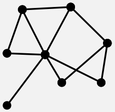


# Constraint Satisfaction Problems

# Formalization and Algorithms

Introduction to Artificial Intelligence

Chandra Gummaluru
University of Toronto

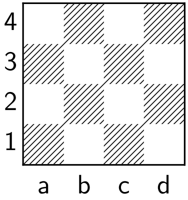
Version W22.1

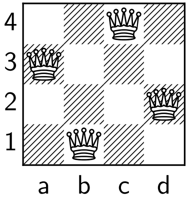
- The following is based on material developed by many individuals, including (but not limited to):
  - Sheila McIlraith
  - Fahiem Bacchus
  - Sonya Allin
  - Craig Boutilier
  - Hojjat Ghaderi
  - Rich Zemel
  - Elliot Creager

- So far, our search problems involved finding a path to a goal state.
- However, in many problems, we only care about finding the goal state itself, i.e., we do not care about the path.
- Such problems are called **constraint satisfaction problems** (CSPs).

**Example:** *N*-Queens Puzzle

- Place $N$ queens on an $N \times N$ board with $N$ so that none of the queens attack each other.
- The search is over the set of all board configurations.
- The element we seek is the specific configuration in which none of the queens attack each other.
- Finding such board configurations is non-trivial.
- However, given a board configuration, it is easy to check that it is a valid solution.



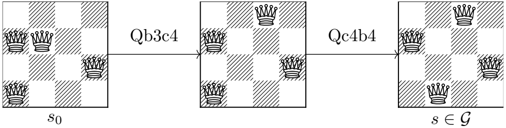



- A CSP is a type of search problem. Thus, we can use the same process to solve it:

  1. Check if the current state, $s$, is the goal.
  2. If not, perform an action, $a \in A(s)$, resulting in a new state, $s' = a(s)$.
  3. Set the current state to $s'$ and repeat until a goal is found.

  **Example:** Searching in $N$-Queens

  - In the $N$-queens puzzle, we define $\mathcal{S}$ as the set of all possible board configurations, and $\mathcal{G}$ as the subset in which no two queens attack each other.
  - We can search for a goal by starting with an arbitrary placement of the queens, and move them one at a time to achieve the desired configuration.




- However, as we will see, this is inefficient.

- We formally define a CSP as a search problem in which:
  - states can be defined using a fixed set of variables, $\mathcal{V}$, where:
    - each state, $s \in \mathcal{S}$, is represented by assigning each variable, $V \in \mathcal{V}$, a unique value, $v \in \mathrm{dom}\, V$, which we represent using a set $\{V = v, V \in \mathcal{V}\}$.
    - the set of all possible assignments is

$$\mathrm{dom}\mathcal{V} := \prod_{V \in \mathcal{V}} \mathrm{dom}\, V.$$

    - every state must be represented as an assignment of the variables, but not every assignment needs to represent a state, i.e., $\mathcal{S}$ is a subset of $\mathrm{dom}\mathcal{V}$.
  - goals can be defined using a fixed set of constraints, $\mathcal{C}$, where:
    - each constraint, $C \in \mathcal{C}$, couples a fixed set of variables, $\mathrm{scp}(C) \subseteq \mathcal{V}$, called its **scope**.
    - given a partial assignment, $\{V = v \text{ s.t. } v \in \mathrm{dom}(V), V \in \mathrm{scp}(C)\}$, the constraint, $C$ returns either `true` or `false`, indicating whether it was satisfied or not.

- **Example:** The $N$-Queens Puzzle as a CSP with $N^2$ Binary Variables
  - Each square may or may not have a queen so let $V_{c,r} \in \{0, 1\}$ denote whether there is a queen at the square in the $c^{\text{th}}$ column and $r^{\text{th}}$ row, where $c, r \in \{1, \ldots, N\}$.
  - The constraints are:
    - **row constraints**: $V_{r,c} = 1 \Rightarrow V_{r',c} = 0, \forall r' \neq r$
    - **column constraints**: $V_{r,c} = 1 \Rightarrow V_{r,c'} = 0, \forall c' \neq c$
    - **diagonal constraints**: $V_{r,c} = 1 \Rightarrow V_{r+i,c+\alpha i} = 0, \forall \alpha \in \{-1, 1\}, i$

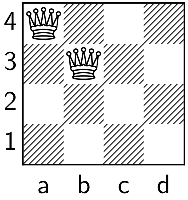
- **Example**: The $N$-Queens Puzzle as a CSP with $N$ $N$-ary variables
  - Each row must have exactly one queen so let $V_r \in \{1, \ldots, N\}$ denote the column in which the queen on the $r^{\text{th}}$ row is located.
  - One way to specify the constraints is as follows:
    - **row/column constraints**: $V_i \neq V_j, \forall i \neq j$
    - **diagonal constraints**: $|V_i - V_j| \neq |i - j|, \forall i \neq j$
  - Another way to specify the constraints is as follows:
    - **all constraints**: $|V_i - V_j| \neq |i - j| \neq 0, \forall i \neq j$

- We can now express the CSP in terms of a GSP.
    - The initial state is some arbitrary assignment, $\{V = v^{(0)}, V \in \mathcal{V}\}$.
    - The feasible actions involve modifying the value of any variable.
    - The goal test function checks that the assignment satisfies all the constraints.
- As stated before, this is inefficient. There are two main reasons why.

1. We must assign all variables simultaneously before checking the constraints, but constraints are often violated much earlier.

**Example:** Invalid Partial Assignment in $N$-Queens

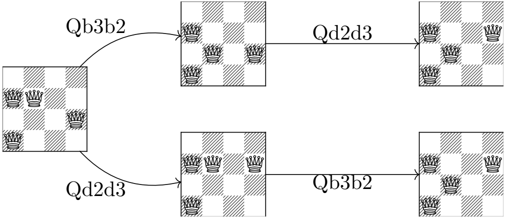
- Suppose we place queens on a4 and b3. Then, no matter where we place the remaining queens, the resulting board configuration will be invalid.
- Still, because we must assign all variables simultaneously, we need to check all 16 board configurations that include queens on a4 and b3.



2. Different action sequences can yield the same state, but since we do not care about the paths, this results in searching more nodes than necessary.

**Example:** Redundant Paths in $N$-Queens

- In the tree below, the order in which the moves Qb3b2 and Qd2d3 are played is irrelevant, as both result in the same board configuration.

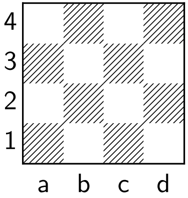


- To address these issues, we can use the notion of a partial assignment.
- Given a partial assignment, we check the **bound** constraints, i.e., those whose scopes are fully assigned (non-bound constraints will not be violated).
- **Procedure**: Backtracking Search
  1. Start with the empty assignment, $\{\}$.
  2. If all variables are assigned, a solution has been found. Otherwise, pick any unassigned variable, $V \in \mathcal{V}$.
  3. For each value, $v \in \operatorname{dom} V$: If every bound constraint is satisfied, continue searching recursively. If every value in $\operatorname{dom} V$ has been exhausted and none satisfied the bound constraints, backtrack.

```
procedure Search()
    if Assigned(V_1, ..., V_|𝒱|) then                  ▷ all variables are assigned
        𝒢.Append(([V_1], ..., [V_|𝒱|]))
    else
        V ← SelectUnassigned(𝒱)                       ▷ choose an unassigned variable
        for v ∈ dom V do
            Assign(v, V)
            γ ← false                                 ▷ flag for if constraints are violated
            for C ∈ 𝒞 : Bound(C) do                   ▷ for each bound constraint
                if Violated(C) then
                    γ ← true                          ▷ the constraint is violated
            if γ = false then
                Search()                              ▷ search extensions
        Unassign(V)
```

- In this example, we perform backtracking search on the 4-Queens puzzle.

| Itr. | $V_1$ | $V_2$ | $V_3$ | $V_4$ |
|---|---|---|---|---|
| 0 | | | | |
| 1 | a | | | |
| 2 | a | a | | |
| 3 | a | b | | |
| 4 | a | c | | |
| 5 | a | c | a | |
| 6 | a | c | b | |
| 7 | a | c | c | |
| 8 | a | c | d | |
| 9 | a | d | | |
| 10 | a | d | a | |
| 11 | a | d | b | |
| 12 | a | d | b | a |
| 13 | a | d | b | b |
| 14 | a | d | b | c |
| 15 | a | d | b | d |
| 16 | a | d | c | |
| 17 | a | d | d | |
| 18 | b | | | |
| 19 | b | a | | |
| 20 | b | b | | |
| 21 | b | c | | |
| 22 | b | d | | |
| 23 | b | d | a | |
| 24 | b | d | a | a |
| 25 | b | d | a | b |
| 26 | b | d | a | c |



- The previous example seems to suggest that backtracking is still inefficient.
    - After assigning $V_1 = \text{a}$, we could remove:
        - 'a' and 'b' from dom $V_2$
        - 'a' and 'c' from dom $V_3$
        - 'a' and 'd' from dom $V_4$
    - This is because the aforementioned choices would violate at least one constraint.
- Ideally, we would check the constraints for possible violations *before* fully assigning their scopes.

- One idea is to look ahead at any constraint with exactly one unassigned variable (so-called **almost-bound** constraints).
- We proceed in the same way as plain back-tracking search but whenever we assign a value, $v$ to a variable, $V$, we do the following:
  1. For each almost-bound constraint, $C$, such that $V \in \text{scp}(C)$, let $V'$ denote the un-assigned variable.
  2. For each $v' \in \text{dom}\, V'$, check whether augmenting the current partial assignment with $\{V' = v'\}$ will violate $C$ and if so, remove $v'$ from dom $V'$.
- When we backtrack, we must restore any pruned values.
- This is called **forward-checking**; it essentially checks each constraint whose scope has exactly one unassigned variable and prunes its domain of values that will cause the constraint to be violated.

# Backtracking Search with Forward Checking: Pseudo-code

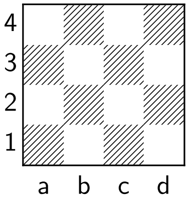
```
procedure SearchWithFC()
    if Assigned(V_1, ..., V_{|𝒱|}) then                          ▷ all variables are assigned
        𝒢.Append(([V_1], ..., [V_{|𝒱|}]))
    else
        V ← SelectUnassigned(𝒱)                                  ▷ choose an unassigned variable
        for v ∈ dom V do
            Assign(v, V)
            γ ← false                                            ▷ flag for if constraints are violated
            for C ∈ 𝒞 : AlmostBound(C) do                        ▷ for each almost bound constraint
                if PruneWithFC(C) = DWO then
                    γ ← true                                     ▷ a domain wipe-out occurred
            if γ = false then
                SearchWithFC()                                   ▷ search extensions
            RestorePruned()                                      ▷ restore pruned domains
        Unassign(V)
```

# Backtracking Search with Forward Checking: Pseudo-code

**procedure** PruneWithFC()<br>
    $V \leftarrow$ FindUnassigned($\mathcal{V}_C$) ▷ find an unassigned variable in $C$'s scope<br>
    **for** $v \in \text{dom } V$ **do**<br>
        Assign($v, V$)<br>
        **if** Violated($C$) **then**<br>
            Remove($v, \text{dom } V$)<br>
            **if** $\text{dom } V = \emptyset$ **then** ▷ domain wipe-out occurred<br>
                **return** `DWO`<br>
    **return** `true`

- In this example, we perform backtracking search with forward-checking on the 4-Queens puzzle, assigning variables in the order $V_1, V_2, V_3, V_4$.

| Itr. | $V_1$ | $V_2$ | $V_3$ | $V_4$ | dom $V_1$ | dom $V_2$ | dom $V_3$ | dom $V_4$ |
|---|---|---|---|---|---|---|---|---|
| 0 | | | | | $\{a, b, c, d\}$ | $\{a, b, c, d\}$ | $\{a, b, c, d\}$ | $\{a, b, c, d\}$ |
| 1 | a | | | | | $\{c, d\}$ | $\{b, d\}$ | $\{b, c\}$ |
| 2 | a | c | | | | | $\varnothing$ | |
| 3 | a | d | | | | | $\uparrow^1 \{b\}$ | $\uparrow^1 \{c\}$ |
| 4 | a | d | b | | | | | $\varnothing$ |
| 5 | b | | | | | $\uparrow^0 \{d\}$ | $\uparrow^0 \{a, c\}$ | $\uparrow^0 \{a, c, d\}$ |
| 6 | b | d | | | | | $\{a\}$ | $\{a, c\}$ |
| 7 | b | d | a | | | | | $\{c\}$ |
| 8 | b | d | a | c | | | | |




- The notation $\uparrow^i$ means that before the domain was pruned, it was first restored to whatever it was in iteration $i$ (which happens after a domain wipe-out).

- If a constraint has multiple unassigned variables, the only way we can prune a value for a particular unassigned variable is if the constraint is violated for *every* assignment of the other unassigned variables in the constraint's scope.
- Suppose we assign a variable, $V$, some value $v$.
- We want to make sure that each constraint, $C$, whose scope, $\text{scp}(C)$, contains $V$, can still be satisfied by assigning some value to each $V' \neq V \in \text{scp}(C)$.
- If this is the case, then we say that the partial assignment $\{V = v\}$ is **supported**. Otherwise, the partial assignment is **unsupported**.

- We proceed in the same way as plain back-tracking search, but whenever a value, $v$,is assigned to a variable, $V$, we do the following:
    1. Assume dom $V = \{v\}$.
    2. Then, check each constraint such that $V \in \text{scp}(C)$, prune the domain of each variable in $\text{scp}(V)$ of any unsupported values.
- When we backtrack, we must restore any pruned values.
- This is called **generalized-arc-consistency**.

# Backtracking Search with Generalized Arc Consistency: Pseudo-code

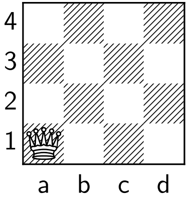
```
procedure SearchWithGAC()
    if Assigned(V_1, ..., V_|𝒱|) then                        ▷ all variables are assigned
        𝒢.Append(([V_1], ..., [V_|𝒱|]))
    else
        V ← SelectUnassigned(𝒱)                              ▷ choose an unassigned variable
        for v ∈ dom V do
            Assign(v, V)
            𝒬 = ∅
            for C ∈ 𝒞 : V ∈ 𝒱_C do                           ▷ for each constraint whose scope contains V
                𝒬.Append(C)
            if not PruneWithGAC(𝒬) = DWO then               ▷ no domain wipe-out occurred
                SearchWithGAC()
            RestorePruned()                                  ▷ restore pruned domains
        Unassign(V)
```

```
procedure PruneWithGAC(Q)
    while Q ≠ ∅ do
        C ← Q.Next()
        for V ∈ V_C do
            for v ∈ dom V do
                if not Satisfiable(C, V, v) then
                    Remove(v, dom V)
                    if dom V = ∅ then
                        return DWO
                    else
                        for C ∈ 𝒞 do
                            if V ∈ V_C  and  C ∉ Q then
                                Q.Append(C)
```

- In this example, we perform backtracking search while enforcing arc consistency on the 4-Queens puzzle. We use $C_i^j : |V_i - V_j| \neq |i - j| \neq 0, \forall i \neq j$.

Itr. 1:

| dom $V_1$ | dom $V_2$ | dom $V_3$ | dom $V_4$ | $\mathcal{Q}$ |
|---|---|---|---|---|
| $\{a,b,c,d\}$ | $\{a,b,c,d\}$ | $\{a,b,c,d\}$ | $\{a,b,c,d\}$ | $\{\}$ |
| $\{a\}$ | $\{a,b,c,d\}$ | $\{a,b,c,d\}$ | $\{a,b,c,d\}$ | $\{C_1^2, C_1^3, C_1^4\}$ |
| $\{a\}$ | $\{c,d\}$ | $\{a,b,c,d\}$ | $\{a,b,c,d\}$ | $\{C_1^3, C_1^4, C_2^3, C_2^4\}$ |
| $\{a\}$ | $\{c,d\}$ | $\{b,d\}$ | $\{a,b,c,d\}$ | $\{C_1^4 C_2^3, C_2^4, C_3^4\}$ |
| $\{a\}$ | $\{c,d\}$ | $\{b,d\}$ | $\{b,c\}$ | $\{C_2^3, C_2^4, C_3^4\}$ |
| $\{a\}$ | $\{d\}$ | $\{b\}$ | $\{b,c\}$ | $\{C_2^4, C_3^4, C_2^1, C_2^4, C_3^1\}$ |
| $\{a\}$ | $\{d\}$ | $\{b\}$ | $\{c\}$ | $\{C_3^4, C_2^1, C_2^4, C_3^1, C_4^1\}$ |
| $\{a\}$ | $\{d\}$ | $\emptyset$ | $\{c\}$ | $\{C_2^1, C_2^4, C_3^1, C_4^1\}$ |

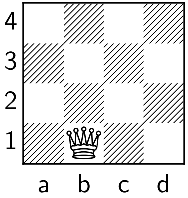


# Backtracking Search with Generalized Arc Consistency: Example

- In this example, we perform backtracking search while enforcing arc consistency on the 4-Queens puzzle. We use $C_i^j : |V_i - V_j| \neq |i - j| \neq 0, \forall i \neq j$.

Itr. 2:

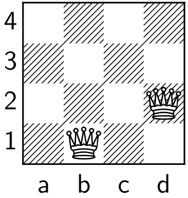
| dom $V_1$ | dom $V_2$ | dom $V_3$ | dom $V_4$ | $\mathcal{Q}$ |
|---|---|---|---|---|
| $\{b\}$ | $\{a, b, c, d\}$ | $\{a, b, c, d\}$ | $\{a, b, c, d\}$ | $\{C_1^2, C_1^3, C_1^4\}$ |
| $\{b\}$ | $\{d\}$ | $\{a, b, c, d\}$ | $\{a, b, c, d\}$ | $\{C_1^3, C_1^4, C_2^3, C_2^4\}$ |
| $\{b\}$ | $\{d\}$ | $\{a, c\}$ | $\{a, b, c, d\}$ | $\{C_1^4, C_2^3, C_2^4, C_3^4\}$ |
| $\{b\}$ | $\{d\}$ | $\{a, c\}$ | $\{a, c, d\}$ | $\{C_2^3, C_2^4, C_3^4\}$ |
| $\{b\}$ | $\{d\}$ | $\{a\}$ | $\{a, c, d\}$ | $\{C_2^4, C_3^4, C_2^1\}$ |
| $\{b\}$ | $\{d\}$ | $\{a\}$ | $\{a, c\}$ | $\{C_3^4, C_2^1, C_4^1\}$ |
| $\{b\}$ | $\{d\}$ | $\{a\}$ | $\{c\}$ | $\{C_2^1, C_4^1, C_4^2\}$ |
| $\{b\}$ | $\{d\}$ | $\{a\}$ | $\{c\}$ | $\{C_4^1, C_4^2\}$ |
| $\{b\}$ | $\{d\}$ | $\{a\}$ | $\{c\}$ | $\{C_4^2\}$ |
| $\{b\}$ | $\{d\}$ | $\{a\}$ | $\{c\}$ | $\{\}$ |



- In this example, we perform backtracking search while enforcing arc consistency on the 4-Queens puzzle. We use $C_i^j : |V_i - V_j| \neq |i - j| \neq 0, \forall i \neq j$.

Itr. 3:

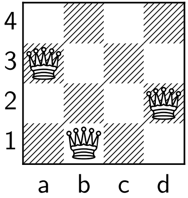
| dom $V_1$ | dom $V_2$ | dom $V_3$ | dom $V_4$ | $\mathcal{Q}$ |
|---|---|---|---|---|
| $\{b\}$ | $\{d\}$ | $\{a\}$ | $\{c\}$ | $\{\}$ |
| $\{b\}$ | $\{d\}$ | $\{a\}$ | $\{c\}$ | $\left\{C_2^1, C_2^3, C_2^4\right\}$ |
| $\{b\}$ | $\{d\}$ | $\{a\}$ | $\{c\}$ | $\left\{C_2^3, C_2^4\right\}$ |
| $\{b\}$ | $\{d\}$ | $\{a\}$ | $\{c\}$ | $\left\{C_2^4\right\}$ |
| $\{b\}$ | $\{d\}$ | $\{a\}$ | $\{c\}$ | $\{\}$ |



- In this example, we perform backtracking search while enforcing arc consistency on the 4-Queens puzzle. We use $C_i^j : |V_i - V_j| \neq |i - j| \neq 0, \forall i \neq j$.

Itr. 4:

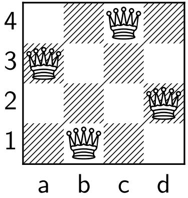
| dom $V_1$ | dom $V_2$ | dom $V_3$ | dom $V_4$ | $\mathcal{Q}$ |
|---|---|---|---|---|
| $\{b\}$ | $\{d\}$ | $\{a\}$ | $\{c\}$ | $\{\}$ |
| $\{b\}$ | $\{d\}$ | $\{a\}$ | $\{c\}$ | $\{C_3^1, C_3^2, C_3^4\}$ |
| $\{b\}$ | $\{d\}$ | $\{a\}$ | $\{c\}$ | $\{C_3^2, C_3^4\}$ |
| $\{b\}$ | $\{d\}$ | $\{a\}$ | $\{c\}$ | $\{C_3^4\}$ |
| $\{b\}$ | $\{d\}$ | $\{a\}$ | $\{c\}$ | $\{\}$ |



- In this example, we perform backtracking search while enforcing arc consistency on the 4-Queens puzzle. We use $C_i^j : |V_i - V_j| \neq |i - j| \neq 0, \forall i \neq j$.

Itr. 5:

| dom $V_1$ | dom $V_2$ | dom $V_3$ | dom $V_4$ | $\mathcal{Q}$ |
|---|---|---|---|---|
| $\{b\}$ | $\{d\}$ | $\{a\}$ | $\{c\}$ | $\{\}$ |
| $\{b\}$ | $\{d\}$ | $\{a\}$ | $\{c\}$ | $\{C_4^1, C_4^2, C_4^3\}$ |
| $\{b\}$ | $\{d\}$ | $\{a\}$ | $\{c\}$ | $\{C_4^2, C_4^3\}$ |
| $\{b\}$ | $\{d\}$ | $\{a\}$ | $\{c\}$ | $\{C_4^3\}$ |
| $\{b\}$ | $\{d\}$ | $\{a\}$ | $\{c\}$ | $\{\}$ |



- In the 4-Queens example, we observed the following:
  - Backtracking search takes many iterations.
  - Using forward checking reduced the number of iterations, but each one takes longer.
  - Enforcing arc consistency reduces the number of iterations even further, but each one takes even longer still.
- This is true in general.