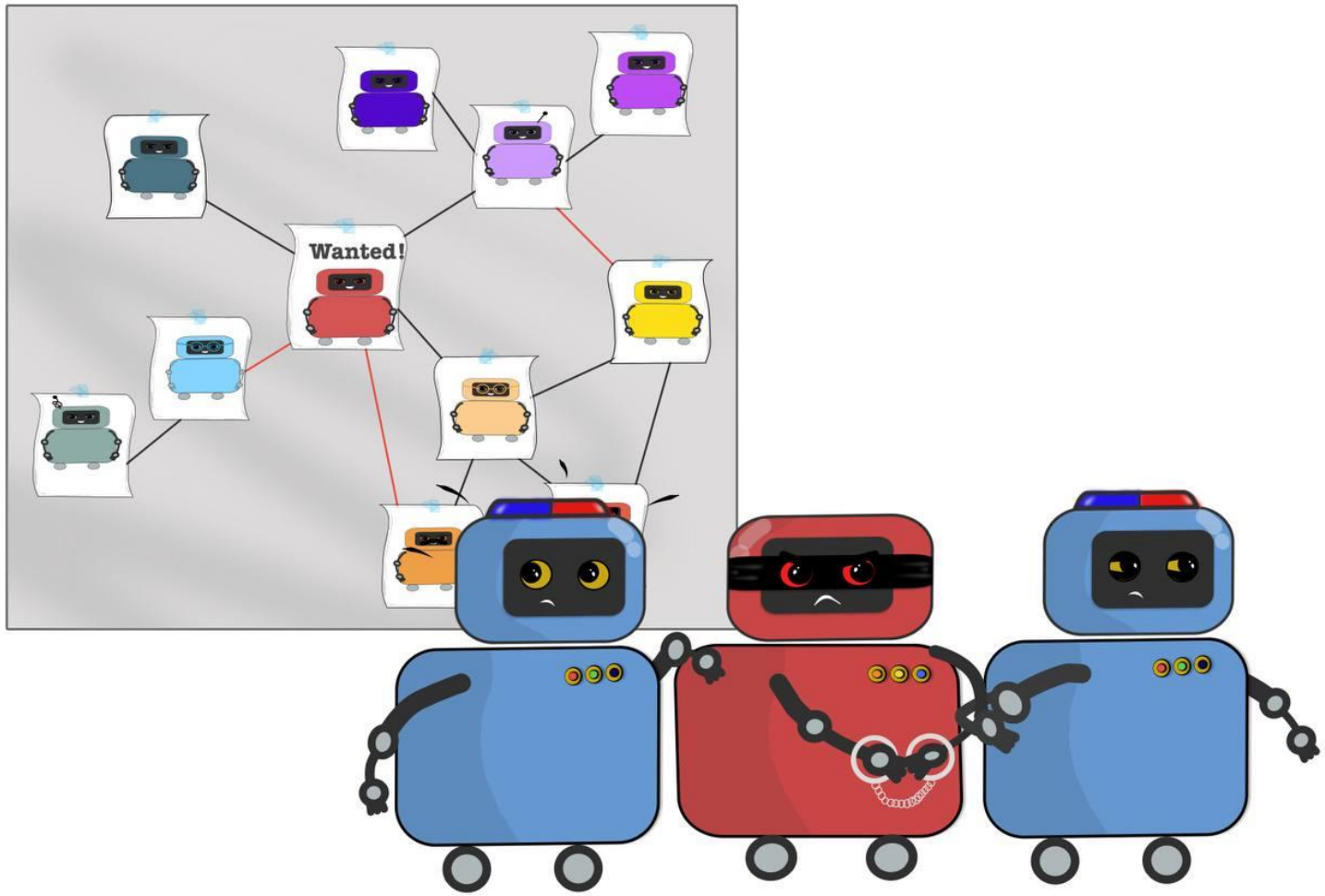
- ادعای 1: پس از پیمایش رو به عقب، تمام یال‌های ریشه به برگ سازگار هستند
- اثبات: هر $X \rightarrow Y$ در یک نقطه از پیمایش سازگار شده بود و دامنه Y’ نمی‌توانست پس از آن کاهش یابد (زیرا فرزندان Y’ قبل از Y پردازش شده بودند)



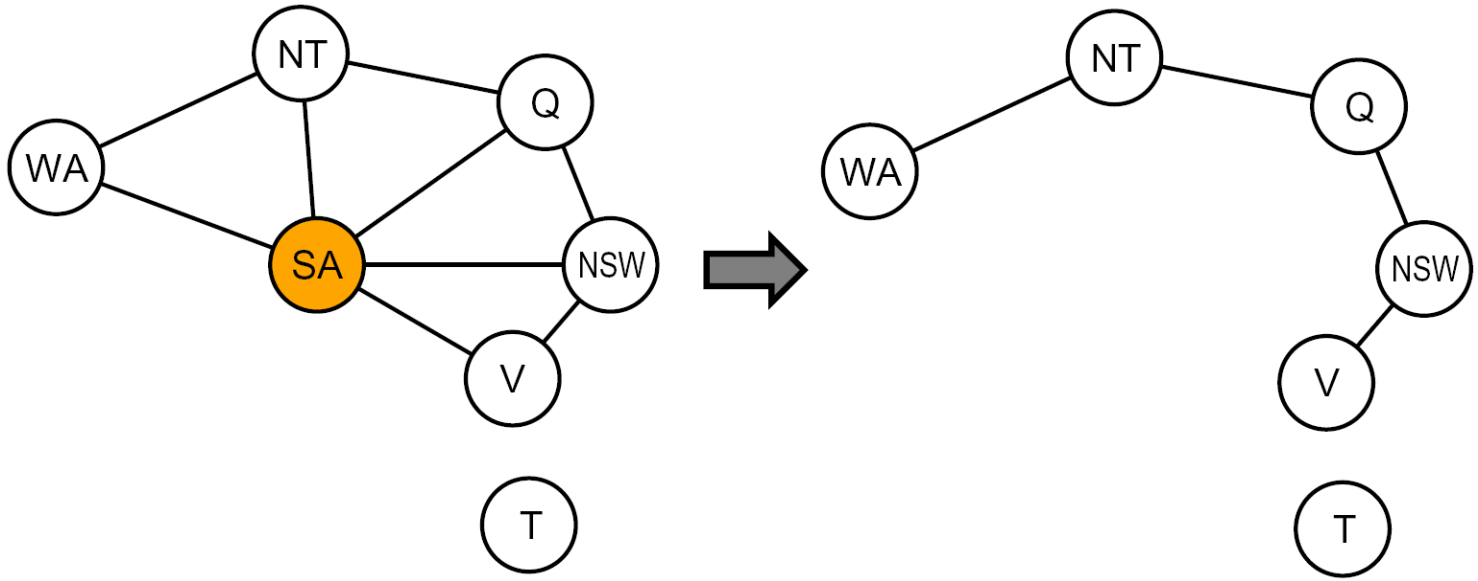
- ادعای 2: اگر یال‌های ریشه به برگ سازگار باشند، تخصیص رو به جلو عقبگرد (Backtrack) نخواهد داشت.

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

بهبود ساختار



CSPهای تقریباً با ساختار درختی



- شرطی‌سازی (Conditioning): یک متغیر را مقداردهی کنید، دامنه‌ی همسایه‌های آن را هرس کنید
- شرطی‌سازی 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 cهای کوچک بسیار سریع است

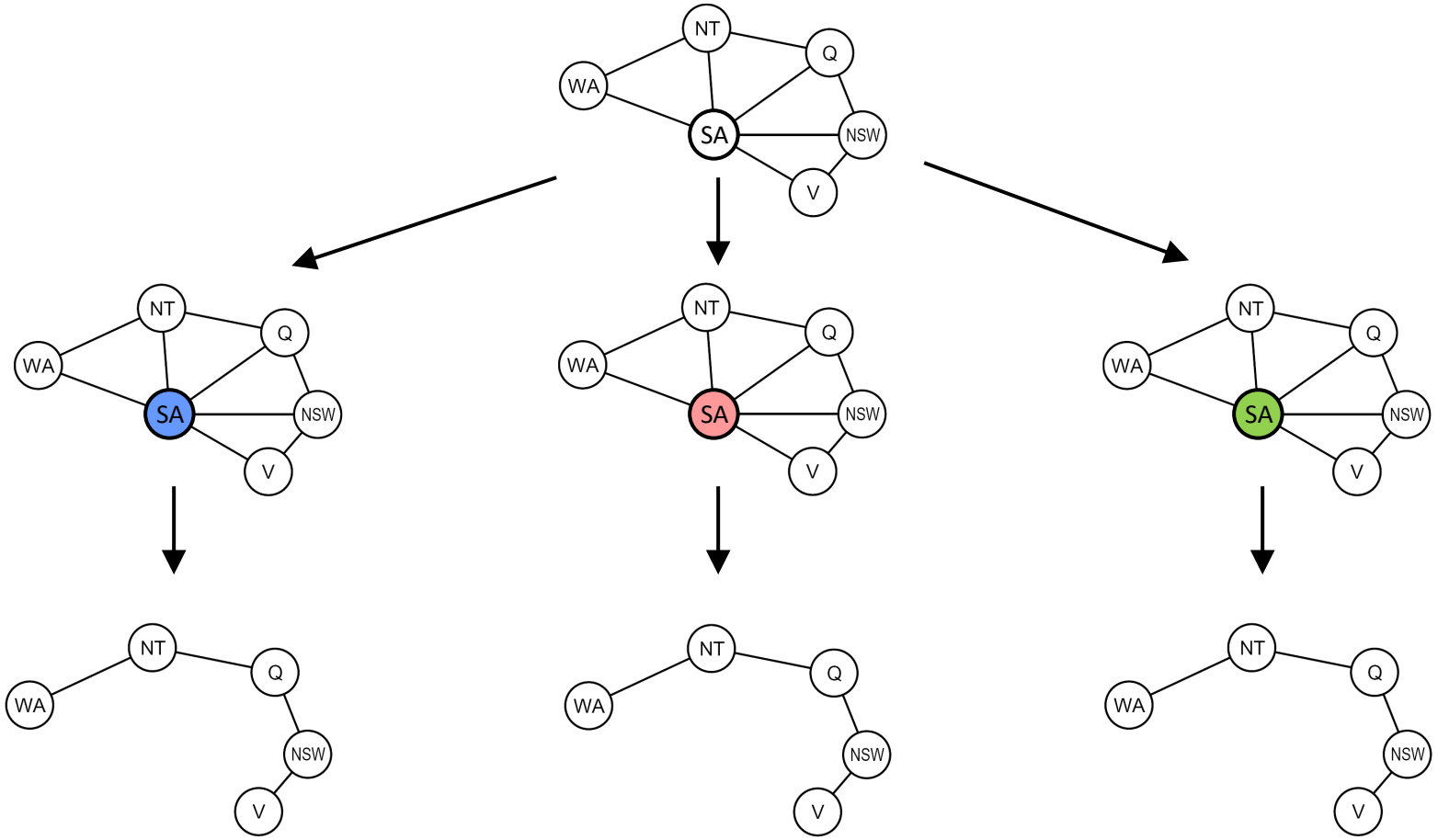
شرطی‌سازی Cutset
(Cutset Conditioning)

یک cutset انتخاب کنید

مقداردهی cutset
(امتحان کردن تمام حالات)

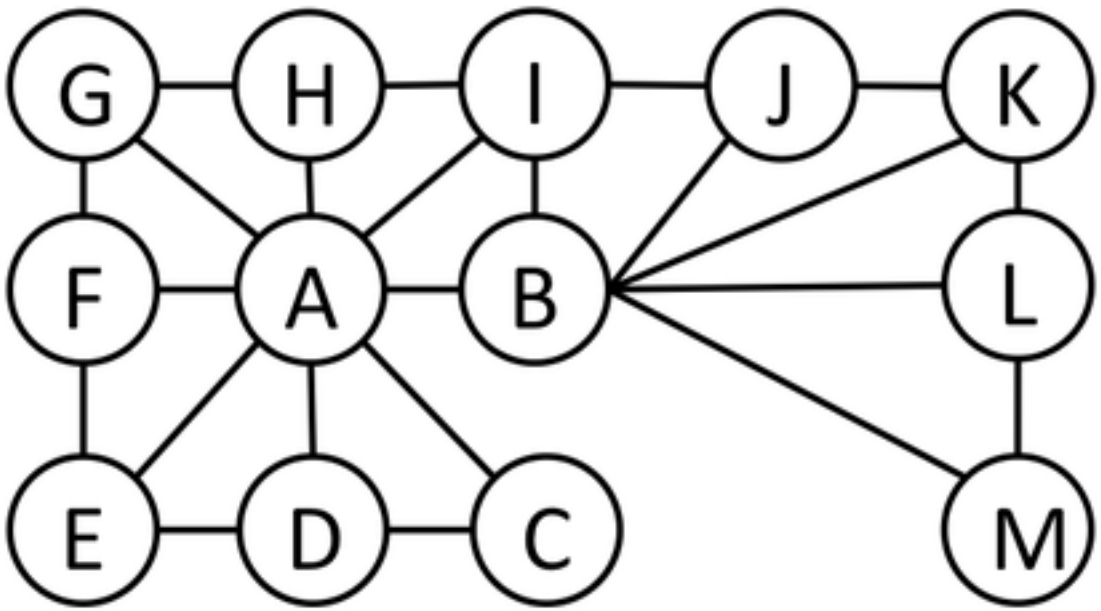
CSP باقی‌مانده را برای هر انتساب محسابه کنید

CSPهای باقی‌مانده را حل کنید (ساختار درختی)

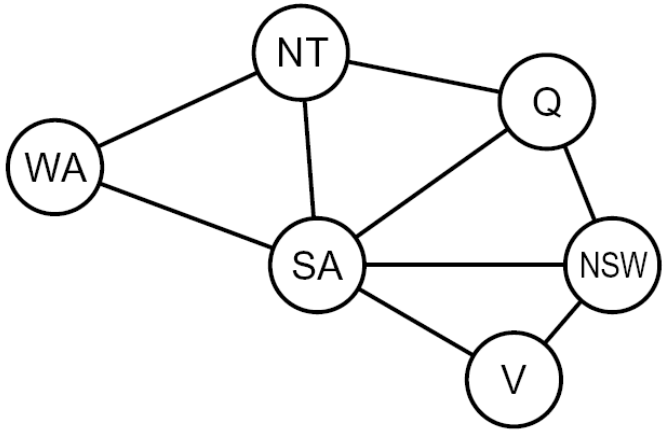


آزمونک Cutset

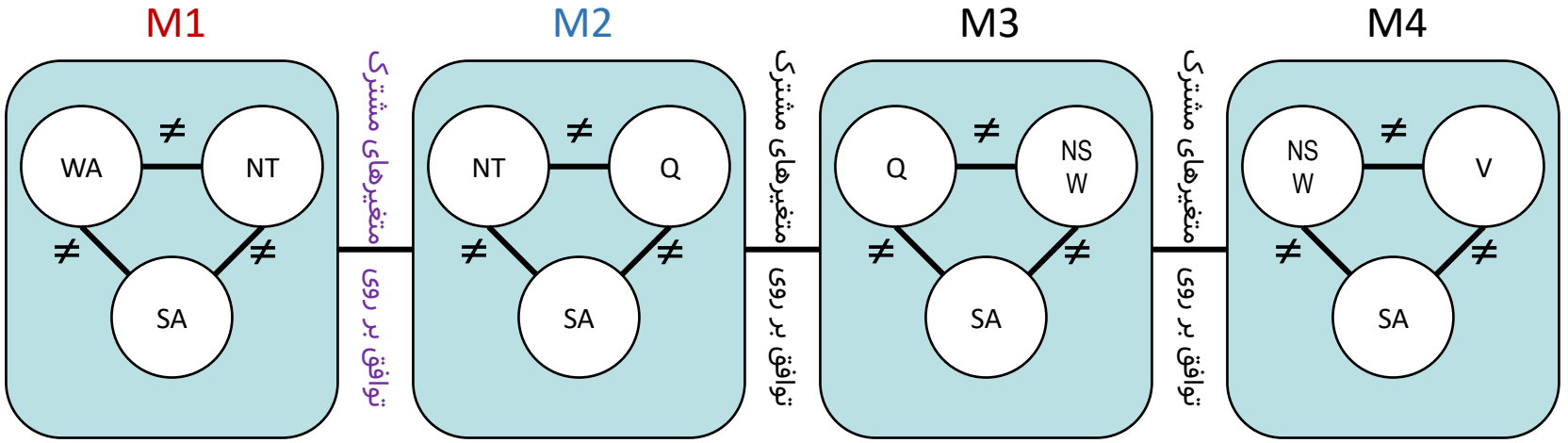
- کوچک‌ترین cutset را برای گراف زیر پیدا کنید



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

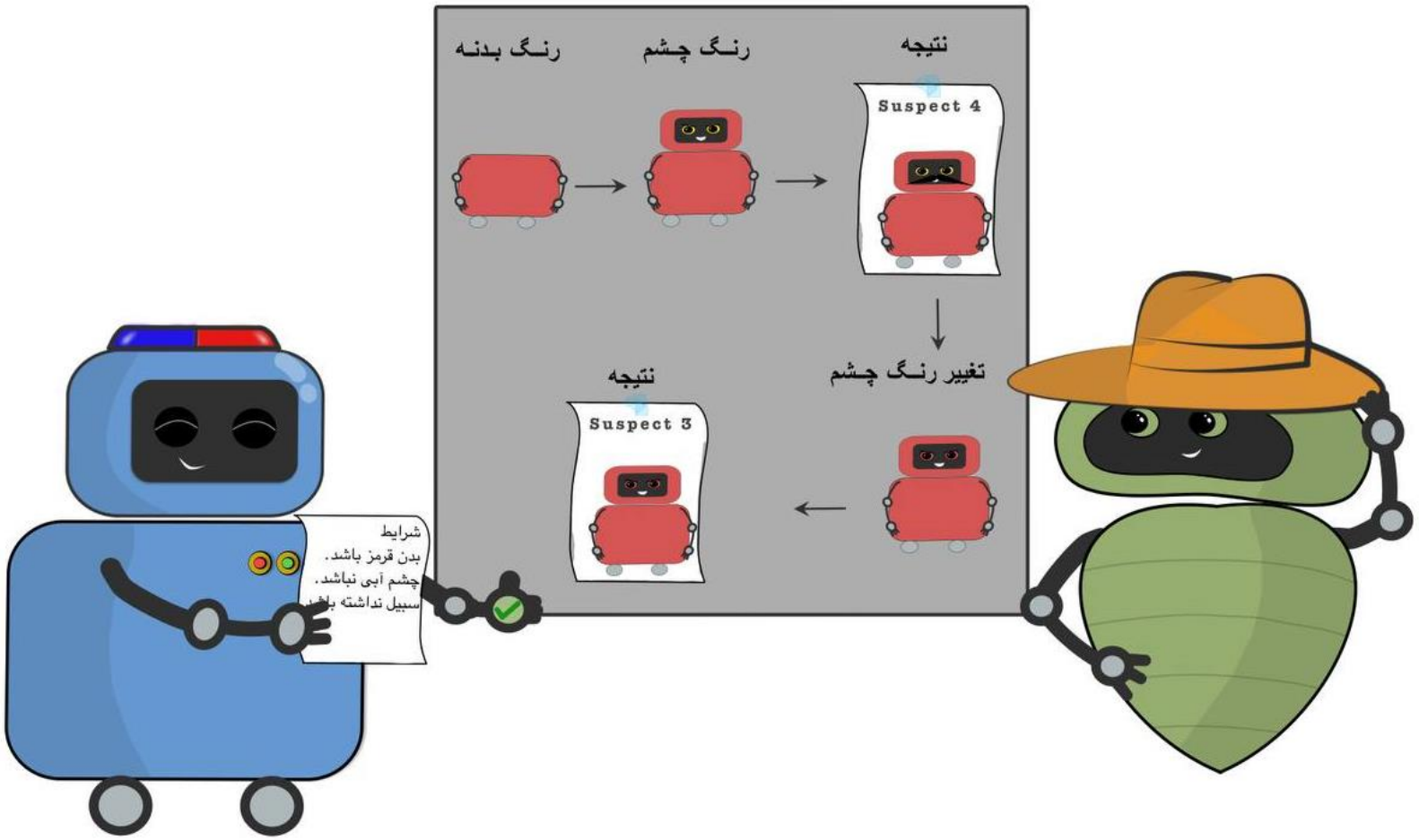


$\{(WA = r, SA = g, NT = b), (WA = b, SA = r, NT = g), \dots\}$

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

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

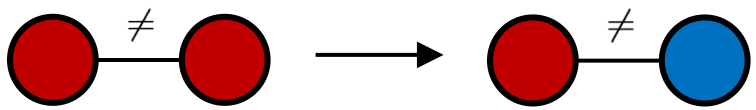
بهبود تکرار شونده
(Iterative Improvement)



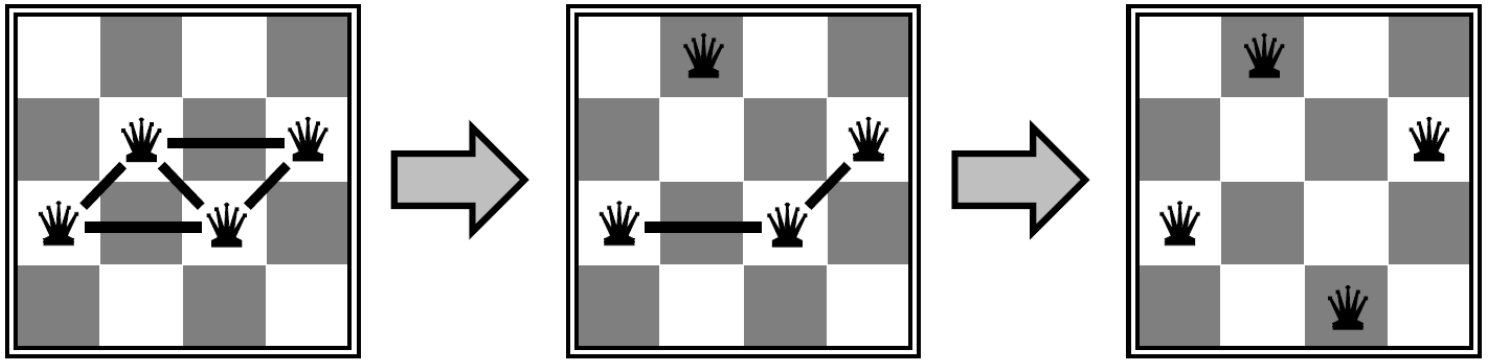
Local Search for CSP

الگوریتم‌های تکرار شونده 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

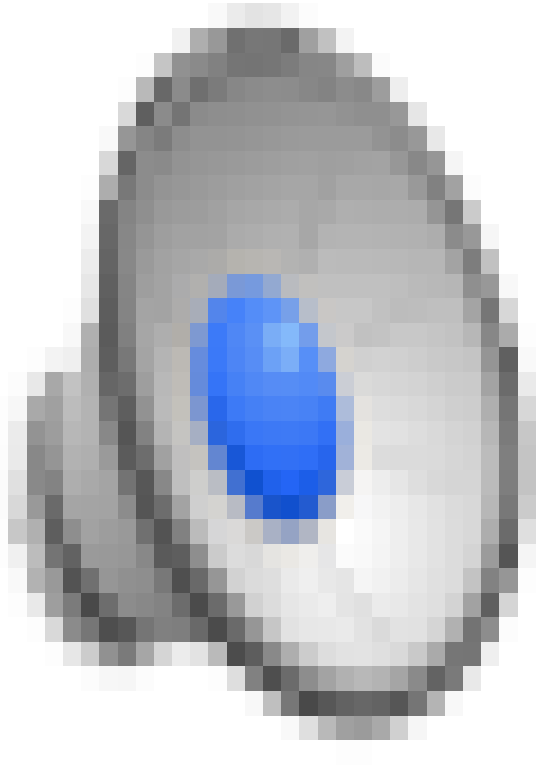


مثال: 4-وزیر

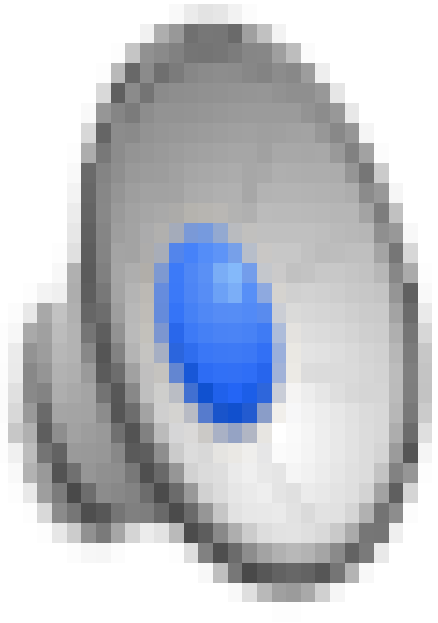


- حالات: 4 وزیر در 4 ستون (4^4 = 256 حالت)
- عملگرها: حرکت ملکه در ستون
- آزمون هدف: وزیرها همدیگر را تهدید نکنند
- ارزیابی: c(n) = تعداد تهدیدها

ویدیوی دموی بهبود تکرارشونده – n-وزیر



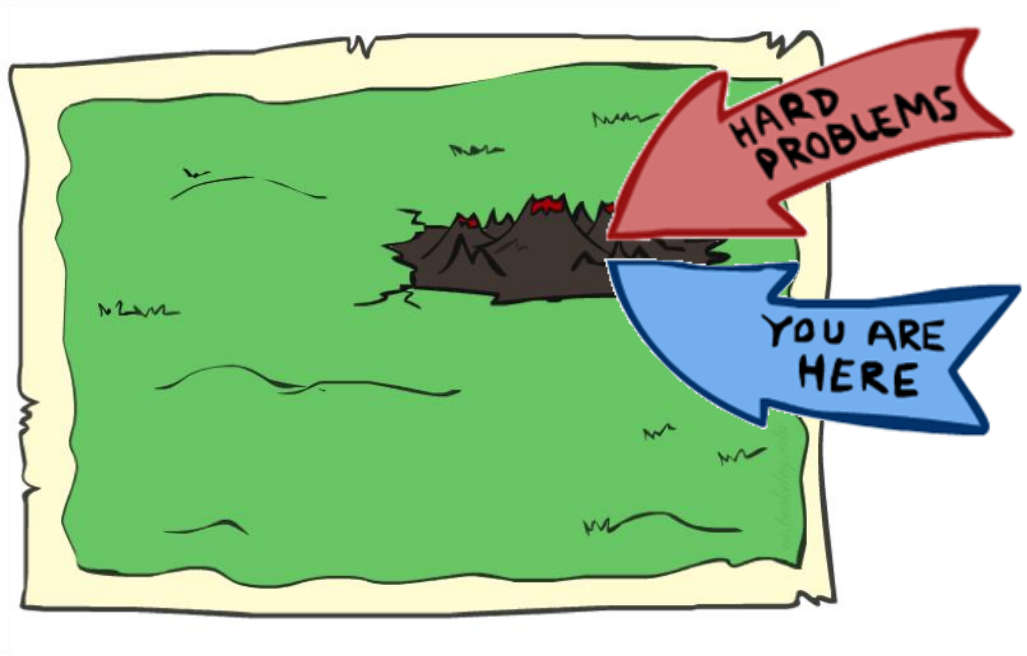
ویدیوی دموی بهبود تکرارشونده – رنگ‌آمیزی



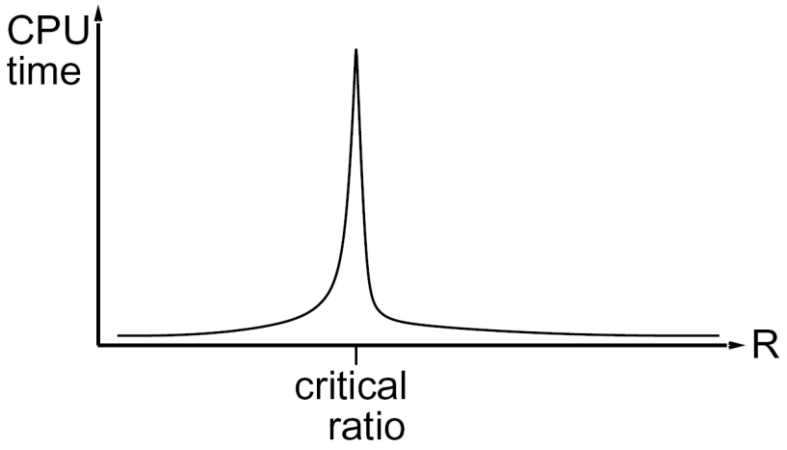
عملکرد Min-Conflict

- با توجه به حالت اولیه تصادفی، می توان 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:

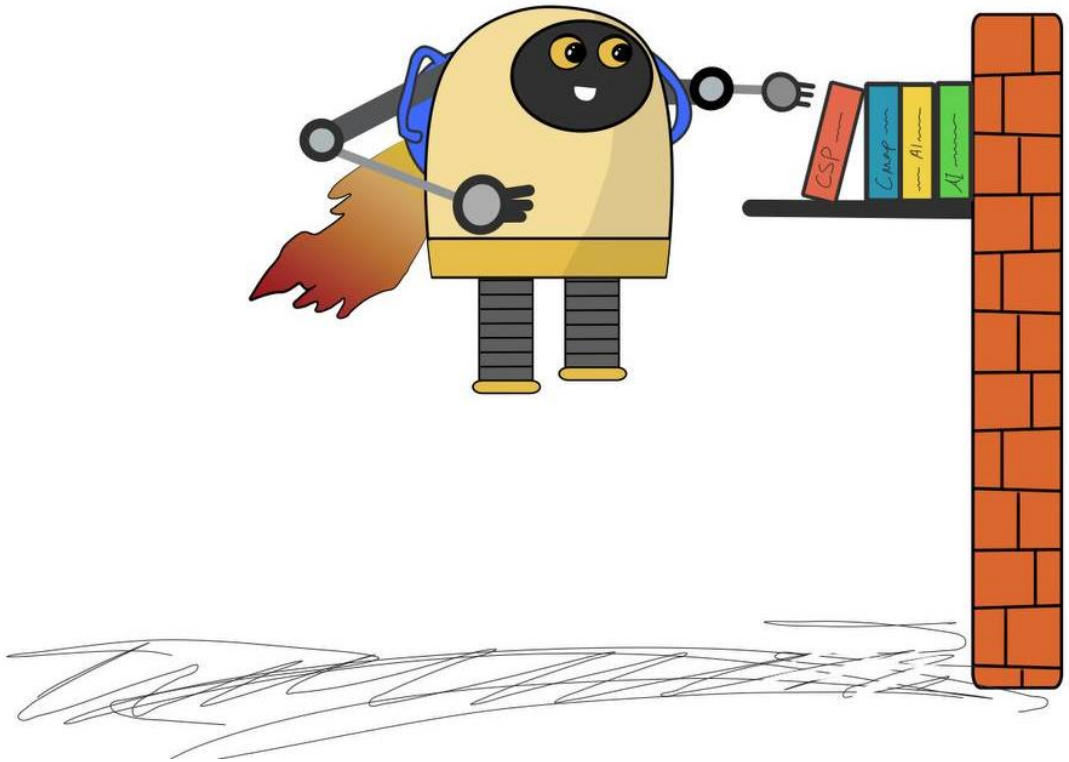


$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



خلاصه: CSPها

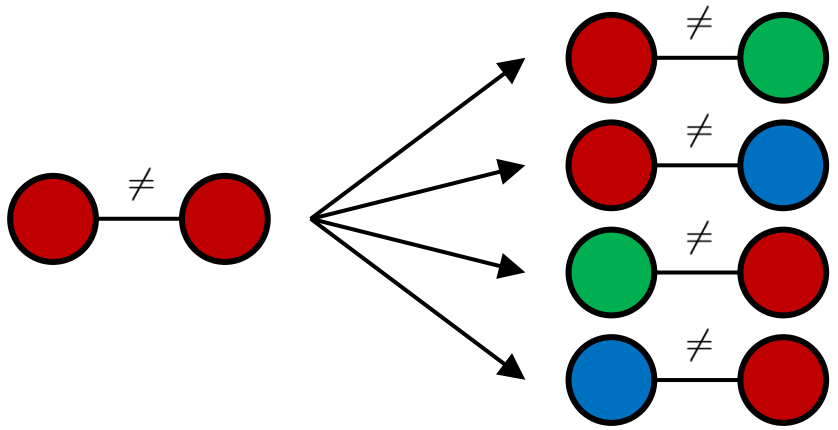
- CSPها نوع خاصی از مسائل جستجو هستند:
  - حالات (states)، انتسابات جزئی هستند.
  - آزمون هدف با محدودیت‌ها تعریف می‌شود.
- راه حل اساسی: جستجوی عقبگرد (Backtracking)
- افزایش سرعت جستجو:
  - مرتب سازی (Ordering)
  - فیلتر کردن (Filtering)
  - ساختار (Structure)
- min-conflictهای تکرارشونده اغلب در عمل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

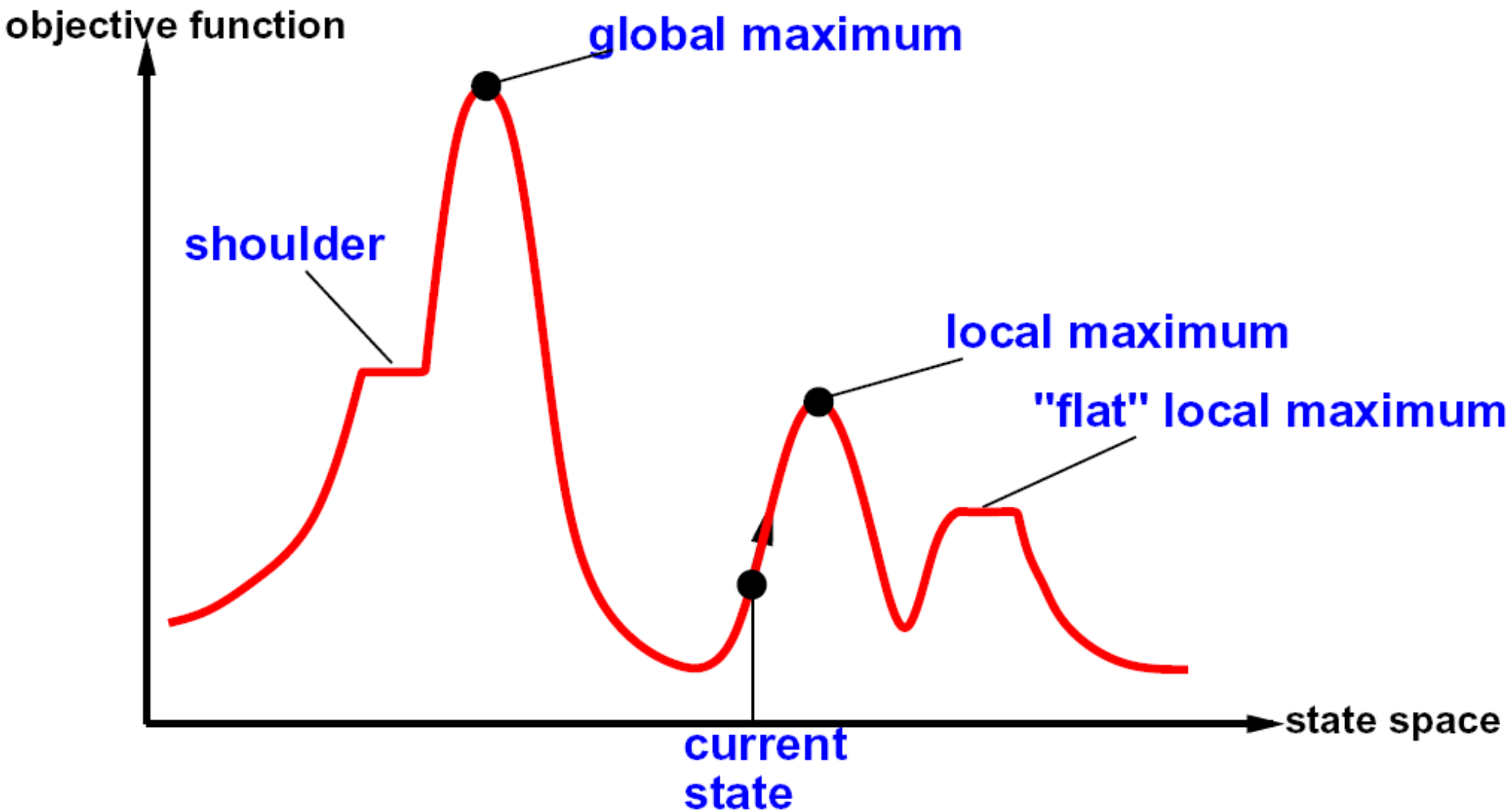


- به طور کلی بسیار سریع‌تر و از نظر حافظه‌ای کم‌هزینه‌تر (اما ناکامل و غیربهینه)

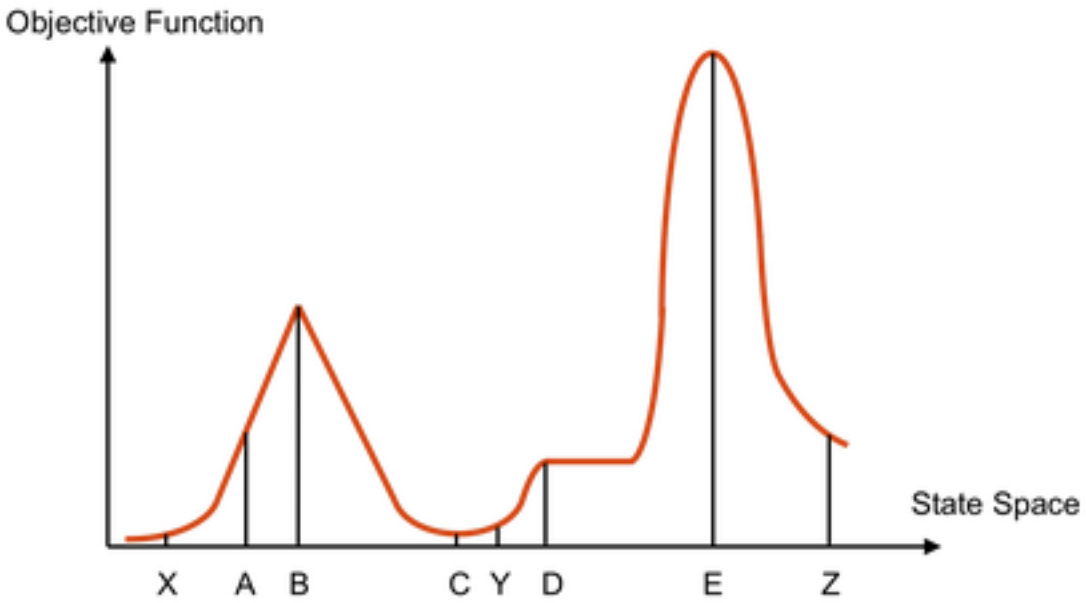
تپه نوردی (Hill climbing)

- 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

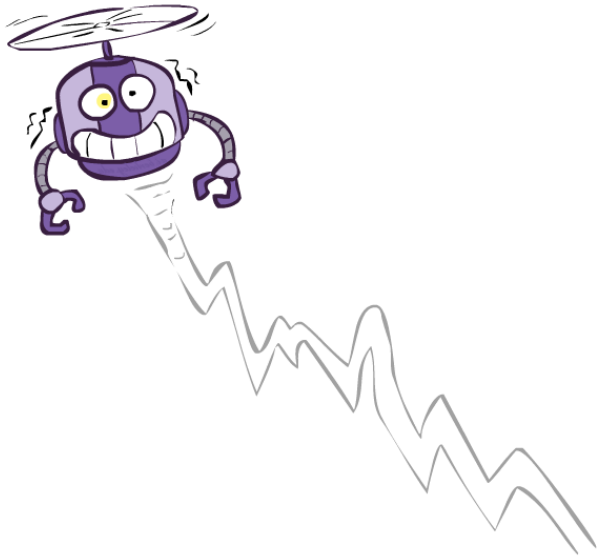


از X شروع کنید، به کجا می رسید؟

از Y شروع کنید، به کجا می رسید؟

از Z شروع کنید، به کجا می رسید؟

تبرید شبیه‌سازی‌شده (Simulated Annealing)

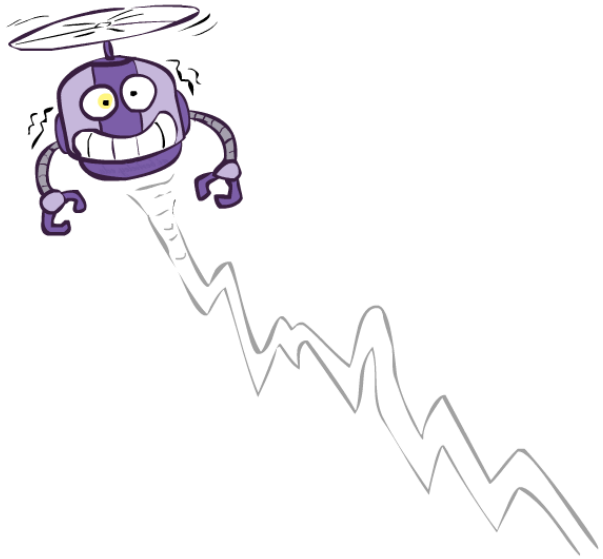


- ایده: با اجازه دادن به حرکت در سراشیبی، از بیشینه‌های محلی فرار کنید
- اما با گذشت زمان آن‌ها را نادرتر کنید

```
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
        ΔE ← VALUE[next] – VALUE[current]
        if ΔE > 0 then current ← next
        else current ← next only with probability e^(Δ E/T)
```

تبرید شبیه‌سازی‌شده

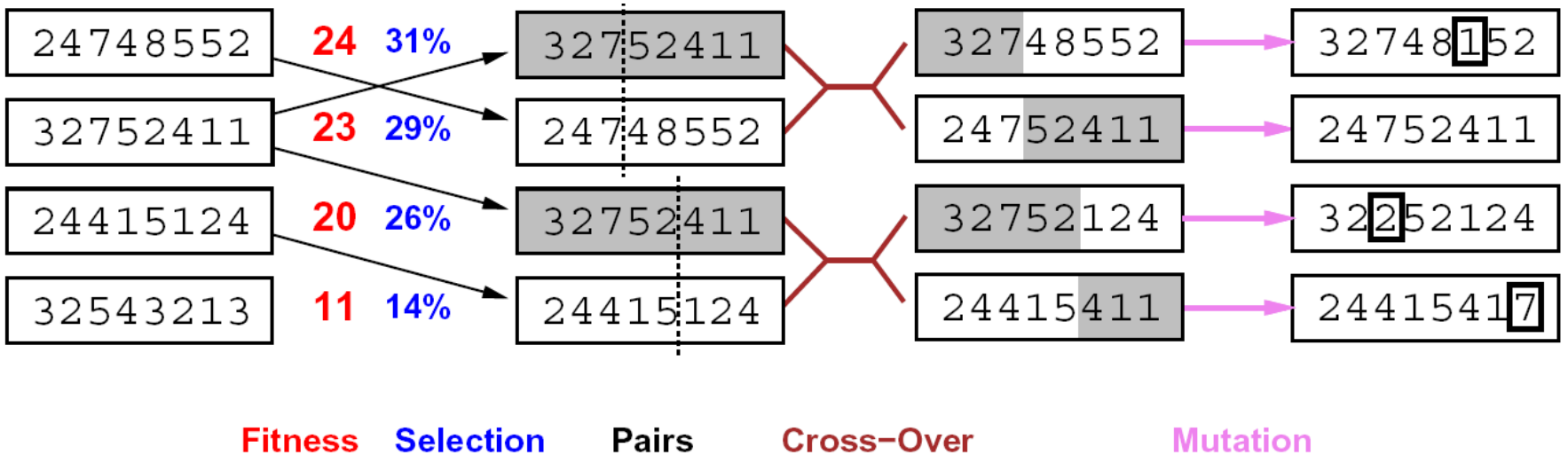


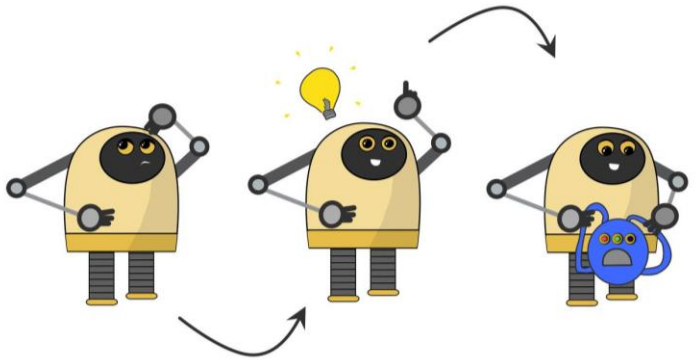
- تضمین تئوری:
  - توزیع ثابت: $p(x) \propto e^{\frac{E(x)}{kT}}$
  - اگر T به اندازه کافی به آرامی کاهش یابد، به حالت بهینه همگرا می شود!

- آیا این تضمین جالبی است؟

- جادو به نظر می رسد، 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

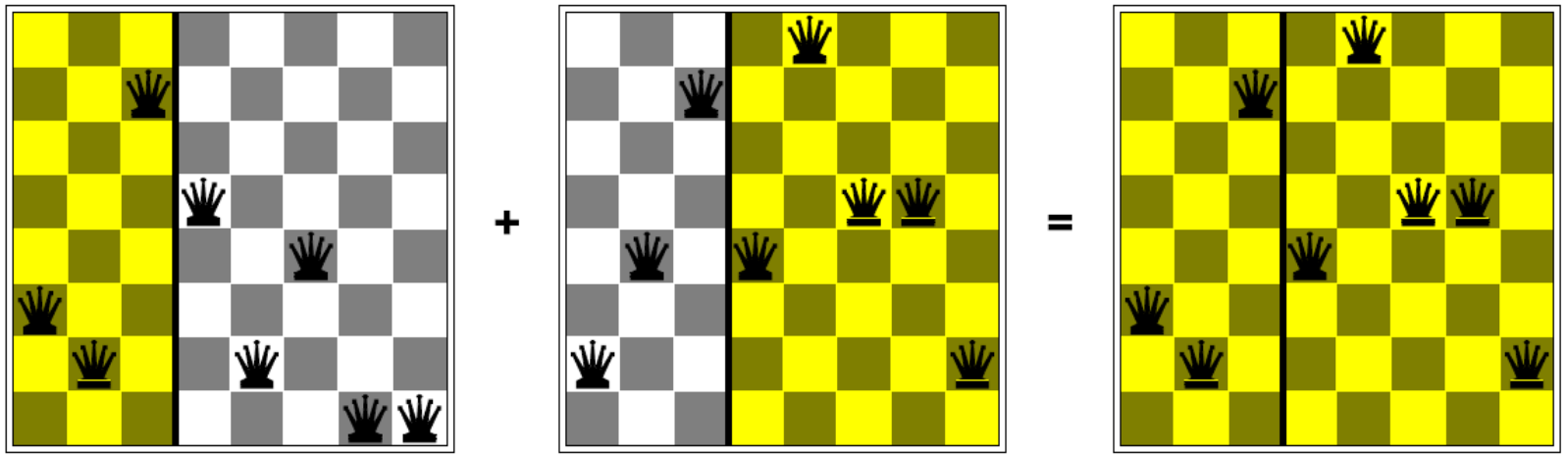




- الگوریتم های ژنتیک از تشبیه انتخاب طبیعی (Natural Selection) استفاده می‌کنند
  - بهترین N فرضیه را در هر مرحله (انتخاب) بر اساس تابع شایستگی (Fitness Function) نگه دارید
  - همچنین دارای عملگرهای متقاطع جفتی(Cross-Over)، با جهش (Mutation) اختیاری برای ایجاد تنوع

- احتمالاً به اشتباه تعبیرشده‌ترین، به اشتباه به کار گرفته‌شده‌ترین (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)