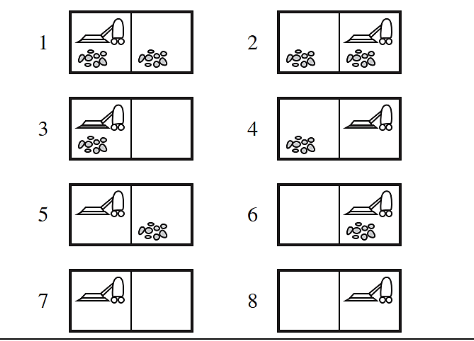
**(W8.1) SEARCHING IN NONDETERMINISTIC ENVIRONMENTS**

1. **Tìm kiếm trong môi trường không xác định**

* Một state thực hiện 1 action trả về nhiều state mới:
* Results (State, action) 🡪 States{s1, s2, s3,…}
* Ví dụ: Robot thực hiện hút bụi :



+ Robot sau khi thực hiện hút bụi, Robot có thể lựa chọn nhiều action, có thể di chuyển sang phải, di chuyển sang trái hay đứng yên tại chỗ.

1. **AND-OR search**
2. **Ý tưởng:**

* Trong 1 AND-OR search trees

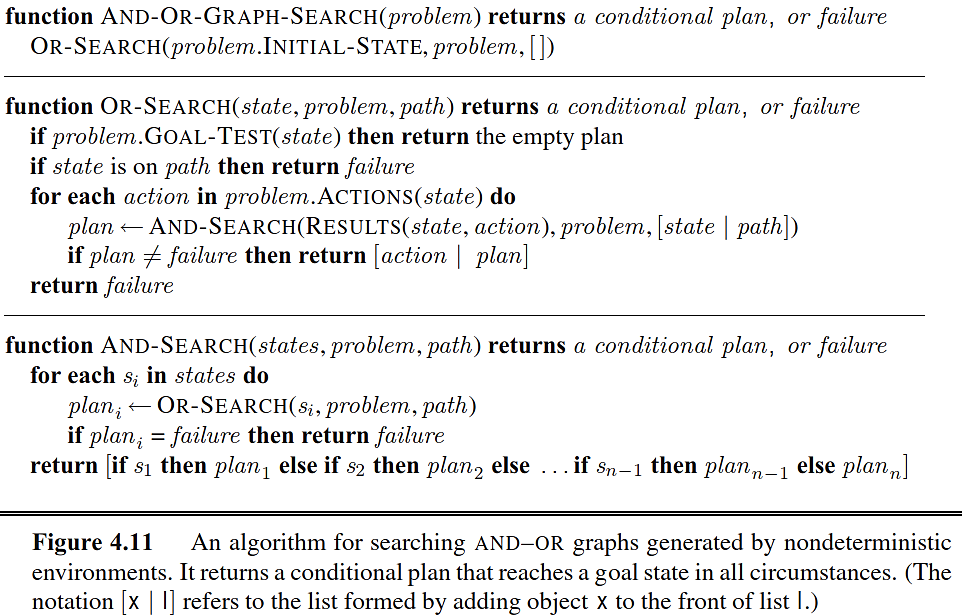
+ OR nodes là các states

+ AND nodes là các linkes

* Từ một state hay action có thể thực hiện các hoạt động 🡪 Chọn 1 hành động cụ thể để tiến hành thực hiện
* Thực hiện một action sẽ cho ra 1 state mới (And) 🡪 các states mới được tạo ra
* Solution là một cây con với tất cả các nút lá là một goal để đảm bảo các action đều dẫn đến đích (ở AND nodes chúng ta phải lấy hết các trường hợp có thể xảy ra)

1. **Đầu vào- đầu ra:**

* Đầu vào : một problem cần giải quyết
* Đầu ra: một solution hoặc có thể trả ra failure (nói cách khác là không tìm đc ra subtree)

1. **Thuật toán:**

* Xử lý:

+ Khi đang ở 1 state, ở OR-nodes sẽ thực hiện OR-search, sau đó thực hiện AND-search ở các AND-nodes phía sau, ở các AND-nodes tiếp tục thực hiện OR-search trên các OR-nodes ở phía sau nó. (Đệ quy)

+ Path sẽ chứa các state đã xuất hiện.

+ Plan sẽ lưu trữ lại các kết quả được trả về

+ Thực hiện Actions(states) trả ra nhiều states mới và gọi lại AND-search trên các state đó

+ RESULTS (state, action) trả về tất cả state mới khi thực hiện action tại state hiện tại.

+ Điều kiện dừng :

* Kiểm tra State có phải là Goal hay không, nếu là Goal thì trả về empty plan (Đã tìm đến đích).
* State đang xét đã trong Path, trả về failure (loại nhánh ra khỏi solution).
* Trên AND-nodes nếu states = failure thì trả về OR-nodes trước đó

+ Nếu Plan khác failure thì Solution đã được tìm thấy. Mặt khác, nếu Plan = failure thì trả về failure

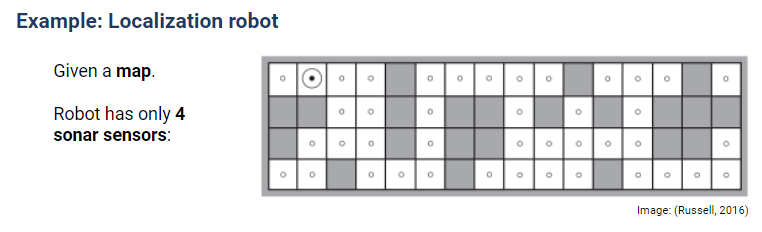
(W8.2) 2 SEARCHING IN NON-FULLY-OBSERVABLE ENVIROMENTS

* Không biết được một phần của môi trường và vị trí của một agen trong môi trường đó
* Ưu điểm:
  + Tiết kiệm thời gian để lấy thông tin
* Nhược điểm:
  + Không xử lý được nhiều bài toán.
  + Xử lý vấn đề không hiệu quả khi thực hiện trong một môi trường lớn

1. Search with no observation:

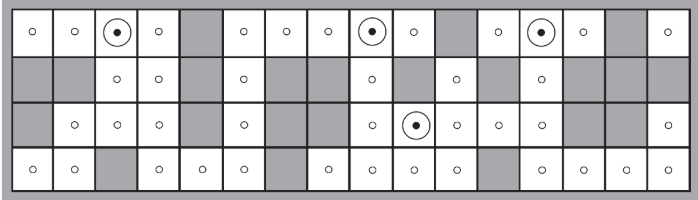
* Ý tưởng giải:
  + Trong một môi trường vật lý, chúng ta không thể xác định được vị trí của agent.Từ một môi trường vật lý, chúng ta chuyển thành 1 Belief env.
  + Trong một belief env. bao gồm các belief states(tập hợp các state mà agent có thể đang ở đó)
* Xây dựng belife sate:
  + Initial state: Tất cả các state trong thực thể có thể có trong belief env.
  + Possible actions:
    - ACTIONS(belief state b) 🡪 b = {s1, s2}
    - actionsp(s1) = {a, b, c}
    - actionsp(s2) = {a, c, d}
    - có thể chọn actions(b) = {a, b, c, d}
    - hoặc = {a, b}
  + Transition model: RESULT (belief state b, action a) = new belief state b’
    - b’ ={s’:s’ = Resultp (s, a) and sϵb}
  + Goal test: toàn bộ trạng thái trong belief state để là goal state
    - Nếu b = {s1, s2,s3} đều về goal thì trả về true.
    - Ngược lại trả về false.
  + cost: costp()

(W9): II. Searching for partially observable problem:

* Tương tự như Search with no observation (Physical env. 🡪 Belief env.)nhưng transition model sẽ được thu hẹp dần nhờ observable (hay còn gọi là percept ), nếu observable thỏa điều kiện thì state đó được giữ lại trong belief states.
* Ví dụ:

Belief state ban đầu là tất cả các vị trí có trên map

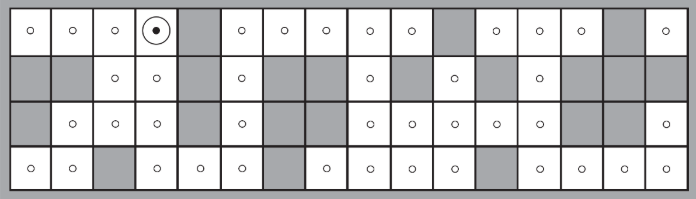
* Bước 1: Di chuyển sang phải



Percept có 1 tường chặn ở trên

🡪Belief state được thu hẹp còn lại 4 state

* Bước 2: Di chuyển sang phải



Percept có tường chặn ở trên và bên phải

🡪Belief state còn lại một state

(W9.2) SEARCHING WITH NO TRANSITION MODEL – ONLINE SEARCH

1. Online searching:

* So sánh Online search và offline search:
  + Offline search
    - Là các thuật toán chạy trên máy tính trước tiên, để tìm ra solution và đưa các agent vào trong thực tế để tìm ra solution. Mỗi action của agent đều có thể biết được kết quả dựa trên transition model.
    - Offline search gồm 5 thành phần : initial state, possible actions, transition model, goal test, cost.
  + Online search
    - Là các thuật toán vừa chạy trong môi trường thực tế vừa học để tìm ra solution, các agent được đặt vào trong thực tế, tại thực tế các agent mới được tìm kiếm đến khi gặp được goal và dừng lại.
    - Online search gồm 4 thành phần : initial state, possible actions, goal test, cost.
* Ưu điểm của Online search:
  + Có Penalty khi chương trình thực hiện quá lâu, giúp chương trình vừa tìm chạy vừa tìm solutions.
  + Không có transition model, giúp chương trình có khả năng giải các chương trình ở tầm tổng quát hơn.

1. Competitive ratio:

* Trong online search có 2 loại của path costs :
  + Actual path cost : cost thực tế mà agent thực hiện, bao gồm các bước agent được chạy trong môi trường và tìm ra solution
  + Shortest path cost : cost ngắn nhất để tìm ra solution

Competitive ratio = actual path cost / shortest path cost (độ hiệu quả của thuật toán)

1. Dead-end state:

* Các agent không được thực hiện trên môi trường giả lập mà chạy trên môi trường thực tế, nên môi trường phải đảm bảo an toàn.
* Các agent khi thực hiện thì không thể quay lại được các bước trước đó.

1. Online depth-first search agent:

* Đầu vào : (s’) là một percept cho biết thông tin các state hiện tại
* Đầu ra : với mỗi state mà agent nghĩ mình đang ở đó, và trả về một action đối với 1 state dựa trên việc agent có thể tự xác định được việc mình nên làm gì.
* Mô tả thuật toán:
  + persistent (biến toàn cục)
    - Result : trả về một bảng chứa (s, a , s’)
    - Untried : một mảng chứa các state và action chưa được thử qua.
    - Unbacktracked : một mảng chứa các state của các trạng thái trước đó chưa được xét tới.
  + Kiểm tra xem (s’) có phải là trạng thái Goal chưa, nếu đúng thì dừng lại.
  + Nếu (s’) là một sate mới thì ta đưa các Action(s’) vào untried[s’].
  + Nếu s không null (nó chỉ null ở lúc bắt đầu) thì ta qua bước 7.
    - Ta đưa [s, a] 🡨 s’ vào bảng result (có nghĩa là ghi được một dòng ở state s thực hiện action a thu được state s’)
    - Thêm (s) và unbacktracked[s’]
  + Nếu action của (s’) là rỗng thì return a.
    - Nếu unbacktracked[s’] là rỗng thì ta dừng lại.
    - Ngược lại, unbacktracked[s’] không là tập rỗng, thì ta thực hiện action b để quay lại trạng thái trước đó để thực hiện (backtrack)
  + Nếu Action [s’] không rỗng thì ta lấy ra 1 action từ untried[s’]
  + s’ = s.
  + return action a.

1. **Online A\* search : ( tìm được solution tốt một cách nhanh chóng )**

* Ý tưởng : ở từng bước, chúng ta chọn một successor’s theo estimated

-

**VI. Learning Real – Time A\* agent :**

* Đầu vào : (s’) percept cho biết cái state hiện tại
* Đầu ra : với mỗi state mà agent nghĩ mình đang ở đó, và trả về một action đối với 1 state dựa trên việc agent có thể tự xác định được việc mình nên làm gì.

**(W9.3)** **Constraint Satisfactoin Problems**

1. **Constraint satisfaction problems (CSP)**

* Atomic state representations : một state tương đương như một thực thể mà ta không thể nhìn thấy và mô tả chi tiết bên trong.
* Factored representation : một state là một tập hợp của các biến và ràng buộc.
* Constraint satisfaction problems : tìm ra phép gán cho state để tất cả các giá trị thỏa mãn ràng buộc.
* Trong CSP được miêu tả bởi 3 components :
  + X ( tập hợp tất cả các biến )
  + D ( tập hợp các miền giá trị mà các biến)
  + C ( tập hợp các ràng buộc mà các biến phải thỏa mãn )
* Solution của CSP : gán giá trị cho tất cả các biến với giá trị thuộc domains của X sao cho thỏa mãn tất cả các ràng buộc trong C

**(W10) II. Constraint propagation:**

* Ý tưởng: thu hẹp domains của các biến xung quanh nhờ sử dụng các biến có domains là 1 giá trị. CSP giải quyết các bài toán nhanh hơn các thuật toán search thông thường trong nhiều trường hợp không gian tìm kiếm quá rộng thì các thuật toán search không tìm ra được nhưng CSP vẫn tìm ra được.
* Cụ thể: sử dụng những ràng buộc để thu hẹp domain cho các biến, khi một biến nào đó đã được thu hẹp domain thì có thể nó sẽ ảnh hưởng đến domain của các biến khác Đến khi thu hẹp domain cho mỗi biến chỉ còn 1 giá trị thì chúng ta đã tìm được solution

**III. Local consistency (node , arc , path )**

* Là một trạng thái của CSP mà trong đó là 1 nhóm các giá trị còn lại của các domain
  + Consistency : các biến thỏa mãn các ràng buộc thì ta gọi là consisten
  + Cách hoạt động của local consistency : mình làm cho các biến nằm gần nhau thỏa mãn hết các ràng buộc, sau đó lan truyền sự thỏa mãn điều kiện đó sang các biến xung quanh, để đảm bảo các biến xung quanh consisten. (loại bỏ những giá trị nằm trong domain làm cho biến đó không consisten )

IV. Constraint graph:

* Nodes: là tất cả các biến (variables)
* Arcs: là 2 biến trong 1 constraints.

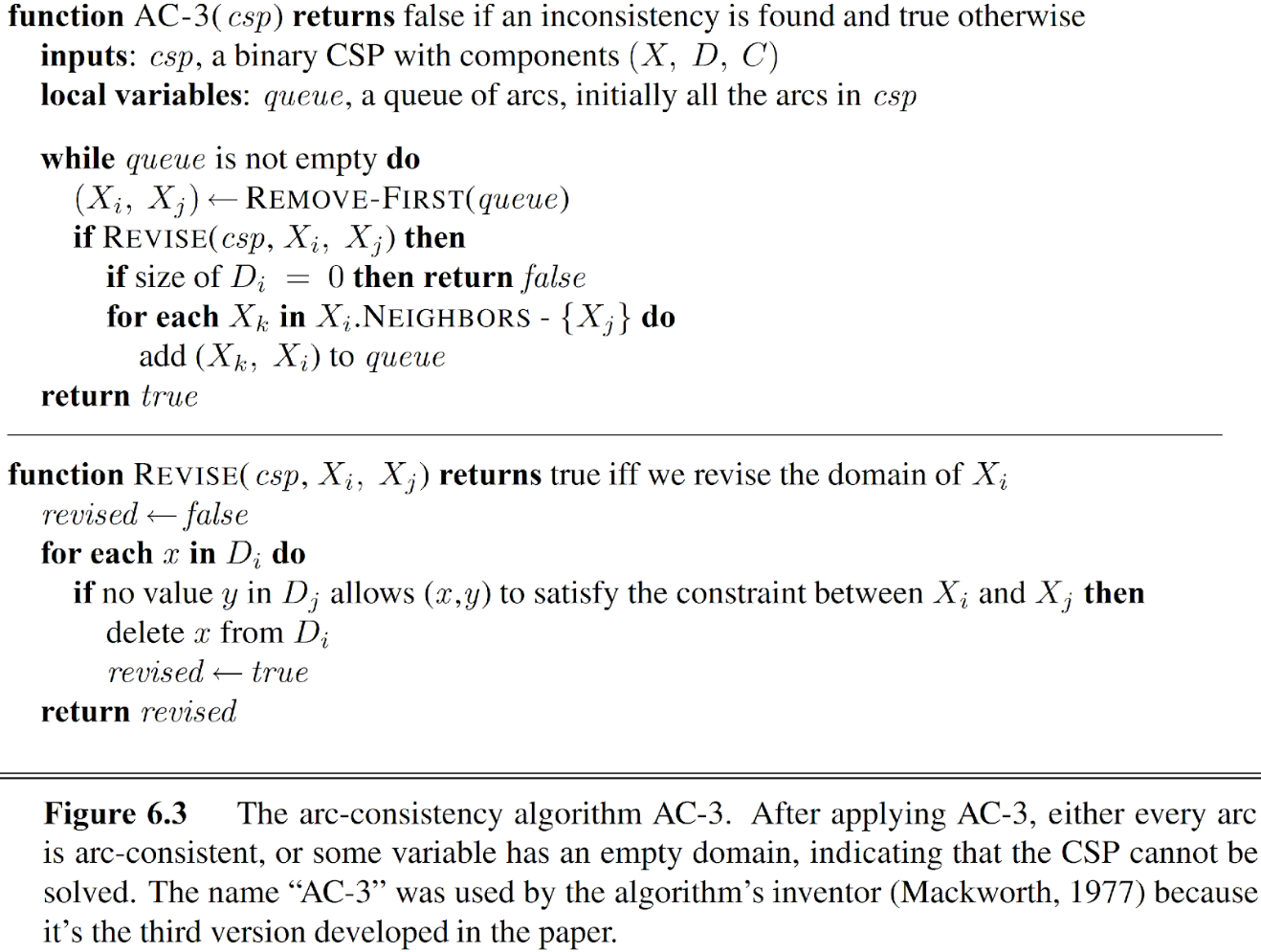
1. Node consistency :

- Đảm bảo constraints 1 biến

- Thu hẹp domains sao cho các domains còn lại thỏa mãn tất cả các constraints của một biến.

VI. Arc consistency

* Đảm bảo các constraints 2 biến.
* Một biến Xi được gọi là Arc-consistent với Xj nếu như lấ 1 giái trị của Xi thì luôn tìm được 1 giá trị của Xj

VII. AC-3 algorithm :

* Đầu vào thuật toán AC-3 của CSP gồm có 3 components (X, D, C)
* inputs (CSP, một binary CSP với các components (X, D, C))
* queue (tập hợp của những constraint có 2 biến)
* Chạy lần lượt qua tất cả các constraint đó
  + Lấy ra 1 constraint(Xi, Xj), sau đó đưa (Xi, Xj) vào hàm Revise
  + Nếu hàm Revise trả về true ( có thu hẹp domain của Xi) thì ta kiểm tra tập domain.
    - Nếu tập domain của Di = 0 (có nghĩa là không có giá trị nào) thì ta return false (bài toán không có solution)
    - Chọn ra biến Xk  lân cận với biến Xi. Thêm (Xk , Xi) vào queue
* Hàm Revise (csp, Xi, Xj)🡪 trả về true nếu có thu hẹp domain của Xi, false nếu không thu hẹp được domain của Xi
* Gán giá trị cho biến revised là false
* Lấy ra từng giá trị x trong tập domain Di của Xi .Xét nếu không tồn tại giá trị y trong tập domain Dj của Xj , giúp cho (x, y) thỏa mãn ràng buộc thì ta qua loại x ra khỏi tập Di
  + Gán cho revised là true
* Trả về revised
* Nếu đã thực hiện xét xong các constraint trong queue thì trả về true.

1. **Path consistency :**

* Sử dụng cho các constraint 3 biến. (2 constraints 2 biến nhưng có 1 biến chung)
* Với bộ 2 biến { Xi, Xj } được xem là path – consistency , khi ta có { Xi = a, Xj = b} sau đó ta tìm được giá trị cho Xm để thoản mãn constraints với {Xi, Xm} và {Xm, Xj}.

1. **K-consistency : (dạng tổng quát của local consistency)**

* Ý tưởng : lấy ra một tập có K -1 biến , sau đó tìm giá trị cho biến thứ K, nếu tìm được thì nó đạt K-consistency, nếu tìm không được thì nó không đạt K-consistency. K càng lớn thì càng thu hẹp được domains nhiều nhưng việc kiểm tra thuật toán càng tốn thời gian hơn.

1. **Global constraints :**

* Định nghĩa : là một constraint có số lượng biến bất kỳ.
* Ví dụ : Alldiff constraint, Atmost constraint
  + Alldiff( x1, x2, x3, x4,….x9)
  + Atmost(10,P0, P1, P2, P3, P4, P5, P6, P7)

1. **Solving a Sodoku puzzle : (AC – 3, Triplets )**

* Biến: 81
* Domain: Di{1,2,3,4,5,6,7,8,9}; Dj={a}
* Constraint:
  + Alldiff( x1, x2, x3, x4,….x9)(x27)
* Cách giải :
  + B1(AC-3 algorithm) : Thu hẹp domains cho các biến bằng cách sử dụng constraints
  + B2(Triplets) : Tìm ra một unit chứa các biến trong một Alldiff, có chứa 3 ô với domain là tập con của 3 con số, ta có thể loại bỏ 3 con số đó ra khỏi domain.