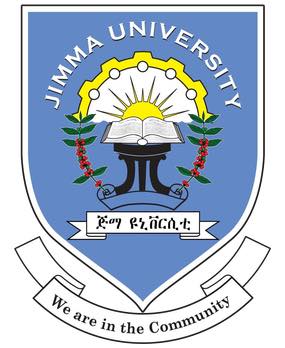
****

**JIMMA INSTITUTE OF TECHNOLOGY**

**SCHOOL OF COMPUTICS AND INFORMATICS**

**DEPARTMENT OF COMPUTER SCIENCE**

**SECTION TWO**

**AI ASSIGNMENT ONE**

## GROUP MEMBER

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **FNAME** | **LNAME** | **ID NO** |
| **1** | **NANIYE** | **DEMIRE** | **RU2868/11** |
| **2** | **YONAS** | **TARIKU** | **RU3084/11** |
| **3** | **BELETE** | **WASSIE** | **RU 2795/11** |
| **4** | **MABRATE** | **AFARA** | **RU0757/10** |

## 

**SUBMITTED TO:-DESALEGN YOHANNES**

**FACULTY OF COMPUTING, JIT, JIMMA UNIVERSITY, JIMMA, OROMIA, ETHIOPI**

**CHAPTER ONE**

**CONSTRAINT SATISFACTION PROBLEMS**

Sub\_topic

* Constraint Satisfaction Problems
* Backtracking Search for CSPs
* Variable and value ordering
* Propagating information through constraints
* Intelligent backtracking: looking backward

The Structure of Problems

**Constraint Satisfaction Problems**

**constraint satisfaction problems:** or **CSPs** for short, are a flexible approach to searching that have proven useful in many AI-style problems

CSPs can be used to solve problems such as

* **graph-coloring**: given a graph, the a coloring of the graph means assigning each of its vertices a color such that no pair of vertices connected by an edge have the same color
  + in general, this is a very hard problem, e.g. determining if a graph can be colored with 3 colors is NP-hard
  + many problems boil down to graph coloring, or related problems
* **job shop scheduling**: e.g. suppose you need to complete a set of tasks, each of which have a duration, and constraints upon when they start and stop (e.g. task c can’t start until both task a and task b are finished)
  + CSPs are a natural way to express such problems
* **Cryptarithmetic puzzles**: e.g. suppose you are told that TWO + TWO = FOUR, and each of the letters corresponds to a *different* digit from 0 to 9, and that a number can’t start with 0 (so T and F are not 0); what, if any, are the possible values for the letters?
  + while these are not directly useful problems, they are a simple test case for CSP solvers

the basic idea is to have a set of variables that can be assigned values in a constrained way

* Constraint Satisfaction Problems

**Constraint satisfaction problems**, whose states and goal test

Conform to a standard, structured, and very simple **representation**

CSPs are a special kind of problem:

states defined by values of a fixed set of variables

goal test defined by constraints on variable values

CSP is defined by 3 components (X, D, C):

state: a set of variables *X*, each *Xi* , with values from domain *Di*

goal test: a set of constraints *C*, each *Ci* involves some

subset of the variables and specifies the allowable combinations of values for that subset Each constraint *Ci* consists of a pair <scope, rel>, where scope is a tuple of variables and rel is the relation, either represented explicitly or abstractly

Formally speaking, a **constraint satisfaction problem** (or CSP) is defifined by a set of **vari-ables**, X1, X2, . . . , Xn, and a set of **constraints**, C1, C2, . . . , Cm. Each variable Xi has a

nonempty **domain** Di of possible **values**. Each constraint Ci involves some subset of the

variables and specififies the allowable combinations of values for that subset. A state of the

problem is defifined by an **assignment** of valuesto some or all of the variables, {Xi = vi, Xj = vj , . . .}. An assignment that does not violate any constraints is called a **consistent** or legal assignment. A complete assignment is one in which every variable is mentioned, and a **solution** to a CSP is a complete assignment that satisfifies all the constraints. Some CSPs also require a solution that maximizes an **objective function**.

Example:

X1 and X2 both have the domain {A, B}

Constraints:

<(X1, X2), [(A, B), (B, A)]>, or

<(X1, X2), X1 ≠ X2>

Solution

Each state in a CSP is defined by an assignment of values to some or all of the variables

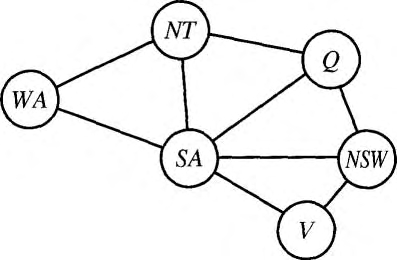
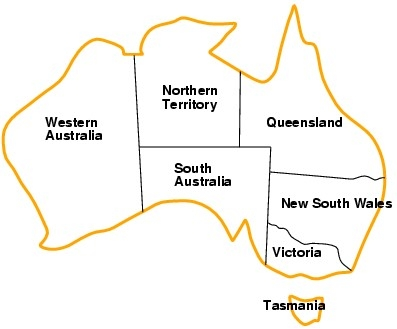
An assignment that does not violate any constraints is called a consistent or legal assignment

A complete assignment is one in which every variable is assigned

A solution to a CSP is consistent and complete assignment

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

Example: Map Coloring

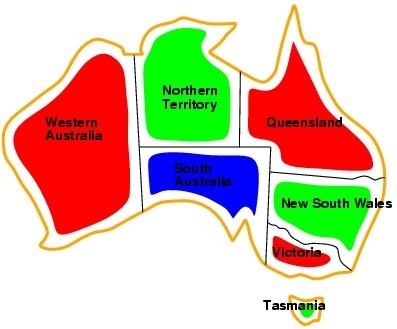
Variables: X = {*WA, NT, Q, NSW, V, SA, T* }

Domains: *Di* = {red, green, blue}

Constraints: adjacent regions must have different colors

Solution?

Solution: Complete and Consistent Assignment



Variables: X = {*WA, NT, Q, NSW, V, SA, T* }

Domains: *Di* = {red, green, blue}

Constraints: adjacent regions must have different colors

Solution? {WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = red}.

* Backtracking Search for CSPs

The term **backtracking search** is used for a depth-first search that chooses values for

one variable at a time and backtracks when a variable has no legal values left to assign.

A simple backtracking algorithm for constraint satisfaction problems. The algorithm is modeled on the recursive depth-first search

* **Variable and value ordering**

The backtracking algorithm contains the line

Var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp).

By default, SELECT-UNASSIGNED-VARIABLE simply selects the next unassigned variable

in the order given by the list VARIABLES[csp]. This static variable ordering seldom results in

the most efficient search. For example, after the assignments for WA = red and NT = green,

there is only one possible value for SA, so it makes sense to assign SA = blue next rather than

assigning Q. In fact, after SA is assigned, the choices for Q, NSW , and V are all forced. This

intuitive idea—choosing the variable with the fewest “legal” values—is called the **minimum**  **remaining values** (MRV) heuristic. It also has been called the “most constrained variable” or

“fail-first” heuristic, the latter because it picks a variable that is most likely to cause a failure

soon, thereby pruning the search tree. If there is a variable X with zero legal values remaining, the MRV heuristic will select X and failure will be detected immediately—avoiding pointless searches through other variables which always will fail when X is finally selected

* **Propagating information through constraints**

So far our search algorithm considers the constraints on a variable only at the time that the

variable is chosen by SELECT-UNASSIGNED-VARIABLE. But by looking at some of the

constraints earlier in the search, or even before the search has started, we can drastically reduce the search space.

**Constraint propagation :**is the general term for propagating the implications

of a constraint on one variable onto other variables; in this case we need to propagate from

WA and Q onto NT and SA, (as was done by forward checking) and then onto the constraint

between NT and SA to detect the inconsistency. And we want to do this fast: it is no good

reducing the amount of search if we spend more time propagating constraints than we would

have spent doing a simple search.

* **Intelligent backtracking: looking backward**

The BACKTRACKING-SEARCH algorithm has a very simple policy for what to

do when a branch of the search fails back up to the preceding variable and try a different

value for it. This is called **chronological backtracking**, because the *most recent* decision

point is revisited. In this subsection, we will see that there are much better ways.

Consider what happens when we apply simple backtracking with a fixed

variable ordering Q, NSW , V , T, SA, WA, NT. Suppose we have generated the partial

assignment {Q = red, NSW = green, V = blue, T = red}. When we try the next variable,

SA, we see that every value violates a constraint. We back up to T and try a new color

for Tasmania! Obviously this is silly—recoloring Tasmania cannot resolve the problem with

South Australia.

A more intelligent approach to backtracking is to go all the way back to one of the

set of variables that *caused the failure*. This set is called the **conflict set**; here, the conflict

set for SA is {Q, NSW , V }. In general, the conflict set for variable X is the set of previously assigned variables that are connected to X by constraints.

* The Structure of Problems

*structure* of the problem, as represented by the constraint graph, can be used to find solutions quickly. Most of the approaches here are very general and are applicable to other problems besides CSPs, for example probabilistic reasoning. After all, the only way we can possibly hope to deal with the real world is to decompose it into many sub problems

CHAPTER TWO

PROBLEMS SOLVIN G HEIRUSTIC SEARCHING STRATAGY

problems solving heuristic searching strategy:

Heuristic Search Techniques

Heuristic Dependence

Best-First Search

Simulated Annealing

Modified State Evaluation

Admissible Heuristics

* problems solving heuristic searching strategy:

is a goal-based agent called a **problem-solving agent** Problem-solving agents think about the world using **atomic** representations, as described in visible to the problem-solving algorithms. Goal-based agents that use more advanced **factored** or **structured** representations are usually called **planning agents**

We start our discussion of problem solving by defining precisely the elements that constitute a “problem” and its “solution,” and give several examples to illustrate these definitions.

We then describe several general-purpose search algorithms that can be used to solve these problems. We will see several **uninformed** search algorithms—algorithms that are given no information about the problem other than its definition. Although some of these algorithms can solve any solvable problem, none of them can do so efficiently. **Informed** search algorithms, on the other hand, can often do quite well given some idea of where to look for solutions. the solution to a problem is always a *fixed sequence* of actions

A **problem** can be defined formally by five components:

The **initial state** that the agent starts in. For example, the initial state for our agent in Romania might be described as In(Arad).

• A description of the possible **actions** available to the agent. Given a particular states,

ACTIONS(s) returns the set of actions that can be executed in s. For example,from the state In(Arad), the possible actions are {Go(Sibiu), Go(Timisoara), Go(Zerind)}.

• A description of what each action does; the formal name for this is the

**transition model:** specified by a function RESULT(s, a) that returns the state that results from TRANSITION MODEL doing action a in state s. We will also use the term **successor** to refer to any state SUCCESSOR reachable from a given state by a single action RESULT(In(Arad),Go(Zerind)) = In(Zerind) .

Together, the initial state, actions, and transition model implicitly define the **state space**

STATE SPACE of the problem—the set of all states reachable from the initial state by any sequence of actions. The state space forms a directed network or **graph** in which the nodes

GRAPH are states and the links between nodes are actions. (The map of Romania for two driving actions, one in each direction.) A **path** in the state space is a sequence PATH of states connected by a sequence of actions.

2 Many treatments of problem solving, including previous editions of this book, talk about the **successor function**, which returns the set of all successors, instead of actions and results. Although convenient in some ways, this formulation makes it difficult to describe an agent that knows what actions it can try but not what they achieve.

Heuristic Search Techniques

* Direct techniques (blind search) are not always possible (they require too much time or memory).Weak techniques can be effective if applied correctly on the right kinds of tasks. Typically require domain specific information.
* 8 Puzzle Heuristics Blind search techniques used an arbitrary ordering (priority) of operations. Heuristic search techniques make use of domain specific information - a heuristic. What heuristic(s) can we use to decide which 8-puzzle move is “best” (worth considering first).

For now - we just want to establish some ordering to the possible moves (the values of our heuristic does not matter as long as it ranks the moves).Later - we will worry about the actual values returned by the heuristic function

. A Simple 8-puzzle heuristic

* Number of tiles in the correct position.The higher the number the better. Easy to compute (fast and takes little memory).Probably the simplest possible heuristic.
* Heuristic Dependence :Hill climbing is based on the value assigned to states by the heuristic function. The heuristic used by a hill climbing algorithm does not need to be a static function of a single state. The heuristic can look ahead many states, or can use other means to arrive at a value for a state.
* Best-First Search :Combines the advantages of Breadth-First and Depth-First searchs.DFS: follows a single path, don’t need to generate all competing paths.BFS: doesn’t get caught in loops or dead-end-paths. Best First Search: explore the most promising path seen so far.
* Simulated Annealing :Based on physical process of annealing a metal to get the best (minimal energy) state. Hill climbing with a twist: allow some moves downhill (to worse states)start out allowing large downhill moves (to much worse states) and gradually allow only small downhill moves.
* Modified State Evaluation :Value of each state is a combination of: the cost of the path to the state estimated cost of reaching a goal from the state. The idea is to use the path to a state to determine (partially) the rank of the state when compared to other states. This doesn’t make sense for DFS or BFS, but is useful for Best-First Search. Why we need modified evaluation?
* Consider a best-first search that generates the same state many times. Which of the paths leading to the state is the best? Recall that often the path to a goal is the answer (for example, the water jug problem) Admissible Heuristics Given an admissible heuristic h’, path length to each state given by g, and the actual path length from any state to the goal given by a function h. We can prove that the solution found by A\* is the optimal solution.