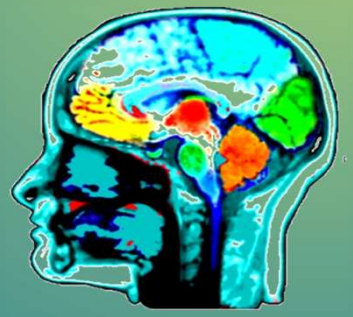
Introduction
To
Artificial Intelligence

Isfahan University of Technology (IUT)

1402



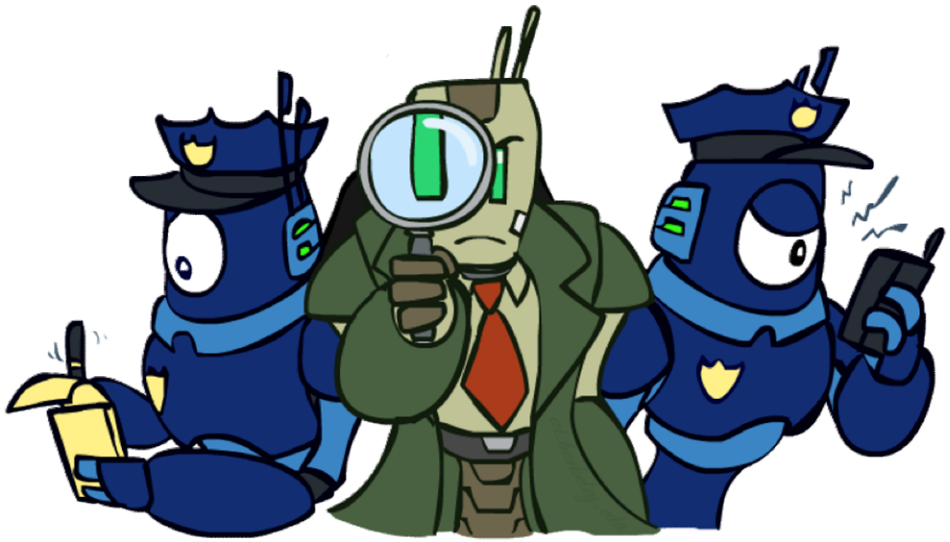
Constraint Satisfaction Problems

Dr. Hamidreza Hakim
hakim@iut.ac.ir

[These slides were created by Dan Klein and Pieter Abbeel for CS188 Intro to AI at UC Berkeley.]


بسم الله الرحمن الرحيم

Constraint Satisfaction Problems



یک وقتایی پیش میاد که توی فرایند سرچ به ما یکسری محدودیت هایی تحمیل شده و ما به دنبال حالت هایی هستیم که یک محدودیت هایی رو برامون تایید بکنن
توی این حالت ها میگیم اون مسئله، مسئله ای است که مبتنی بر قیود باید حل بشه

What is Search For?

- Assumptions about the world: a single agent, deterministic actions, fully observed state, discrete state space

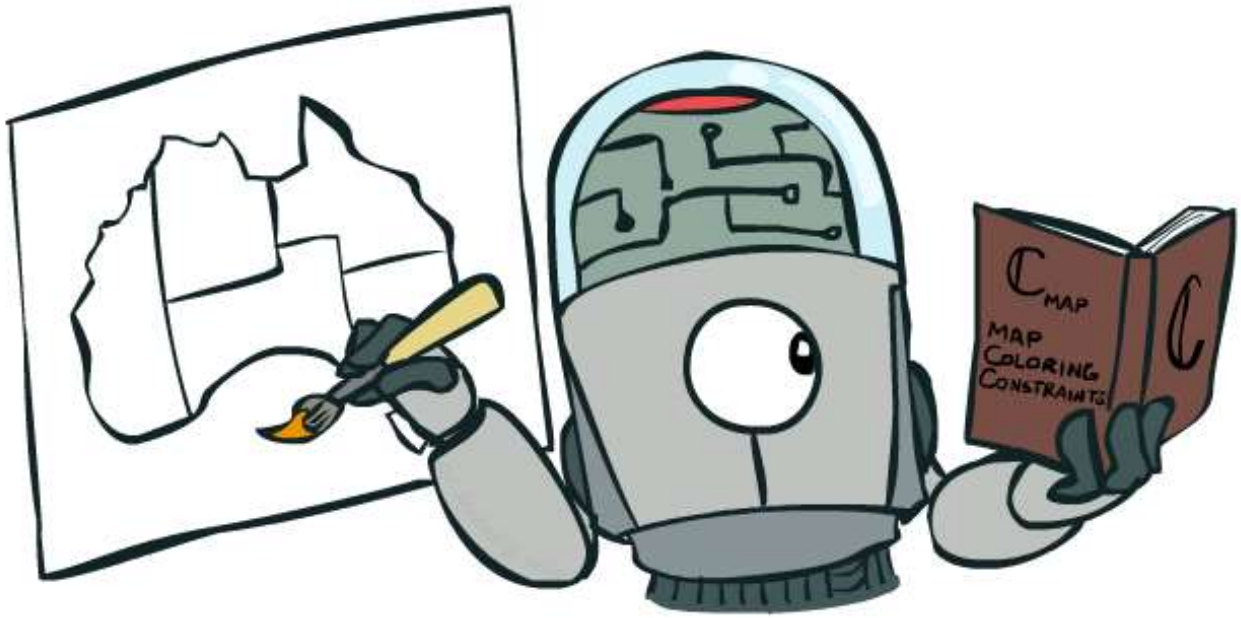
- Planning: sequences of actions
  - The path to the goal is the important thing
  - Paths have various costs, depths
  - Heuristics give problem-specific guidance

- Identification: assignments to variables
  - The goal itself is important, not the path
  - All paths at the same depth (for some formulations)
  - CSPs are specialized for identification problems

مسائلی که تا الان باهاشون سر و کار داشتیم جنس مسائل Planning بودن انگار ما یک سری حالت هایی داشتیم ینی یک عامل قرار بود توی یکسری حالت هایی این جستجو رو انجام بده و از یه جایی شروع میکرد تا به یک هدف برسه و قبل از اینکه کارش رو شروع بکنه می اومد یک مسیر رو پیدا میکرد از اون حالت ابتدایی تا به اون هدف برسه و مسیر براش مهم بود و حتی هزینه رسیدن به اون هدف هم براش مهم بود و در کنار این جستجویی که داشت یک هیوریستیک فانکشنی بهش کمک میکرد که فاصله اش رو تا اون هدف مشخص بکنه و از این مسیر بریم یا نریم به این مسائل می گیم مسائل Planning یا برنامه ریزی

توی مسائل Identification ها ما فقط اون هدف برامون مهمه میخوایم یک شرایطی پیدا بکنیم که به اون هدف برسیم و مسیر برامون مهم نیست

Constraint Satisfaction Problems



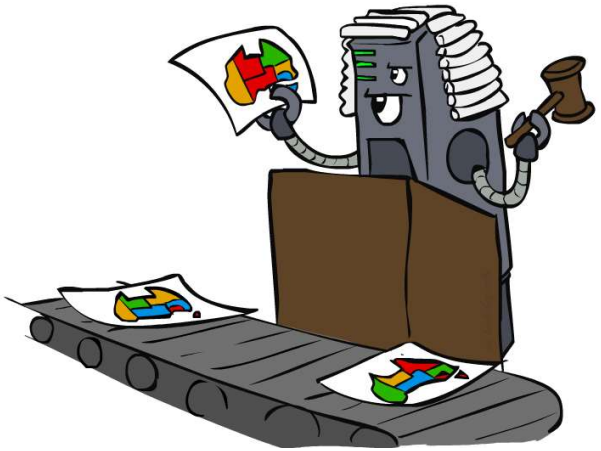
Constraint Satisfaction Problems

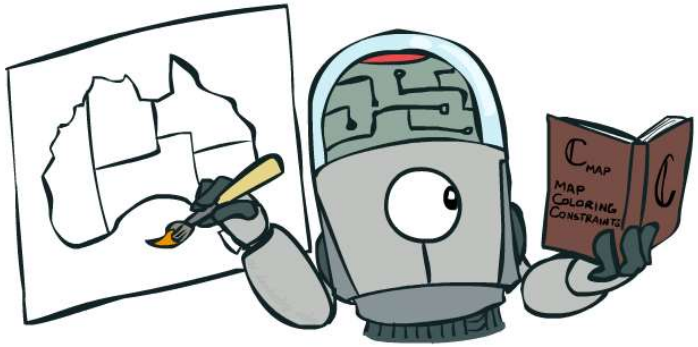
- Standard search problems: سرچ عادی
  - State is a “black box”: arbitrary data structure
  - Goal test can be any function over states
  - Successor function(reachable State) can also be anything

- Constraint satisfaction problems (CSPs):
  - A special subset of search problems
  - State is defined by variables X_i with values from a domain D (sometimes D depends on i)
  - Goal test is a set of constraints specifying allowable combinations of values for subsets of variables

- Simple example of a *formal representation language*

- Allows useful general-purpose algorithms with more power than standard search algorithms





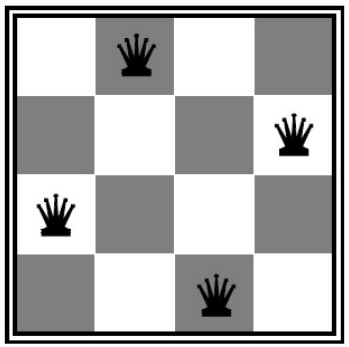
csp:
ما یکسری استیت داشتیم که استیت ها مثل یک حالت black box بودن ینی یک نمایی برامون بودن که این استیته چقدر خوبه و این استیت رو چک میکردیم با یک تابعی
یک Successor function داشتیم که این بهمون میگفت اگر این حالت هستیم به چه حالتی می رویم یا یک استراتژی برای حالت ساختنمون بود

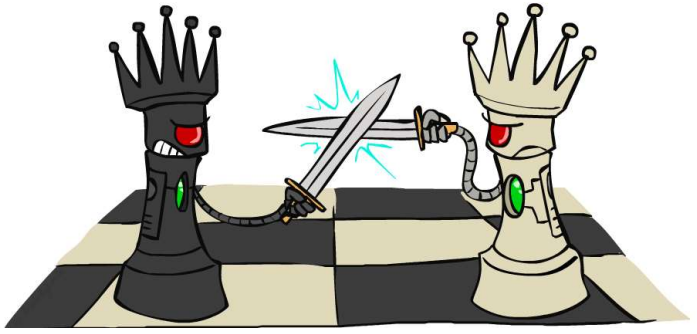
توی csp:
دنبال این هستیم که دیگه نریم همه حالت ها رو بررسی کنیم بلکه یک فضای کوچیکی از این حالت ها رو نگاه بکنیم --> اینجا یک چیزی داریم به نام variables و میایم دنیا رو با اون variables و یک domain مشخص می کنیم
یکسری variables داریم که این variables ها تحت تعریف مقادیری که پیدا می کنن ینی تحت تاثیر یک domain مقدار می گیرن
و این متغییرها یک دامنه مقدار دارن که بهش می گن domain
و یک تابع goal test هم داریم که بهمون میگه Constraint هایی که توی variables شده الان ارضا شده که این Constraint ها می تونه روی یک تعدادی از variables تعریف بشه

می تونیم تک تک این خونه ها رو به عنوان
یک متغییر در نظر بگیریم

Example: N-Queens

- Formulation 1:
  - Variables: X_{ij}
  - Domains: $\{0, 1\}$
  - Constraints





find a configuration in which to place N queens on the board such that no two queens attack each another

$$\forall i, j, k \quad (X_{ij}, X_{ik}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{kj}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{i+k,j+k}) \in \{(0,0), (0,1), (1,0)\}$$
$$\forall i, j, k \quad (X_{ij}, X_{i+k,j-k}) \in \{(0,0), (0,1), (1,0)\}$$
$$\sum_{i,j} X_{ij} = N$$

مثال:
توی n وزیر دنبال این بودیم که موقعیت چینش 4 تا وزیر توی یک فضای 4*4 پیدا بکنیم که این وزیرها همدیگر رو نزنن --> صرفا پیدا کردن جواب نهایی برامون کافی بود

حالا میخوایم اینو تبدیل بکنیم به یک مسئله csp:
اول باید بیایم Variables ها رو تعریف بکنیم:
Variables ها موقعیت هر کدوم از این خونه ها هست
مثلا این صفحه یک موقعیت i , j داره که یا وزیر توش قرار میگیره یا نمی گیره

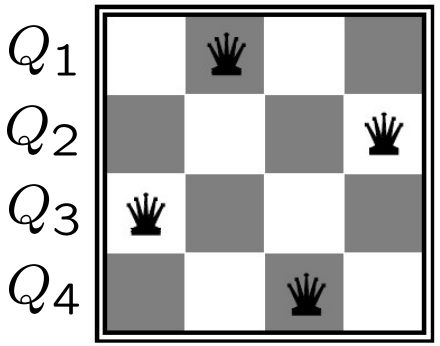
Constraints: Constraints ها شرایطی است که اگر ما به اون استیت هدف رسیدیم میگه که مسئله رو حل کردیم

اولی: اگر هر نقطه ای رو برداشتی ینی نقطه ای که i,j اش مشخصه و با یک نقطه دیگه خواستی Constraints رو نگاه بکنی --> اگر قرار باشه توی یک نقطه ای مثل i,j وزیری وجود نداشته باشه توی همه نقاطی که هم سطر اون هست ینی i,k میشن همه نقاطی که از لحاظ سطری با این نقطه یکی هستن فقط ستونشون با هم فرق داره توی همه اون نقاط یا می تونی وزیر بذاری یا نذاری ولی اگر توی یک نقطه ای ینی نقطه i وزیر بود دیگه نمی تونه توی بقیه نقاط که هم سطر اون هستن وزیر بذاری

Example: N-Queens

- Formulation 2:
  - Variables: Q_k

و یا هم می تونیم سطر یا ستون رو به عنوان متغییر در نظر بگیریم



  - Domains: $\{1, 2, 3, \ldots N\}$
  - Constraints:

غیر تهدید کننده باشن

Implicit: $\forall i, j$ non-threatening(Q_i, Q_j)

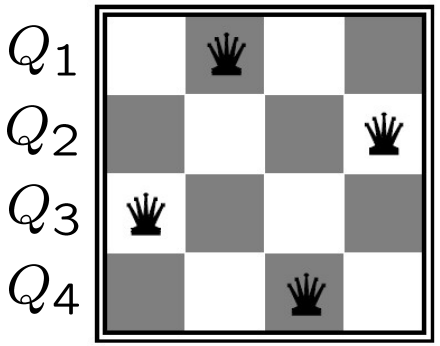
Explicit: $(Q_1, Q_2) \in \{(1, 3), (1, 4), \ldots\}$

$\cdots$

Constraints رو باید روی Variables تعریف بکنی

Example: N-Queens

- Formulation 2:
  - Variables: Q_k
  - Domains: $\{1, 2, 3, \ldots N\}$
  - Constraints:



Implicit:

- $Q_i = k \rightarrow Q_j \neq k$
  - for all j = 1 to N, j≠i
- $Q_i = k_i, Q_j = k_j \rightarrow |i-j| \neq |k_i - k_j|$
  - for all j = 1 to N, j≠i

CSP Examples

given a set of colors and must color a map such that
no two adjacent states or regions have the same color.

{red, green, blue}

مثال:
می خوایم این نقشه رو رنگ امیزی بکنیم و نحوی رنگ امیزی این نقشه باید به صورتی باشه که شهرهای مجاور هم رنگشون یکی نباشه

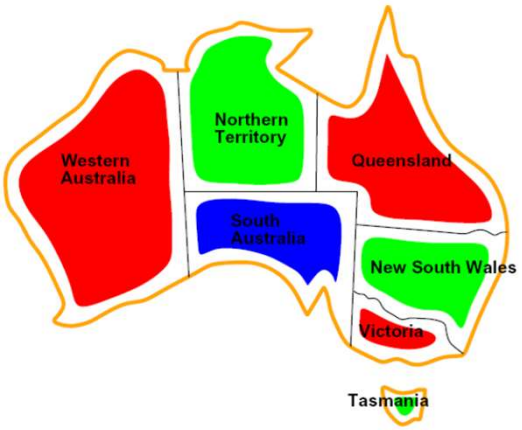
Example: Map Coloring

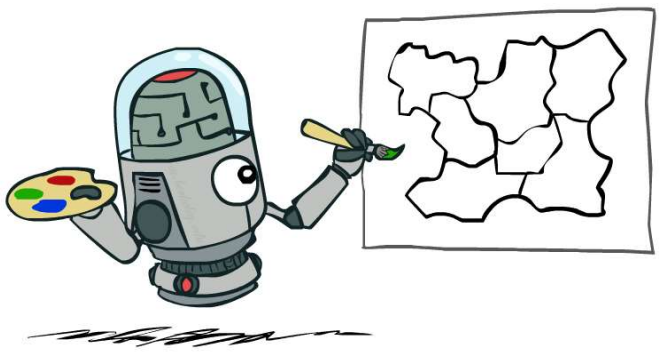
- Variables: $\mathrm{WA},\ \mathrm{NT},\ \mathrm{Q},\ \mathrm{NSW},\ \mathrm{V},\ \mathrm{SA},\ \mathrm{T}$
- Domains: $\mathrm{D} = \{\mathrm{red}, \mathrm{green}, \mathrm{blue}\}$
- Constraints: adjacent regions must have different colors

  Implicit: $\mathrm{WA} \neq \mathrm{NT}$

  Explicit: $(\mathrm{WA}, \mathrm{NT}) \in \{(\mathrm{red}, \mathrm{green}), (\mathrm{red}, \mathrm{blue}), \ldots\}$
- Solutions are assignments satisfying all constraints, e.g.:

  $\{\mathrm{WA}{=}\mathrm{red},\ \mathrm{NT}{=}\mathrm{green},\ \mathrm{Q}{=}\mathrm{red},\ \mathrm{NSW}{=}\mathrm{green},$
  $\mathrm{V}{=}\mathrm{red},\ \mathrm{SA}{=}\mathrm{blue},\ \mathrm{T}{=}\mathrm{green}\}$





به دید csp:
متغییرها: شهرها می تونن باشن --> قراره به هر شهری یک مقداری رو نسبت بدیم که این مقدار رنگ هست

دامنه: رنگ هایی که می تونیم به هر کدوم از این شهرها بدیم

شرط: این متغییرها رنگ هاشون مثل هم نباشه
به صورت Implicit: هر مقداری که متغییر wa داشت متغییر nt نمی تونه داشته باشه و به همین صورت برای بقیشون هم می تونیم بگیم

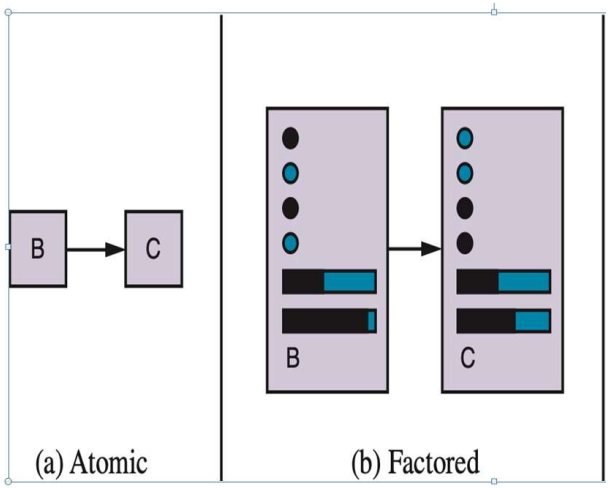
به صورت Explicit: زوج هایی که می تونه برای شهرها باشه رو بنویسیم به صورت واضح

جواب نهایی:
همینی که الان نوشته --> این یکی از جواب ها میشه

نکته: اگر اینو بخوایم با تکنیک های قبلی حل بکنیم چی کار باید بکنیم؟
باید یک حالتی تعریف می کردیم ینی استیت استیتی تعریف می کردیم و بعد یک استیت هدف داشتیم که می گفته این چینش رنگه و اونجا باید می اومدیم حالت های مختلف رنگ امیزی تک تک این ها و سرچ می کردیم یا به صورت لوکال سرچی یا به صورت A*
چه اتفاقی برای فضای حالت پیش می اومد اگر می خواستیم رندوم اینو سرچ بکنیم ؟ هر خونه ای 3 تا حالت داشت و ما 7 تا شهر داریم ینی 3 به توان 7 حالت رو باید سرچ می کردیم توی بدترین حالت که به جواب برسیم ینی همه این حالت ها رو باید سرچ می کردیم تا به اون حالت نهایی برسیم حالا توی این csp می خوایم از این 3 به توان 7 فاصله بگیریم

Why CSP Solver?

- Constraint satisfaction problems are NP-hard
- Inform and uniform search are atomic state-space search
  - we can only ask
    - is this specific state a goal? No?
    - What about this one?



نکته:
مسائل csp از جمله مسائل np hard هستن --> همین مثال قبلی ما باید 3 به توان 7 حالت رو انالیز بکنیم و دیگه این فضای سرچ چند جمله ای نیست و فضای سرچ خیلی بزرگ است

اینجا میخوایم یکم اون بحث سرچ رو هدفمندتر بکنیم

Why CSP Solver?

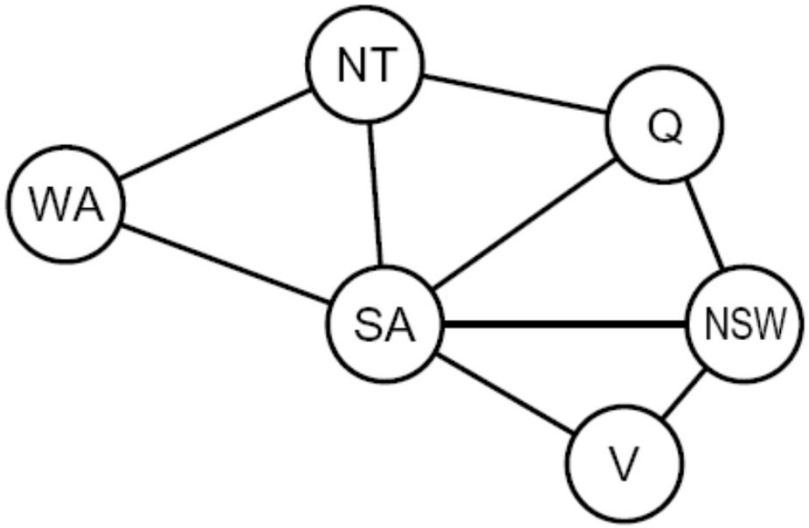
CSP solvers

- Define a natural representation for a wide variety of problems;
  - It is often easy to formulate a problem as a CSP.
  - Define general Successor function(reachable State) and Goal Test
  - Define General Efficient Heuristic (without special knowledge about Problem details)
- Be fast and efficient (Years of development).
  - A CSP solver can quickly prune large swathes of the search space that an atomic state-space searcher cannot.
  - 3^5=243 assignments for the five neighboring variables . in CSP 2^5=32 (87% reduction!)
- Be factored representation (for each state: a set of variables, each of which has a value)
- find out that a partial assignment violates a constraint, we can
  - Discard further refinements of the partial assignment.
  - Why the assignment is not a solution—
  - We see which variables violate a constraint—
  - So we can focus attention on the variables that matter.

توی چارچوب CSP Solver ها:
ما میایم یک قالبی ارائه میدیم برای اینجور مسائل که دیگه درگیر هیوریستیک فانکشن نباشیم ینی مسئله رو ببر به قیدهاش و قیدهاشو از خود اون استیت هدف جدا بکن و بعد این ها بهمون کمک میکنن که با بررسی این قیدها از این حالتی که شروع کردیم چجوری توی کمترین زمان برسیم به حالت هدف --> پس با حداقل دانش راجع به مسئله می تونیم اون مسئله رو حل کنیم چون
هیوریستیک فانکشن توی خیلی از مسائل برمی گشت به دانش مسئله ینی مثلا مسئله مسیریابی داریم یا ... ولی اینجا دیگه درگیر این ها نمیشه
پس یک چارچوب کلی برای هدف یابی داریم
یک تعریف جنرالی داریم --> از هر حالتی به حالت بعدی بریم که به استیت هدف برسیم اون رو بهمون توی Csp ها اشاره میکنه
چندین سال هم این الگوریتم ها توسعه پیدا کردن و می تونن توی مرتبه زمانی خیلی خوبی ما رو به استیت هدف برسونن --> مثلا برای همین مسئله رنگ امیزی ما سه تا رنگ داریم برای 5 تا شهر و تعداد حالت هایی که می تونیم این ها رو مقدار دهی بکنیم میشه 243 که با تکنیک های csp این مرتبه خیلی کاهش پیدا می کنه

یک ویژگی خوب دیگه که Csp ها دارن اینه که:
وقتی که ما میخوایم بین استیت ها جابه جا بشیم و فرایند جستجوی بعدی رو بسازیم می تونیم از اشتباهات گذشتمون درس بگیریم ینی شهرهارو اینجوری رنگ امیزی کردیم و بعد به این نتیجه رسیدیم از این مسیر رنگ امیزی اشتباه است و از این درس می گیریم و بعضی از حالت ها رو دیگه ادامه نمی دیم

Constraint Graphs



رنگ این تک جزیره می تونه هر چی باشه
چون هیچ قیدی نداریم باهاش

اولین گام برای حل مسائل csp:
اینه که یک گراف ازش ایجاد بکنیم --> این گراف خیلی به کارمون میاد --> یک گراف روی متغییرها تعریف بکنیم و نحوه ساختن این گراف بر این مبنا است هر نودش متغییرهای ما هست و اتصال بین این متغییرها نشان دهنده یک قید است
مثلا اینجا قیدمون برابر نبودن است

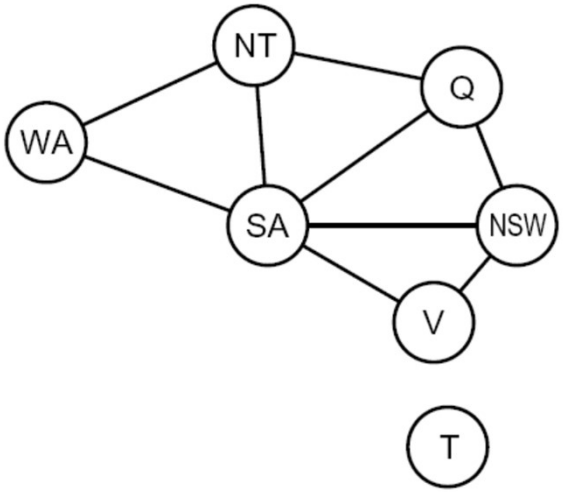
چرا گراف ساختیم؟ می خوایم سریعتر شرایط بد رو بفهمیم و با بحث همسایه بودن یا نبودن میفهمیم که باید به این همسایه نگاه بکنیم یا نکنیم توی قید

Constraint Graphs

- Binary CSP: each constraint relates (at most) two variables
- Binary constraint graph: nodes are variables, arcs show constraints
- General-purpose CSP algorithms use the graph structure to speed up search.(**why**?)

E.g., Tasmania is an independent sub-problem!

Big Graph convert to Two Small Graph



برای گراف می تونیم تیپ های مختلف داشته باشیم:
باینری csp:
ینی بین هر دوتا متغییر یک یالی (این یال نشون دهنده csp ما است) است و دوبه دو است

Varieties of CSPs and Constraints

Varieties of Constraints

- Varieties of Constraints
  - **Unary constraints** involve a single variable (equivalent to reducing domains), e.g.:

    $SA \neq green$

  - **Binary constraints** involve pairs of variables, e.g.:

    $SA \neq WA$

  - **Higher-order constraints** involve 3 or more variables:
    e.g., cryptarithmetic column constraints

- Preferences (soft constraints):
  - E.g., red is better than green
  - Often representable by a cost for each variable assignment
  - Gives constrained optimization problems
  - (We'll ignore these until we get to Bayes' nets)

انواع Constraints ها:
بعضی وقت ها Constraints به صورت یونری است ینی مثلا این شهر هیچ وقت سبز نباشه --> ینی راجع به یک متغییر یک قید خاص داریم
بعضی وقت ها قیود دو متغییره است --> ینی دوتا متغییر رو داره بهم مرتبط می کنه ینی هر مقداری این گرفت مثلا اون یکی باید سه مقدار بیشتر بگیره
یک تیپ دیگه از قیدها موقع هایی است که ما بیشتر از دوتا متغییر رو داریم ینی مثلا سه تا متغییر با هم در ارتباط می شن --> اینجا یک ذره چالش داریم چون سه تا متغییر رو نمی تونیم توی گراف با یال اینا نشون بدیم --> یک تکنیکی داره میان یک متغییر کمکی اضافه می کنن توی مثال بعدی میگه --> اگر بیش از دوتا متغییر در داخل یک قید وجود داشته باشه ما چجوری اون قید رو با یک یال نشون بدیم

توی بعضی از مسائل ما میخوایم بگیم که ترجیح میدیم این متغییر این مقدار رو بگیره --> مثلا توی زمان بندی کلاس ما یک تعداد نفر داریم و قراره به یکسری منابع دست پیدا بکنند و ترجیحمون این است که این استاد توی این ساعت تدریس بکنه --> برای این ها میان معمولا یک تابعی تعریف می کنن و اون تابع میاد بهمون میگه و دیگه به سادگی اینا نیست که ما بگیم خب برای این متغییر این مقدار رو داشته باشیم یا نداشته باشیم پس یک تابعی است

Example: Cryptarithmetic(Several Variable)

- Variables:

  $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$

- Domains:

  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

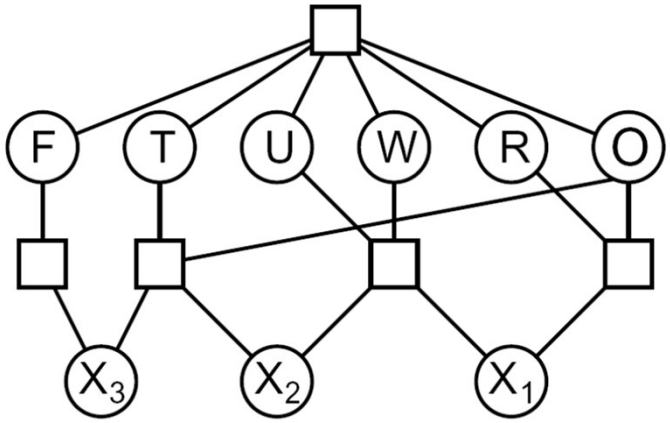
- Constraints:

  alldiff(F, T, U, W, R, O)

  $O + O = R + 10 \cdot X_1$

  $\cdots$ $X_1 + W + W = U + 10 \cdot X_2$
  $X_2 + T + T = O + 10 \cdot X_3$
  $X_3 = F,\ T \neq 0,\ F \neq 0$

```
  T W O
+ T W O
-------
F O U R
```

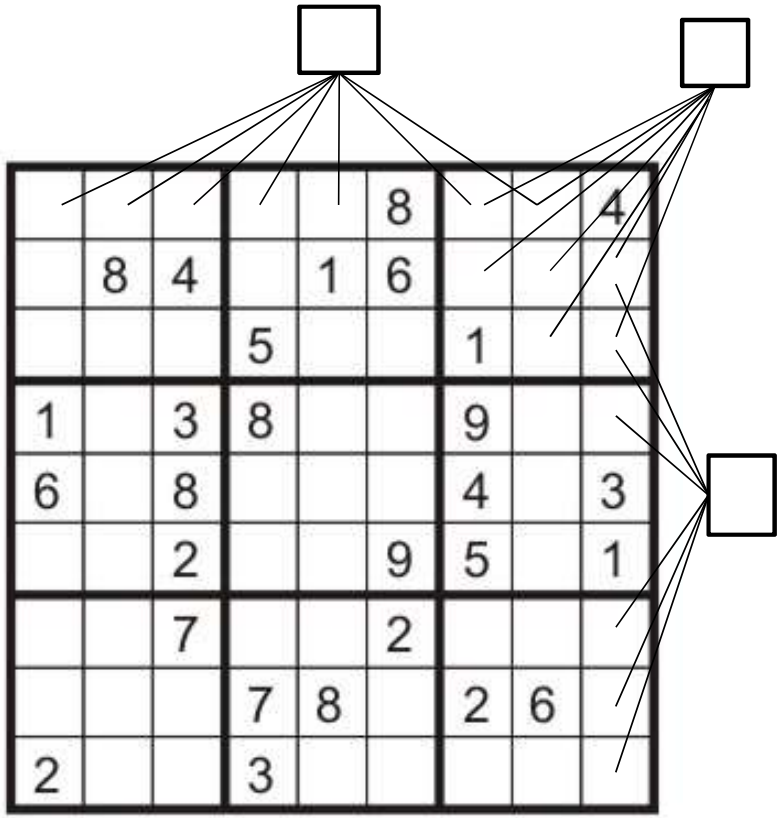


مثال:
مسئله حساب رمزی:
قراره که ما یکسری مقادیری برای این حروف پیدا بکنیم که توی این قیده بخونه
ما یکسری حروف داریم --> T W O U R F
می خوایم بین اعداد 0 تا 9 یکسری مقادیر بذاریم توی این حروف طوری که ما اگر این حروف رو به صورت اسمی نوشتیم این عملیات توشون صادق باشه ینی هر مقداری توشون گذاشتیم نتیجه دو دو بده 4
متغییرهای ما اینجا میشه: که نوشته اینجا
دامنه هم نوشته توی صفحه

قید:
مقادیر F T U W R O باید متفاوت باشه
x1, x2, x3 این ها متغییرهای کمکی ما هستن --> این متغییرهای کمکی برای ارتباط های سطح بالاتر است
این متغییرهای کمکی اون عملیات کری رو داره مدیریت میکنه ینی اینکه ما میخوایم یک واحدی توسعه پیدا بکنه بره به مرحله بالاتر
پس ما اومدیم یک مسئله با قیود بیشتر از دوتا متغییر رو تبدیل کردیم به همچین گرافی که یکسری نودهای کمکی داره

نهایتا ما دنبال این هستیم که مقادیر مختلف رو روی این گراف حفظ بکنیم و ببینیم چه مجموعه مقادیری روی این گراف صدق می کنه

Example: Sudoku



- Variables:
  - Each (open) square
- Domains:
  - {1,2,…,9}
- Constraints:

9-way alldiff for each column

9-way alldiff for each row

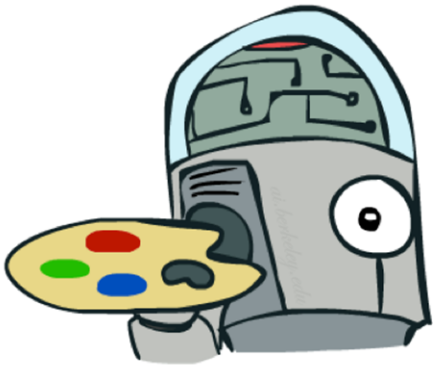
9-way alldiff for each region

(or can have a bunch of pairwise inequality constraints)

یک مسئله دیگه از چند قیدی بودن:
برای سطر و ستون و مربعی می تونیم یک متغییر کمکی در نظر بگیریم و این ها رو چک بکنیم
ما میدونیم که توی هر سطری باید یک دنباله اعداد 1 تا 9 قرار بدیم و عددها باید متمایز باشه و نمیشه یکسان باشه و توی هر ستون و هر سطر و هر خونه مربعی باید عددها متمایز باشه و تکراری نباشه

Varieties of CSPs

- Discrete Variables
  - Finite domains
    - Size d means $O(d^n)$ complete assignments(d different values)
    - E.g., Boolean CSPs, including, N-Queens, Boolean satisfiability (NP-complete)
  - Infinite domains (integers, strings, etc.)
    - E.g., job scheduling, variables are start/end times for each job
    - Linear constraints solvable, nonlinear undecidable



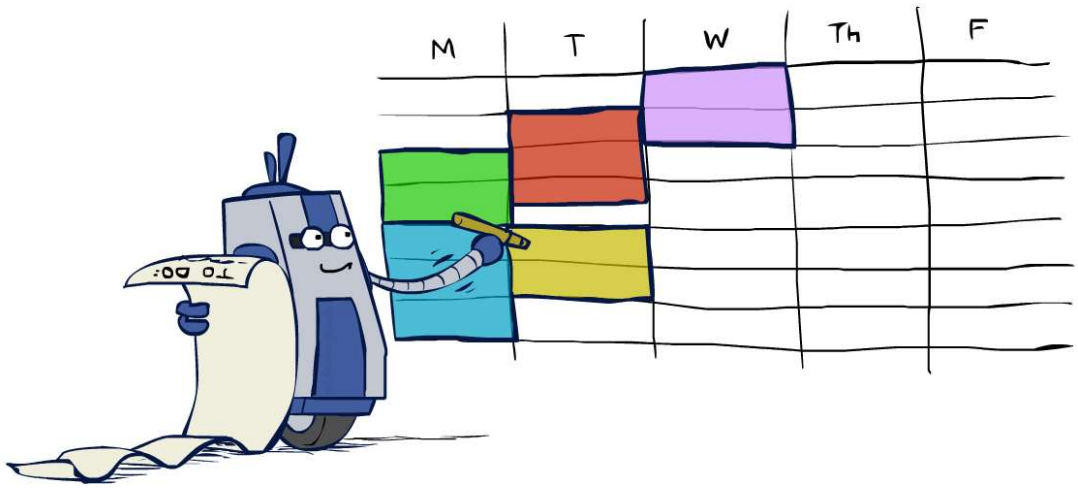
- Continuous variables
  - E.g., start/end times for Hubble Telescope observations
  - Linear constraints solvable in polynomial time by LP methods

خود csp ها چند دسته دارن:
از دید متغییرها می تونیم یک دسته بندی بکنیم که الان متغییرها قراره گسسته باشن یا پیوسته
متغییرهای گسسته هم دوتا تیپ دارن --> مقادیری که میخوایم نسبت بدیم به تک تک این متغییرها
مقادیر محدود هستن و اگر محدود باشه فضای سرچ مشخص است و csp ها بیشتر دارن روی این متمرکز میشن
در کنار اون فضای گسسته محدود یک فضای نامحدود هم وجود داره مثل جاب اسکجول ینی ما چند تا کار رو با یک رویه ای انجام بدیم ینی این کار بعد از این کار باید انجام بشه یا ... --> که دنباله مقادیری که متغییرها می تونن بگیرن دنباله لزوما محدودی نیست چون مقادیر می تونه خیلی بزرگ بشه
اینجا دوتا تیپ مسئله داریم بعضی هاشون رو می تونیم حل بکنیم و بعضی هاشون اصلا قابل حل نیستن که تحت عنوان solvable و undecidable این ها رو تقسیم بندی می کنن

کنار متغییرهای گسسته، متغییرهای پیوسته رو هم داریم

Real-World CSPs

- Assignment problems: e.g.,
  who teaches what class
- Timetabling problems: e.g.,
  which class is offered when and where?
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Circuit layout
- Fault diagnosis
- … lots more!



- Many real-world problems involve real-valued variables…

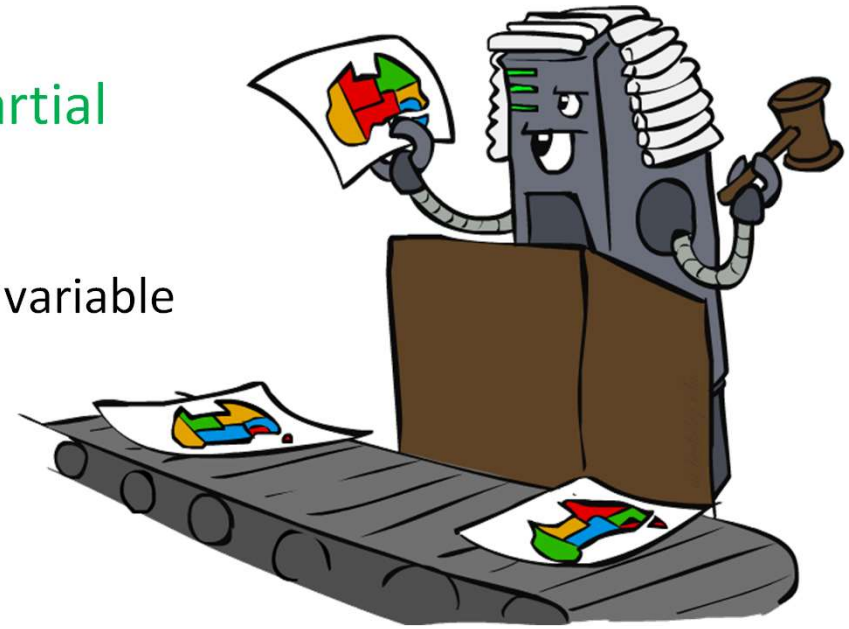
مثال واقعی csp:

Solving CSPs

Standard Search Formulation

Standard search formulation of CSPs

- States defined by the values assigned so far (partial assignments)
  - Initial state: the empty assignment, {}
  - Successor function: assign a value to an unassigned variable
    - Regardless of Constraint (if impossible -> Failed)
  - Goal test:
    the current assignment is complete and satisfies all constraints
- Constraint define
  - Implicit
  - Explicit



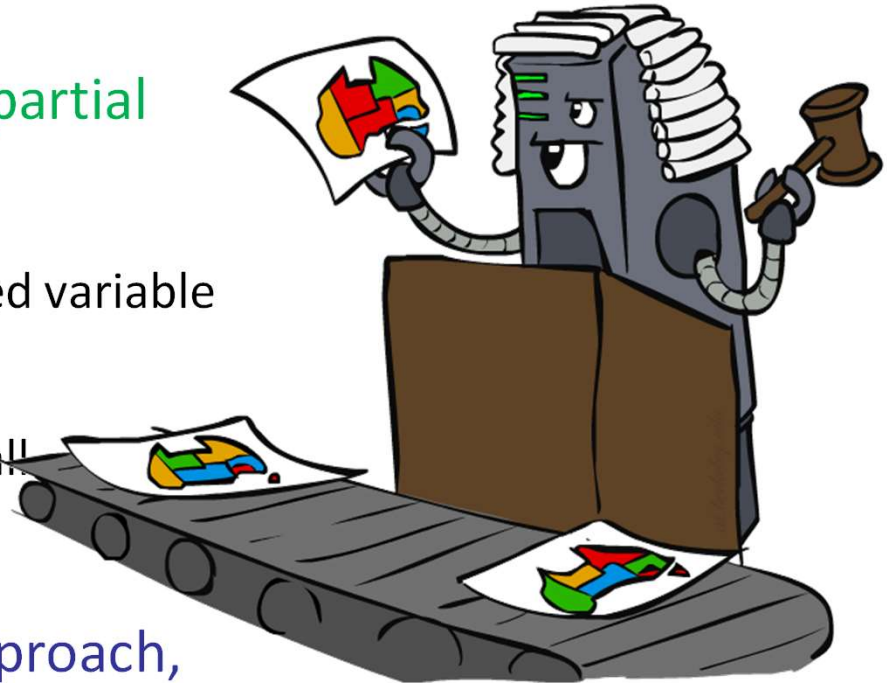
ساختار اولیه برای حل کردن اون ها:
یک حالت اولیه ای تولید میکنیم و توی اون حالت اولیه ما هیچ کدوم از این استیت ها رو مقدار ندادیم و یک Successor function تعریف میکنیم که بهمون میگه اینی که مقدار دادیم به واسطه این بعد ما چه متغییرهایی رو قراره در ادامه این رنگ امیزی بکنیم توی این مسئله رنگ امیزی و نهایتا بعد از اینکه همه متغییرهای مقدار گرفت ما یک goal test داریم که میگیم ایا این به حالت نهایی رسیده یا نه

Constraint هم ما می تونیم به صورت اون گراف تعریفش بکنیم

Standard Search Formulation

Standard search formulation of CSPs

- States defined by the values assigned so far (partial assignments)
  - Initial state: the empty assignment, {}
  - Successor function: assign a value to an unassigned variable
    - Regardless of Constraint (if impossible -> Failed)
  - Goal test:
    the current assignment is complete and satisfies all constraints

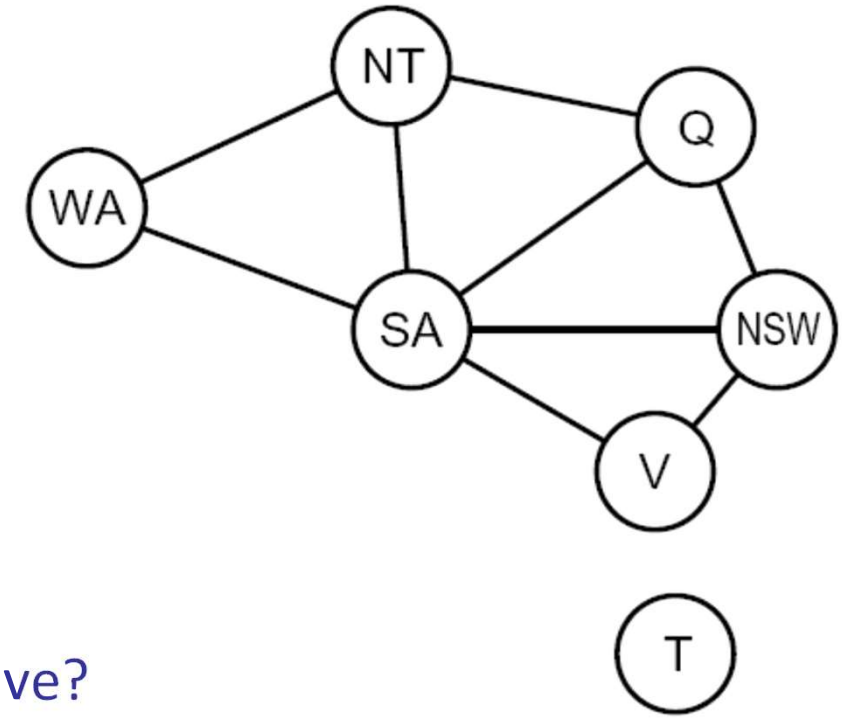


- We'll start with the straightforward, naïve approach, then improve it

---

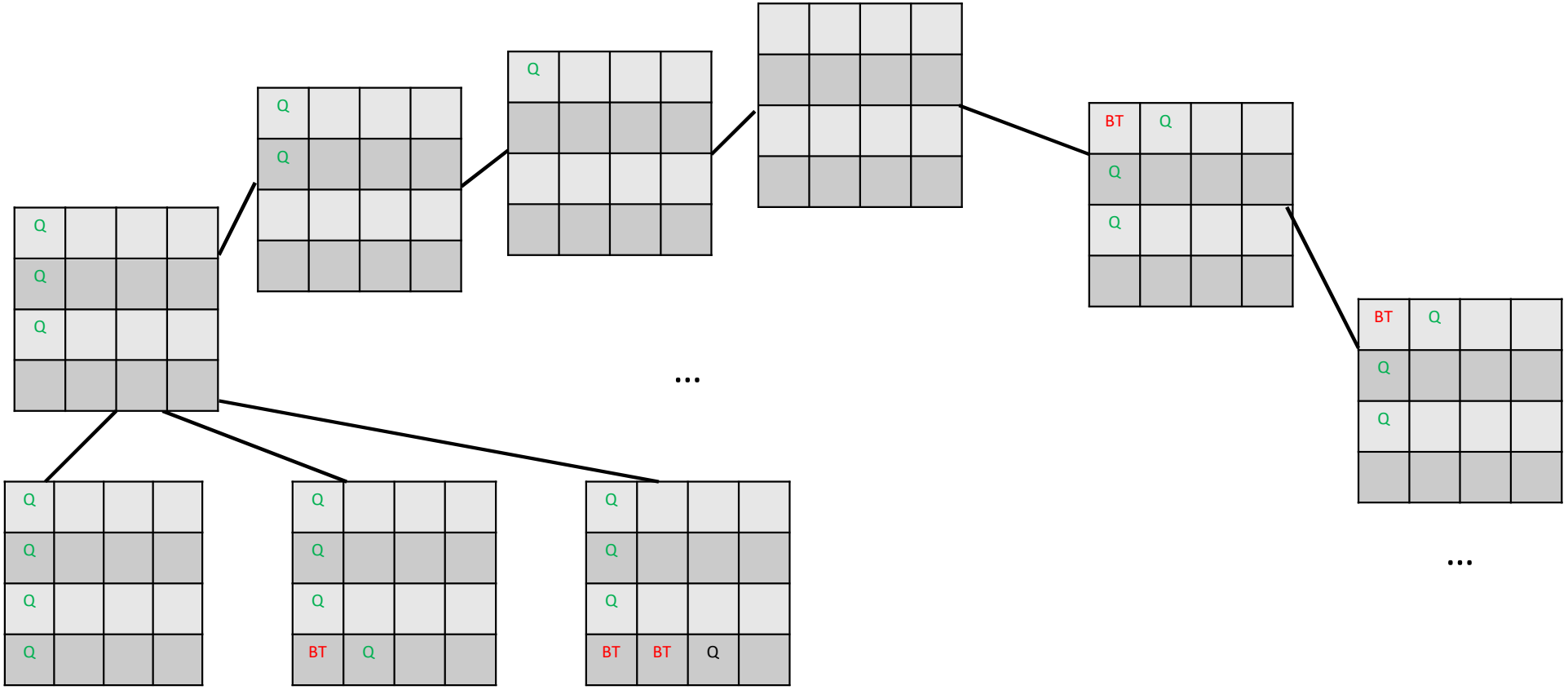
Search Methods

- What would BFS do?
- What would DFS do?
- What problems does naïve search have?



با BFS میخوایم این کارو انجام بدیم با این رویه حل میکنیم --> می خوایم روی این گراف حرکت بکنیم و تک تک این نودها رو مقدار بدیم ؟؟؟
با DFS --> در واقع کی رو انتخاب بکنیم برای رنگ امیزی و بعد که انتخاب کردیم چه رنگی بهش بدیم (هم برای DFS و هم برای BFS)

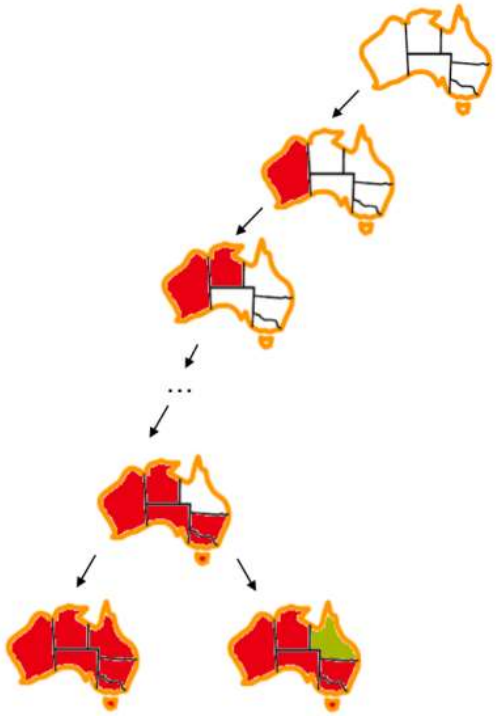
Naïve depth fist search



مسئله n وزیر:
ما میخوایم n تا وزیر رو قرار بدیم --> اول یک خونه خالی داریم از وزیرها حالا میخوایم یک وزیر رو یه جایی بذاریم و بعد بحث وزیر دوم پیش میاد که متغییر دوم میشه که متغییر دوم رو خونه اول گذاشته --> به صورت سرچ کورکورانه داره جلو می ره --> و دنبال اینه که 4 تا وزیر رو بذاره و بعد بیاد چکش بکنه و ببینه این خوب بود یا نه --> وقتی که 4 امین وزیر رو گذاشتیم دیگه وزیری نداریم که بخوایم قرار بدیم ینی دیگه متغییری نداریم که بخوایم بذاریم یا مقدار بهش بدیم --> حالا باید بیایم حالت بعدی رو چک بکنیم (این تکنیک چی هست؟ عمقی است ینی داریم عمقی می ریم جلو) --> خب ما برمیگردیم و اخرین قید رو عوض میکنیم و توی عمق حرکت میکنیم و این وزیر رو چک میکنیم و درنهایت مشخص میشه که نحوی که وزیر اول رو انتخاب کردیم توی مکان اول اصلا نمی تونه جواب ما باشه ینی هیچ موقع ما نباید وزیر اول رو بذاریم توی خونه اول پس وزیر رو می ذاریم توی خونه دوم و همین عملیات سرچ رو انجام بده تا در نهایت تموم بشه

عملیات DFS که داشتیم کل فضای حالت رو داره می گرده و خیلی کورکورانه داره برخورد میکنه

DFS



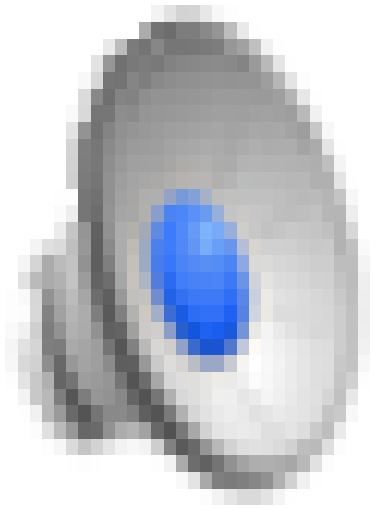
DFS برای مسئله رنگ امیزی:
اول بدون رنگ داریم --> به اولین ایالت رنگ قرمز میدیم و می ریم جلو....
بعد به اخرش می رسه و می بینه که این استیت هدف نیست پس برمیگرده عقب و اخرین رنگ امیزی رو هی تغییر میده

- In depth I branch factor (n - l)d
- Number of states $n!d^n$

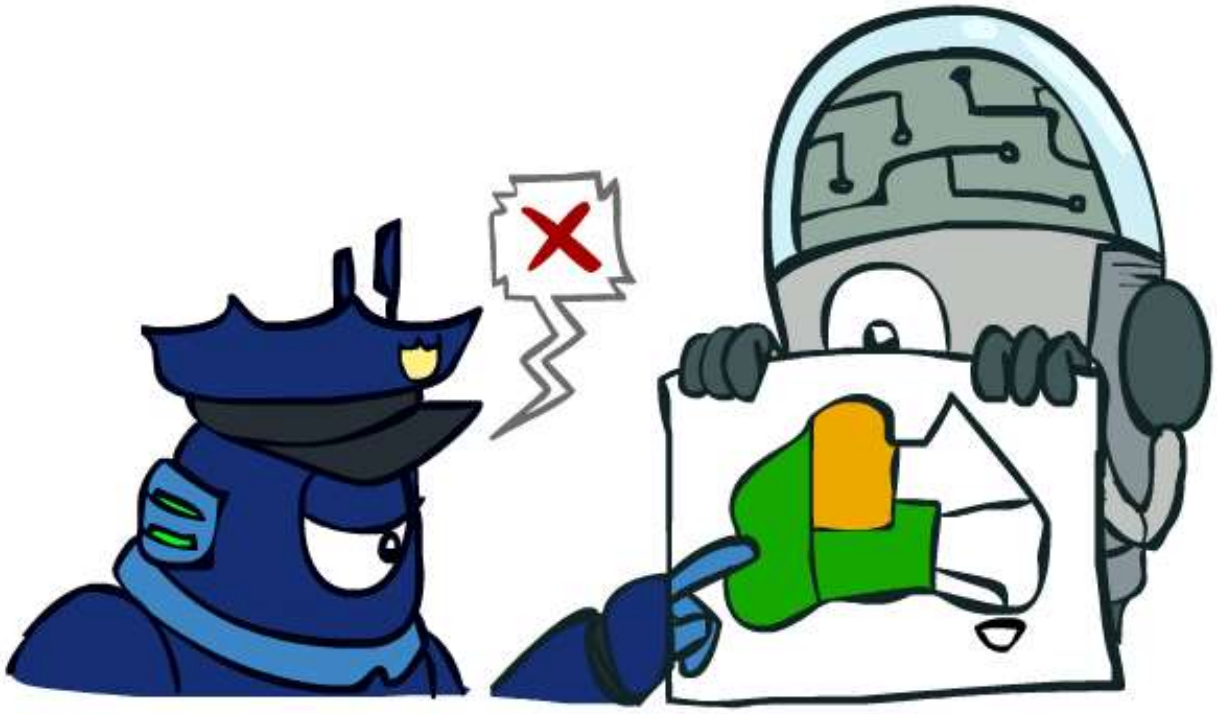
  $(nd) * [(n-1)d] * [(n-2)d] * \ldots d = n!d^n$

  8-Queens -> $8!.8^8$ -> 8^8

Video of Demo Coloring -- DFS

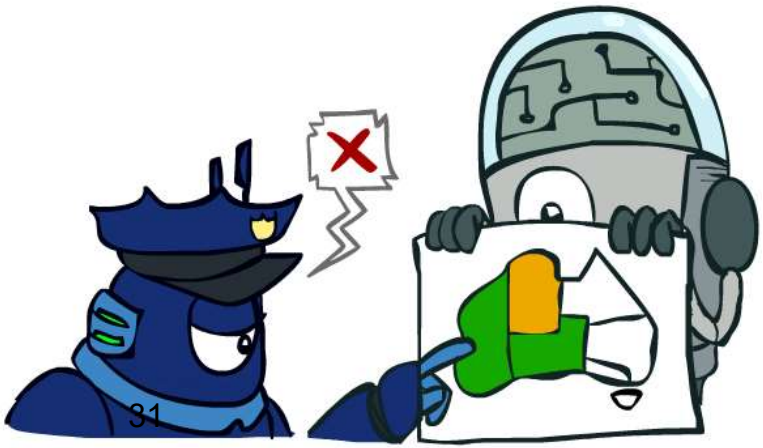


Backtracking Search



Backtracking Search

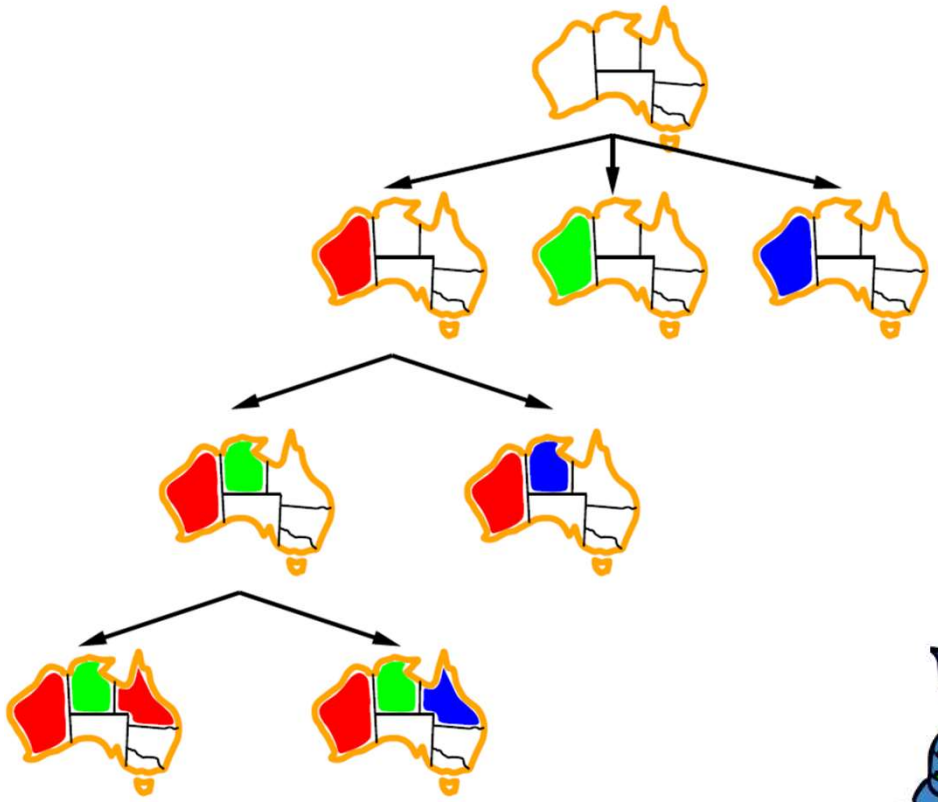
- Backtracking search is the basic uninformed algorithm for solving CSPs
- Idea 1: One variable at a time
  - Variable assignments are commutative, so fix ordering
  - I.e., [WA = red then NT = green] same as [NT = green then WA = red]
  - Only need to consider assignments to a single variable at each step
- Idea 2: Check constraints as you go
  - I.e. consider only values which do not conflict previous assignments
  - Might have to do some computation to check the constraint
  - "Incremental goal test"
- Depth-first search with these two improvements is called *backtracking search* (not the best name) *(uninform search)*
- Can solve n-queens for n ≈ 25

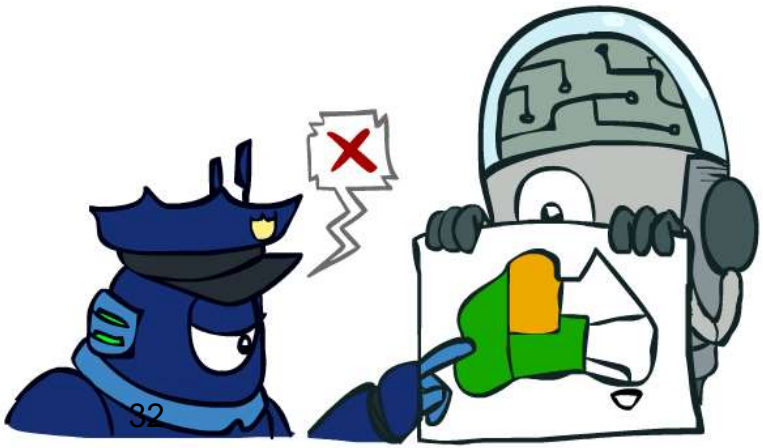


این BS جز جستجوی های uninformed مطرح میشه ینی خیلی اطلاعاتی راجع به مسئله در نظر نمی گیره و میخواد خیلی وارد شرایط مسئله نشه
ایده اش: بیا همیشه یک متغییر رو انتخاب بکن و یک مقداری رو بهش بده مثلا برای مسئله رنگ امیزی یک مقدار رنگ رو نسبت می دیم و گام دوم که مقدار رو نسبت دادیم چک میکنه که اون مقداری که نسبت دادیم برای اون متغییر با constraints ها می خونه یا نه اگر با constraints ها نخوند بهش اجازه نده توی اون مسیر ادامه پیدا بکنه --> این رویه یکسری مسیر رو برامون حذف می کنه به کمک این constraints هایی که توی مسئله داشتیم و این constraints ها رو توی گراف تعریف کردیم --> همینطوری که داریم روی گراف حرکت میکنیم هر نود جدیدی رو که مقدار دادیم لازمه که همسایه های اون نود رو چک بکنیم چون همسایه های اون نود میشن متغییرهای که یک constraints روشون تعریف شده و توی مقداردهی ما فقط باید حواسمون به اونا باشه و اگر قراره با همسایه هاش نخونه بهتره دیگه اون رو مقدار ندیم

این BS می اومد عملیات سرچ رو هوشمندتر می کرد

Backtracking Example





ابتدای کار ما سه تا نود رو داریم و فرض میکنیم ایالت اول رو می خوایم رنگ امیزی بکنیم --> DFS میخوایم کار رو جلو ببریم پس اولی رو انتخاب میکنیم و می ریم برای رنگ امیزی متغییر بعدی و برای متغییر بعدی با توجه به constraintsها دوتا حالت داریم ینی نباید قرمز باشن --> اینقدر می ره جلو و می فهمه اشتباه بوده پس دیگه ادامه نمیده اون مسیر رو

Backtracking Search

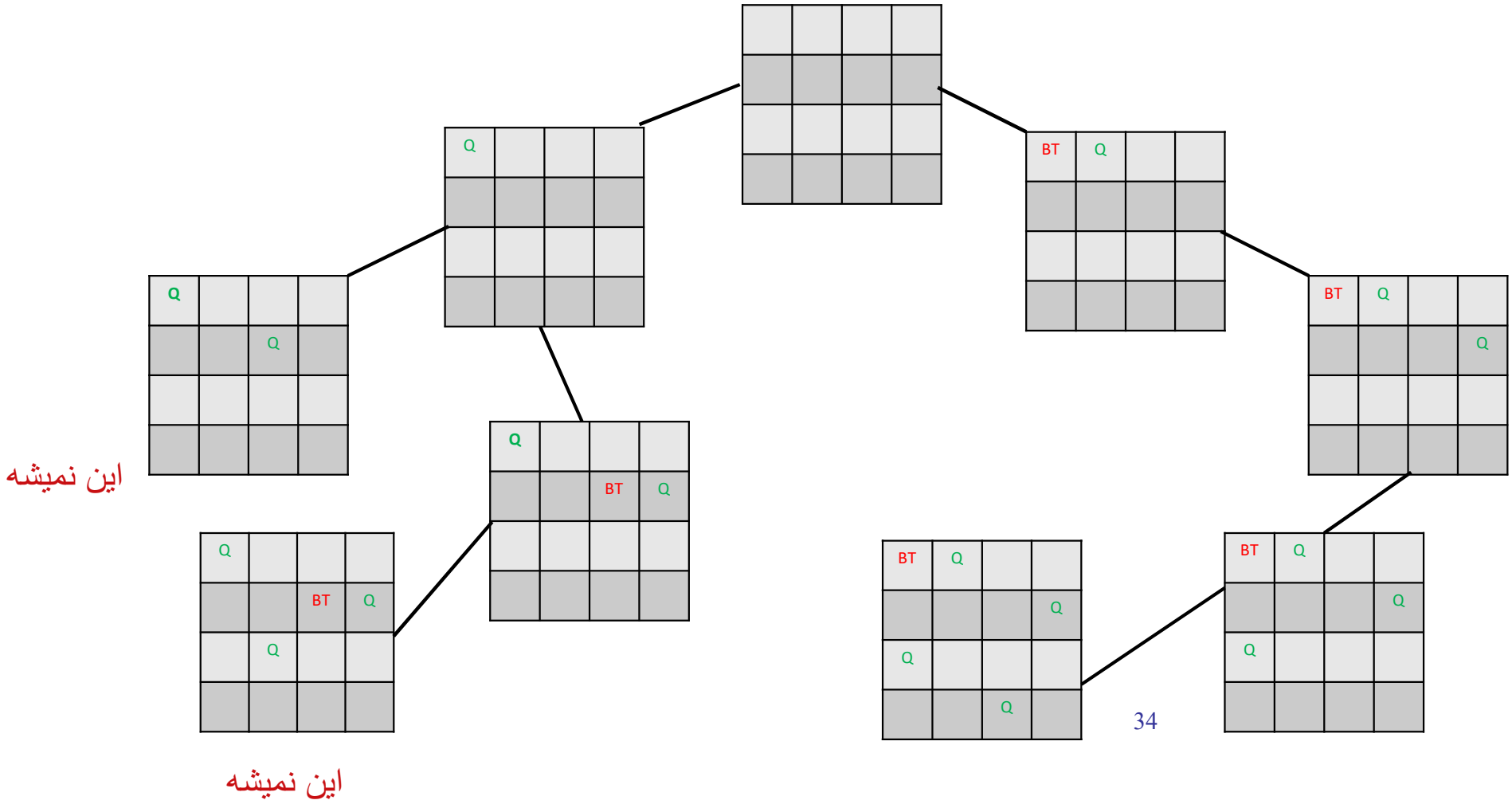
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

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
```

- Backtracking = DFS + variable-ordering + fail-on-violation
- What are the choice points?

ابتدای کار این الگوریتم فرض میکنه که هیچ متغییری وجود نداره و این constraints ها رو توی csp گذاشتیم ینی یک مجموعه قیودی داریم که داخل csp گذاشتیم و هیچ متغییری هنوز انتخاب نشده
الگوریتم اصلی توی این تابع recursive-backtracking است که اگر به شرایطی برسه که همه متغییرها انتخاب شده باشند و مقدار بهشون داده شده باشه در این حالت ما مسئله رو حل کردیم ولی اگر این شرایط نباشه ینی هنوز متغییری داریم یا .. بحث سلکت کردن داریم که الان میخوایم یک متغییر رو انتخاب بکنیم که با کمک تابع select-unassigned.. این کارو انجام میدیم ینی متغییر رو انتخاب میکنیم و بعد که متغییر انتخاب شد مسئله ای که داریم اینه که چه مقداری بهش بدیم که این یک دامنه مقادیر داره که برای دامنه مقادیر با کمک تابع order-domain... همه مقادیر رو یک تست بکنیم پس میاد همه مقادیری که می تونه اون متغییر داشته باشه for each روش می زنه و به ازای همه مقادیرش شروع میکنه به بحث سرچ کردن که سرچش هم به صورت بازگشتی و اگر اون مقداره که انتخاب شده بود سازگار بود با قیود ما می ریم اون متغییر رو با مقدارش اضافه می کنیم به لیست اساین ها و دوباره می ریم مسئله رو با یک فضای کوچیک تر دوباره فراخوانیش میکنیم به صورت بازگشتی
این کار رو روی همه متغییرها انجام میده --> اگر نتیجه جستجو توی فضای کوچیک تر به فضای خوبی رسید همون رو برمیگردونه ولی اگر نه اون متغییر رو حذف میکنه اون بحث BS همین جاست و دنبال این است که یک متغییر دیگه رو در نظر بگیره
الگوریتم Backtracking چند تا قسمت اصلی داره:
1- رویه جستجو است که به صورت DFS است
2- توی صفحه نوشته

Backtracking



مثال n وزير:

Video of Demo Coloring – Backtracking