UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

Búsqueda con Restricciones

Alan Reyes-Figueroa

Inteligencia Artificial (Aula 08) 05.febrero.2024

Búsqueda con Restricciones

Problemas de satisfacción de restricciones (CSP):

- Supuestos sobre el espacio ambiente: un agente único, acciones deterministas, estado plenamente observado, espacio de estados discreto.
- Planning: secuencias de acciones.
  - El camino hacia la meta es lo importante.
  - Los caminos tienen varios costos y profundidades.
  - Las heurísticas brindan una guía específica para el problema.
- Identificación: asignaciones a variables
  - El objetivo en sí es importante, no el camino.
  - Todos los caminos a la misma profundidad (para algunas formulaciones).
  - Los CSP están especializados en problemas de identificación.

Búsqueda con Restricciones

Problemas de búsqueda estándar:

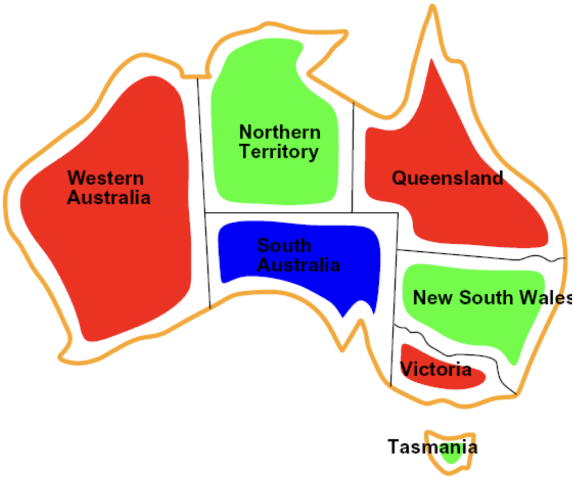
- El estado es una “caja negra”: estructura de datos arbitraria.
- La prueba de objetivos puede ser cualquier función sobre los estados.
- La función sucesora también puede ser cualquier cosa.

Problemas de satisfacción de restricciones (CSP):

- Un subconjunto especial de problemas de búsqueda.
- El estado está definido por variables $\mathbf{x}_i$ con valores de un dominio D (a veces D depende de i).
- La prueba de objetivos es un conjunto de restricciones que especifican combinaciones permitidas de valores para subconjuntos de variables.
- Permite algoritmos útiles de propósito general con más potencia que los algoritmos de búsqueda estándar.

Ejemplos

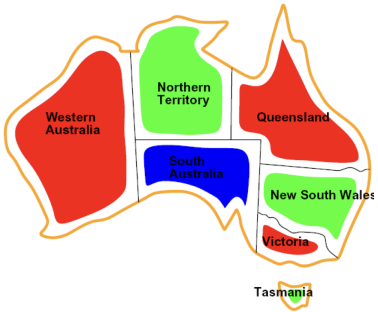
Coloración de Mapas:



Ejemplos

Coloración de Mapas:

- Variables: $\{WA, NT, Q, SA, NSW, V, T\}$
- Dominio: $\{red, green, blue\}$
- Restricciones: regiones adyacentes deben tener diferente color.
  Forma implícita: $WA \neq NT$.
  Forma explícita:
  $(WA, NT) \in \{(red, green), (red, blue), \ldots\}$

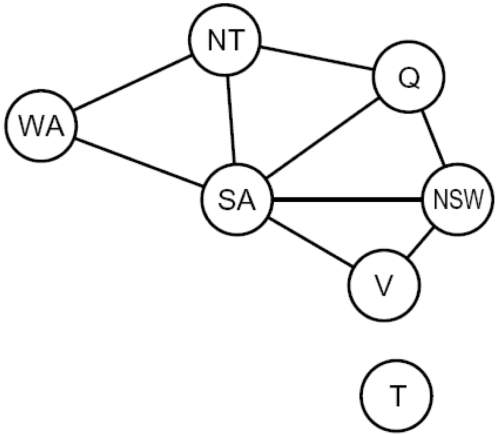


En este caso, una solución es una asignación de colores a cada variables, que además, satisface todas las restricciones. Por ejemplo:

$$WA = red, NT = green, Q = red, SA = blue, NSW = green, V = red, T = green.$$

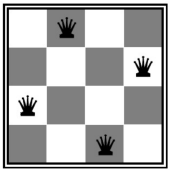
Ejemplos

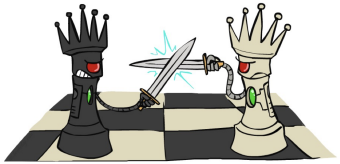
Usualmente en un CSP, las restricciones se modelan mediante grafos de restricciones:



Ejemplos

***N*-reinas**:





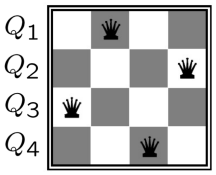
Formulación 1:

- Variables: $\mathbf{x}_{ij}$, para $1 \leq i, j \leq N$.
- Dominio: $\mathbf{x}_{ij} \in \{0, 1\}$.
- Restricciones: Para todo $1 \leq i, j, k \leq N$:

$$
\begin{aligned}
(\mathbf{x}_{ij}, \mathbf{x}_{ik}) &\in \{(0,0), (0,1), (1,0)\}, \\
(\mathbf{x}_{ij}, \mathbf{x}_{kj}) &\in \{(0,0), (0,1), (1,0)\}, \\
(\mathbf{x}_{ij}, \mathbf{x}_{i+k,j+k}) &\in \{(0,0), (0,1), (1,0)\}, \\
(\mathbf{x}_{ij}, \mathbf{x}_{i+k,j-k}) &\in \{(0,0), (0,1), (1,0)\}.
\end{aligned}
$$

Ejercicio

Recordemos que el problema de las N-reinas también se puede formular en términos de permutaciones:

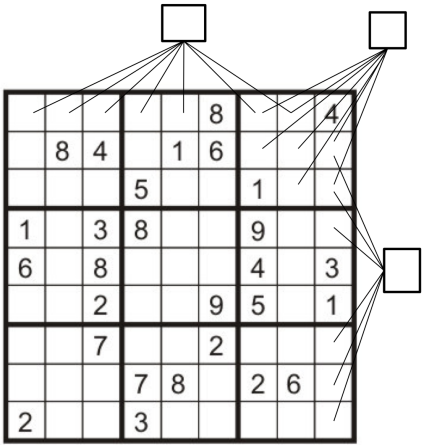


Formulación 2:

- Variables: $\mathbf{p} = (Q_1, Q_2, Q_3, \ldots, Q_N)$ es una permutación de $(1, 2, 3, \ldots, N)$.
- Dominio: ¿?
- Restricciones: ¿?

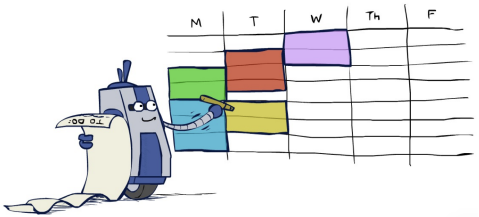
Ejercicio

Sudoku:



Ejemplos

Scheduling y Asignación de Tareas:

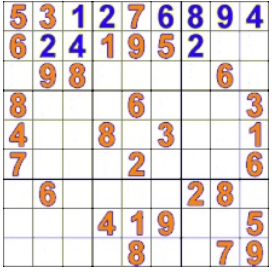


Ejemplos de problemas de asignación de tareas:

- Calendarización de tareas
- Asignación de horarios
- Configuración de hardware
- Problemas de transporte o Programación de Fábrica
- Diseño de circuitos
- . . .

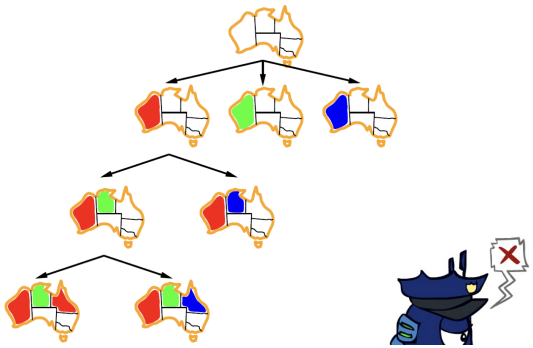
Estrategias de Búsqueda CSP

Backtracking: El **retroceso** o *backtracking* es un algoritmo general para encontrar soluciones a algunos problemas computacionales, en particular problemas de satisfacción de restricciones, que construye gradualmente candidatos a las soluciones y abandona a un candidato ("retrocesos") tan pronto como determina que el candidato no puede ser completado a un valor válido. solución.



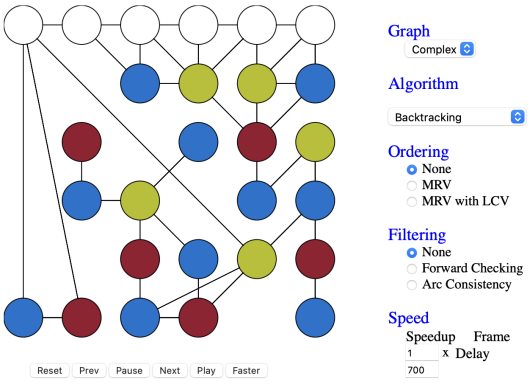
Ejemplo de uso de *backtracking*.

Backtracking



Ejemplo de uso de *backtracking*.

Backtracking



Otro ejemplo de uso de *backtracking*.

Backtracking

La búsqueda de *backtraking* es el algoritmo básico desinformado para resolver problemas CSP.

- **Idea 1:** una variable a la vez.
  - Las asignaciones de variables son conmutativas, así que corregir el orden $\Rightarrow$ ¡mejor factor de ramificación!
  - Es decir, $[WA = red, then\ NT = green]$ es igual que $[NT = green, then\ WA = red]$.
  - Solo es necesario considerar asignaciones a una sola variable en cada paso.

- **Idea 2:** Verificar las restricciones sobre la marcha.
  - Es decir, considerar solo valores que no entren en conflicto con asignaciones anteriores o Es posible que tenga que hacer algunos cálculos para verificar las restricciones.
  - "Prueba de objetivos incrementales".

  La búsqueda en profundidad con estas dos mejoras se llama *backtracking search*.

Backtracking

```
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
```

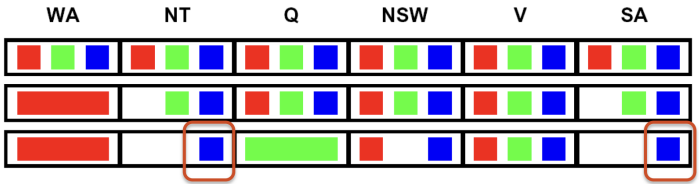
Backtracking

Hay varias formas de mejorar el *backtracking*.

- **Reordenar** (*Ordering*): cambiar el orden de las variables.
  - ¿Cuál variable debo asignar primero? ¿Cuál después?
  - ¿En qué orden se deben probar los valores?

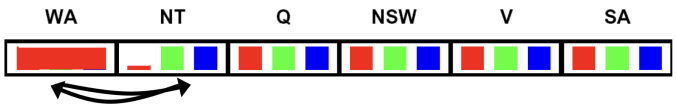
- **Filtrar** (*Filtering*): Detectar las fallas antes de que ocurran.
  - Llevar registro de los dominios para las variables aún no asignadas.
  - Cruzar información y remover malas opciones.
  - *Forward checking*: Tachar valores que incumplen una restricción cuando se agregan a la asignación existente.

Filtering

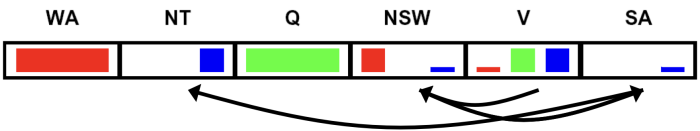


Consistencia de Arcos

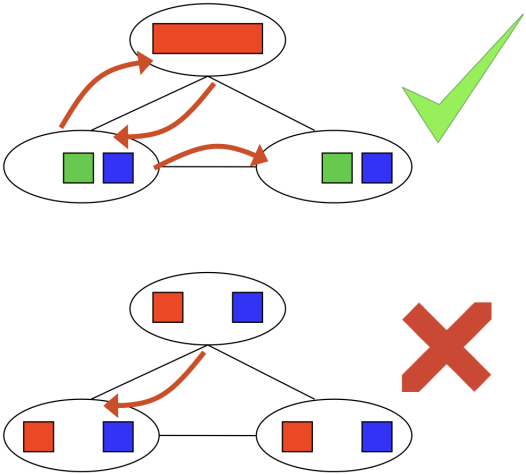
Un arco o arista $X \rightarrow Y$ es **consistente** si, y sólo si, por cada **x** en la cola, existe un elemento **y** en el head que puede asignarse sin incumplir las restricciones.



Una forma simple de mejorar el *backtracking* es propagar la búsqueda asegurándose que todas las aristas son consistentes.



Consistencia de Arcos



Consistencia de Arcos

Importante:

- Si X pierde un valor, es necesario volver a verificar los vecinos de X.
- La consistencia del arco detecta fallas antes de la verificación directa.
- Puede ejecutarse como preprocesador o después de cada asignación.
- ¿Cuál es la desventaja de imponer la coherencia del arco?
  (Detectar todos los posibles problemas adelante es un problema *NP-hard*).

Consistencia de Arcos

function AC-3(csp) **returns** the CSP, possibly with reduced domains
  inputs: csp, a binary CSP with variables $\{X_1, X_2, \ldots, X_n\}$
  local variables: $queue$, a queue of arcs, initially all the arcs in csp

  while $queue$ is not empty **do**
    $(X_i, X_j) \leftarrow$ REMOVE-FIRST($queue$)
    if REMOVE-INCONSISTENT-VALUES(X_i, X_j) **then**
      for each X_k **in** NEIGHBORS$[X_i]$ **do**
        add (X_k, X_i) to $queue$

---

function REMOVE-INCONSISTENT-VALUES(X_i, X_j) **returns** true iff succeeds
  $removed \leftarrow false$
  for each x **in** DOMAIN$[X_i]$ **do**
    if no value y in DOMAIN$[X_j]$ allows (x,y) to satisfy the constraint $X_i \leftrightarrow X_j$
      then delete x from DOMAIN$[X_i]$; $removed \leftarrow true$
  return $removed$