1.1 Agentes Resolventes de Problemas

Siguiendo el enfoque de grafos, el agente más sencillo es el agente resolvente de problemas (Problem-Solving Agent) es un tipo de agente basado en objetivos que es el más sencillo entre los que planifican. Este será el agente que se utilice en búsquedas por que las búsquedas responden a un proceso de planificación. Para las búsquedas mas sencillas se utilizan representaciones atómicas.

Lo primero que se debe hacer es establecer los objetivos, por cuanto esto simplifican los problemas, ya que organizan el comportamiento del agente, limitando sus metas. Por cuanto a medida que se vayan expandiendo los árboles de búsqueda se debe ir verificando si se ha completado el objetivo para detener el proceso de búsqueda iniciado. En problemas donde se tiene claro cual es el objetivo, simplifica mucho a la hora de definir el problema, modelizarlo y definir el agente. Donde el agente deberá descubrir la secuencia de acciones que permita alcanzar el objetivo, lo cual en los grafos son los diferentes nodos, desde el nodo hoja al nodo raíz que representa un camino de solución a un determinado problema. Los grafos son una manera de modelizar una secuencia de acciones, por cuanto se toma un nodo, se aplica una acción y se va a otro nodo y asi sucesivamente hasta llegar a la solución o desubrir que no existe alguna.

El primer paso para resolver un problema es formular sus objetivos. Problemas en los que no tenemos claros los objetivos, no son adecuados para este tipo de agentes.

Una vez que se tengan claros los objetivos, el siguiente paso es la formulación del problema, siendo el proceso para decidir que acciones y estados se deben considerar dado un objetivo (modelizacion). Para esto se hace uso de la abstracción (modelo), que es un proceso de eliminación de detalles de una representación y dejar solo lo justo y necesario para poder solucionar el problema. Es decir lo primero que debemos realizar es un modelo, pudiendo emplearse una representación numérica, simbólica, grafos entre otros. Por ejemplo si dese construir un modelo para un agente que pueda salir de un laberinto, no interesara el material de las paredes, si el piso es asfalto, si existen plantas en el piso entre otros detalles que no son relevantes a los objetivos.

Es importante considerar que, dentro de la formulación del problema, un agente que tiene distintas opciones inmediatas (acciones) pero con valores desconocidos puede decidir que hacer evaluando las posibles diferentes acciones futuras que al final le lleven a estados de valores conocidos. Es decir, cuando tienes varias acciones para realizar y no sabes que acción realmente te acerca a tu objetivo, en ese caso lo que se tiene que hacer es probar las diferentes acciones, y considerar los efectos en relación a posibles acciones futuras, analizando cuales se acercan o alejan del objetivo, lo cual se logra a través de la utilización de las heurísticas, las cuales las estudiaremos posteriormente.

Los algoritmos basados en grafos sirven para resolver un determinado tipo de problemas, considerando una clasificación que engloba a todos los algoritmos de búsqueda (no informada e informada, algoritmos de juegos, de satisfacción de restricción entre otros), cumplen una serie de características de propiedades en su entorno, es decir el tipo de problema tiene que cumplir una serie de características:

1. En la búsqueda en grafos en principio se entiende que el entorno es totalmente observable (no es parcialmente observable), con algunos casos excepcionales, que no son sujeto de estudio de la presente sección.
2. Es determinístico, no es estocástico.
3. Son discretos, es decir tienen estados concretos. No son continuos, si ese fuese el caso hay que discretizarlo.
4. Se da por hecho que pertenecen a escenarios que se conocen sus reglas, sus leyes, son nudos conocidos.
5. Puede ser un agente individual o multiagente.
6. Pueden ser episódicos o secuenciales.
7. Puede ser estático o dinámico.

La solución que nos devuelve un grafo es una secuencia fija de acciones, por cuanto para tratar de resolver un problema considerando un enfoque basado en grados, sebe cumplir obligatoriamente las propiedades anteriormente indicadas.

Ahora que contamos con los elementos teóricos suficientes, podemos definir lo que es búsqueda, como un proceso de encontrar una secuencia de acciones (solución) para alcanzar un objetivo. Donde la ejecución es llevar a cabo la solución.

Muy importante en este tipo de problemas de agentes, ignoran el resto de percepciones que reciben mientras esta calculando y ejecutando una solución (lo que se llama lazo abierto en Teoría de Control).

Todo lo explicado se plasma en el siguiente algoritmo:

```
función AGENTE-RESOLVENTE-PROBLEMAS(percepción)
devuelve acción
  datos: solución: secuencia de acciones (vacía).
         estado  : el actual del mundo en cada momento.
         objetivo: estado a alcanzar (nulo).
         problema: formulación del problema a resolver.
  estado ← ACTUALIZAR-ESTADO(percepción)
  si solución.ESTÁ-VACÍA() entonces
    objetivo ← FORMULAR-OBJETIVO(estado)
    problema ← FORMULAR-PROBLEMA(estado, objetivo)
    solución ← BUSCAR-SOLUCIÓN(problema)
    si solución = fallo entonces devolver acción-nula
  acción ← EXTRAER-PRIMERA-ACCIÓN(solución)
  devolver acción
```

Donde la función Buscar-Solución será reemplazada por cualquiera de los diferentes algoritmos que existen, como es el caso del Algoritmo Primero en Anchura.

1.2 DEFINICION DE PROBLEMAS DE BUSQUEDA.

Para definir un problema de manera que un agente pueda atacarlo se deben considerar los siguientes cinco puntos:

1. **Estado Inicial**, es el estado que percibe el agente racional al comenzar a solucionar el problema, desde donde parte, como es el mundo al principio. Su pseudocódigo es:
   problema.ESTADO-INICIAL -> estado
2. **Acciones,** descripción de las posibles acciones que se puede realizar (aplicar) el agente en cada estado (conjunto). No es suficiente con definir el conjunto de las acciones que el agente puede realizar, sino que a su vez se debe definir en cada estado que acciones se pueden ejecutar. Su pseudocódigo es:
   problema.ACCIONES(estado) -> acciones
3. **Modelo de transición**, es el elemento más importante en la definición de un problema y es responsable de realizar la descripción de lo que hace cada acción en cada estado, siendo el sucesor el nuevo estado resultante, se implementa a través de una función y conocida con el nombre de función Sucesor. Su pseudocódigo es:
   problema.RESULTADO(estado, acción) -> estado-sucesor
4. **Función Objetivo**, función que nos permite establecer si se ha llegado al objetivo o no. Determina si un estado es un objetivo o no. Define el conjunto de estados finales. Se implementa a través de una función que se la conoce también como función test. Su pseudocódigo es:
   problema.ES-OBJETIVO(estado) -> VERDADERO/FALSO

5. **Función coste de camino**, asigna un coste a cada camino. Permite medir el rendimiento. Regularmente, a cada acción se le asigna un coste positivo, y el coste del camino será la suma de las que componen el camino. Donde la solución optima será el camino de coste mínimo. . Su pseudocódigo es:
   problema.COSTE-ACCION(estado, acción) -> coste
   problema.COSTE-CAMINO(estado) -> coste

1.3 ESPACIO DE ESTADOS

Es una composición, una unión del estado inicial, las acciones y el modelo de transiciones, siendo el conjunto de estados alcanzables a partir del estado inicial. El espacio de estados se suele modelizar como un grafo dirigido digrafo, donde los enlaces son acciones y los caminos son secuencias de estados y acciones. No es necesario almacenar todos los estados del espacio de estados, cuando se pueden obtener estos a partir de un determinado nodo, recorriendo el árbol en orden inverso a partir del nodo objetivo hasta el nodo raíz.

1.4 TIPOS DE PROBLEMAS

1.4.1 PROBLEMAS DE JUGUETE

Ilustran y permiten definir y probar algoritmos. Definición clara y concisa, se utilizan para comparar rendimiento de algoritmos. Incluso en este tipo de problemas muchos algoritmos ya son intratables. Sirven de base para explicar algoritmos más complejos.

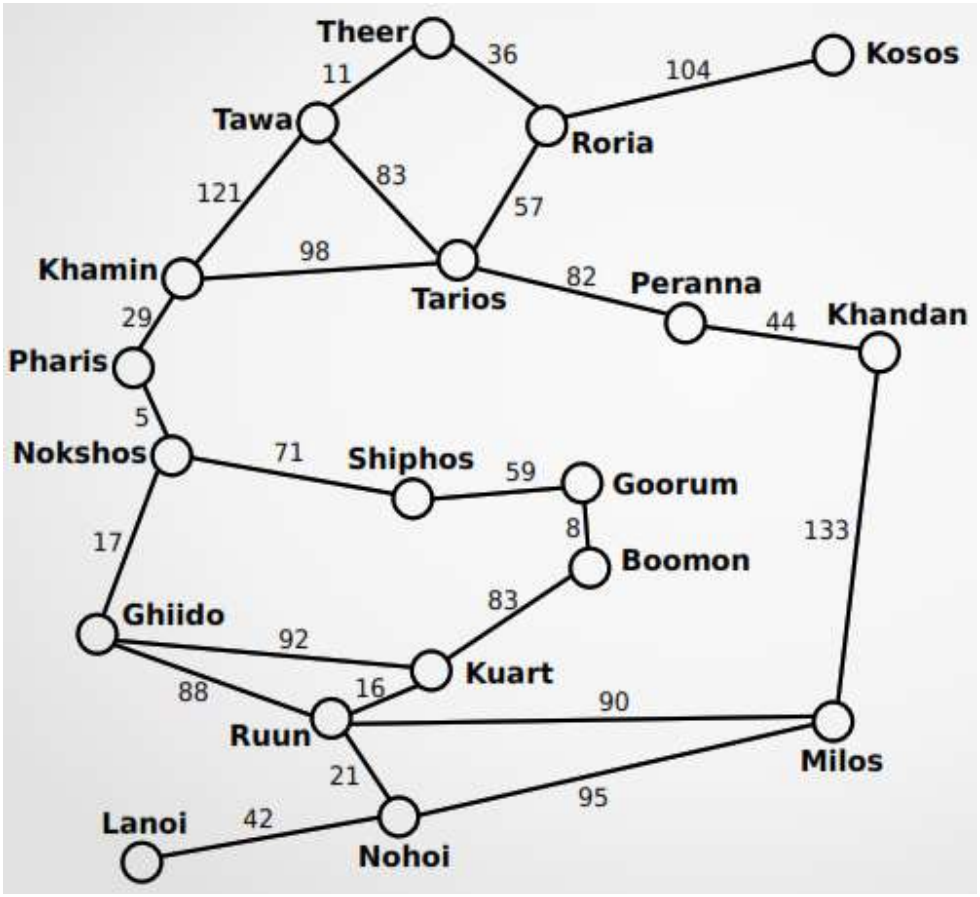
1.4.2 PROBLEMAS DEL MUNDO REAL

Sus definiciones no son claras, y suelen ser muy complejos, pero son los que la gente está interesada en solucionar.

1.5 BUSQUEDA DE SOLUCIONES

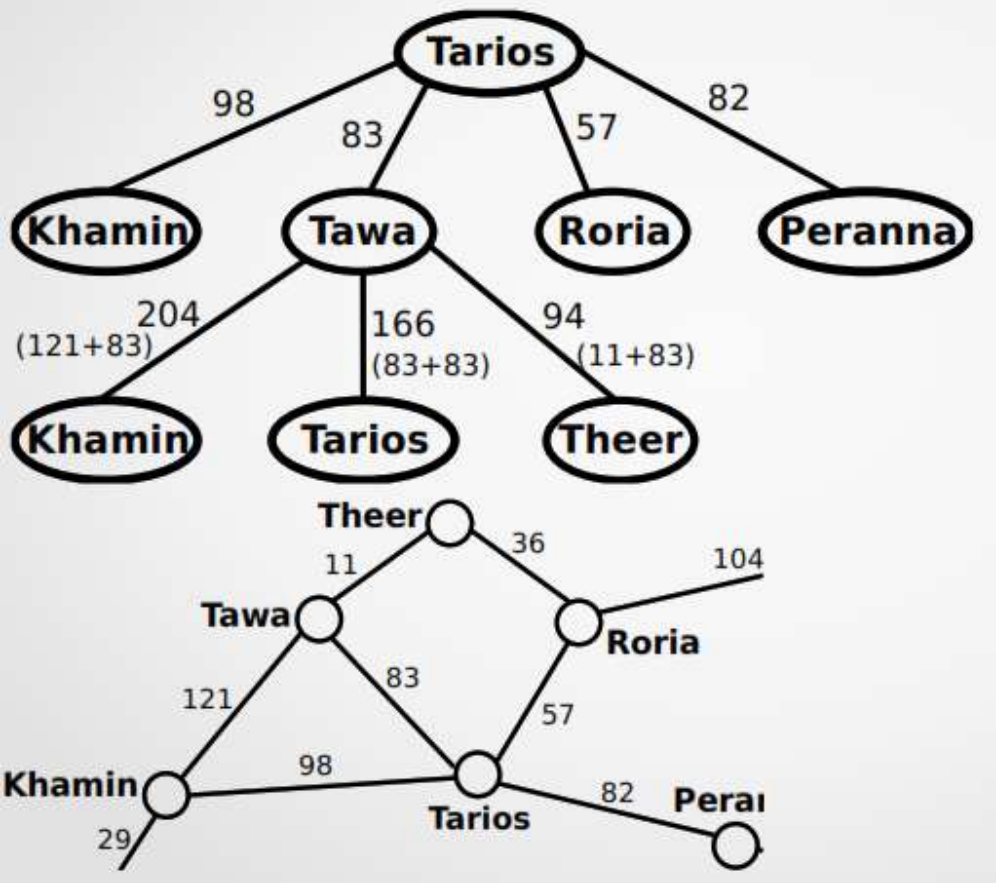
Algo importante es establecer la diferencia entre lo que es en si el problema y la modelización del problema. Cuando se tiene un problema se lo modeliza de una forma y ese modelo se lo suele utilizar en algún tipo de grafos y los estudiantes suelen confundir el grado utilizado para modelizar el problema con el árbol de búsqueda donde se va a realizar las busquedas. Para dejar clara la diferencia el problema se modeliza a través de un gráfico, que se denomina gráfico del espacio de estados, donde ese gráfico es la representación del problema, cada nodo de ese gráfico es un estado y los enlaces son acciones realizadas para pasar de un estado a otro. Los objetivos será uno o mas estados del conjunto total de estados (subconjunto). Cada estado aparee una y solo una única vez. Suele ser demasiado grande para construirlo completo, pudiendo ser incluso infinito, principalmente los que resultan de modelizar problemas del mundo real.

Un ejemplo tradicional de modelo de problema representado por grafos es el de las ciudades y carreteras.


Theer
36
Kosos
11
104
Tawa
Roria
121
83
57
Khamin
98
Tarios
82
Peranna
29
44
Khandan
Pharis
5
Nokshos
71
Shiphos
59
Goorum
8
133
17
Boomon
83
Ghiido
92
Kuart
88
16
90
Ruun
Milos
21
95
Lanoi
42
Nohoi

Los problemas se modelizan en su generalidad por grafos, en cambio los árboles de búsqueda, son árboles como su nombre indica, en esos árboles los nodos también son estados y las ramas acciones. El nodo raíz del árbol es el estado inicial del problema, los caminos son secuencias de acciones, no se necesita expandir entero el árbol solo la parte que permita alcanzar una solución si es que la existe, siendo este un problema en esta representación, donde los estados se repiten y pueden ser infinitos (problemas del mundo real).

Ejemplo de árbol de búsqueda del anterior grafo del problema.



1.5.1 ESTADOS REPETIDOS

Son nodos distintos que representan al mismo estado, aunque el espacio de estados sea finito, provocan que el árbol sea infinito. Aparecen cuando hay más de una manera de ir de un estado a otro (grafos). Para solucionar esta dificultad se recomienda utilizar las siguientes estructuras:

- **Lista abierta (frontera)**, lista de nodos hojas desde las que se escogen los nodos a expandir. Separa los nodos del grafo en explorados y por explorar.
- **Lista cerrada (explorada)**, conjunto de estados ya explorados para evitar procesarlos de nuevo.

1.6 INFRAESTRUCTURA PARA ALGORITMOS DE BÚSQUEDA

Habiendo revisado todos los aspectos teóricos suficientes para trabajar con búsquedas, es importante ahora desarrollar las estructuras de datos y otros elementos que permitirán la implementación de los algoritmos de búsquedas en cualquier lenguaje de programación, como es el caso de Nodo, Frontera (Lista), Tipos de Listas y Medidas de rendimiento.

1.6.1 Nodo

Considerando la programación orientada a objetos un nodo es una estructura de datos representada por una clase que debe contener las siguientes propiedades (atributos):

- Nodo.ESTADO -> estado, es el estado que representa a un nodo, nodos diferentes pueden representar el mismo estado.
- Nodo.Padre -> nodo, guarda una referencia al nodo padre, al nodo que cuando se expandió los heredo, sirve para recorrer el camino de la solución, sin necesidad de almacenar todo el camino permanentemente en memoria, desde el nodo hoja hasta el nodo raíz.
- Nodo.ACCIÖN -> acción, es la acción que permite generar un nuevo nodo.
- Nodo.ACCIONES -> lista-acciones, acciones que se pueden ejecutar dentro un nodo, considerando que no siempre se pueden ejecutar todas las acciones.
- Nodo.COSTE -> coste-camino, coste asociado a un nodo, es un coste que corresponde con el coste del camino. El coste del camino se implementa a través de una función g(n).

Dentro de las funciones que se implementa para el manejo de los nodos se tiene:

Crear nodos

Permite crear un nodo hijo, donde se pasa el problema, el padre y la acción y devuelve el nodo creado.

```
función CREAR-NODO-HIJO(problema, padre, acción)
devuelve nodo
  nodo ← CREAR-NODO()
  nodo.ESTADO   ← problema.RESULTADO(padre.ESTADO,
                                     acción)
  nodo.PADRE    ← padre
  nodo.ACCIÓN   ← acción
  nodo.ACCIONES ← problema.ACCIONES(nodo.ESTADO)
  nodo.COSTE    ← padre.COSTE +
                  problema.COSTE-ACCIÓN(padre.ESTADO,
                                        acción)
  devolver nodo
```

Expandir nodos

Permite crear todos los nodos hijos.

```
función nodo.EXPANDIR(problema) devuelve nodos
  nodos ← CREAR-LISTA()
  por cada acción en nodo.ACCIONES hacer
    nodo-hijo ← CREAR-NODO-HIJO(problema,
                                nodo,
                                acción)
    nodos.AGREGAR(nodo-hijo)
  devolver nodos
```

1.6.2 FRONTERA (Lista)

La frontera se implementa a través de la estructura de datos lista, por que lo que interesa es el orden que esta pueda aplicar a los nodos y dependiendo del orden se emplearán diferentes tipos de listas. La estructura de datos lista debe implementar al menos cuatro métodos:

- Frontera.ESTÁ-VACÍA() -> Verdadero/False, determina si hay elementos en la lista
- Frontera.POP() -> nodo, sacar el primer elemento de la lista y lo devuelve.
- Frontera.AGREGAR(nodo) -> lista, introduce un nodo en la lista para luego ser reordenado.
- Frontera.QUITAR(nodo) - > lista, retira un elemento específico de una lista.

En las búsquedas en grafos se emplean por lo general los siguientes tipos de listas:

- Listas FIFO(Colas), el primer elemento en entrar es el primero en salir (más antiguo)
- Listas LIFO (Pilas), el ultimo elemento en entrar es el primero en salir (más nuevo)
- Listas de Prioridad, sale el de mayor prioridad según función de ordenación.

1.6.3 MEDIDA DE RENDIMIENTO

El trabajo de elaborar algoritmos de búsqueda esta ligado a la acción de comparar algoritmos unos con otros, para establecer un criterio uniforme en relación al rendimiento de los mismo, se considera cuatro datos:

- Completo, es completo si asegura que encuentra la solución.
- Óptimo, si encentra la solución y esta es la mejor.
- Complejidad (Tiempo/Espacio), tiempo de ejecución y memoria.
    - Tamaño del problema
        - Fator de ramificación (b), de media cuantas ramas hijos tiene cada estado
        - Profundidad (d), nivel de la solución.
        - Camino máximo(m), a que profundidad esta la primera solución que encuentre

1.6.4 IMPLEMENTACION EN PYTHON

La clase nodos se implementa en el archivo Nodos.py

```
class Nodo:

    def __init__(self, estado, hijo=None):

        self.estado = estado

        self.hijo = None

        self.padre = None

        self.accion = None

        self.acciones = None

        self.costo = None

        self.set_hijo(hijo)


    def set_estado(self, estado):

        self.estado = estado


    def get_estado(self):

        return self.estado


    def set_hijo(self, hijo):

        self.hijo = hijo

        if self.hijo is not None:

            for s in self.hijo:

                s.padre = self


    def get_hijo(self):

        return self.hijo
```

```
def set_padre(self, padre):
    self.padre = padre

def get_padre(self):
    return self.padre

def set_accion(self, accion):
    self.padre = accion

def get_accion(self):
    return self.accion

def set_acciones(self, acciones):
    self.acciones = acciones

def get_acciones(self):
    return self.acciones

def set_costo(self, costo):
    self.costo = costo

def get_costo(self):
    return self.costo

def equal(self, Nodo):
```

```
        if self.get_estado() == Nodo.get_estado():
            return True
        else:
            return False

    def en_lista(self, lista_nodos):
        enlistado = False
        for n in lista_nodos:
            if self.equal(n):
                enlistado = True
        return enlistado

    def __str__(self):
        return str(self.get_estado())
```

1.7 BÚSQUEDA EN ÁRBOL

Si la estructura de datos del problema corresponde con un árbol, se aplica el siguiente algoritmo base para la implementación de los diferentes algoritmos de búsqueda, por lo cual es importante comprender el siguiente pseudocódigo:

```
función BÚSQUEDA-ÁRBOL(problema) devuelve nodo o fallo
  datos: frontera: lista de nodos hoja (vacía).
  frontera.AGREGAR(problema.ESTADO-INICIAL)
  repetir
    si frontera.ESTÁ-VACÍA() entonces
      devolver fallo
    nodo-hoja ← frontera.POP() # Según tipo.
    si problema.ES-OBJETIVO(nodo-hoja) entonces
      devolver nodo-hoja
    nodos ← nodo-hoja.EXPANDIR(problema)
    frontera.AGREGAR(nodos)
```

Esta función devuelve un nodo si existe o un fallo, se utiliza la estructura de datos frontera, la cuál se verifica si está vacía. El primer paso del algoritmo es agregar el estado inicial del problema a frontera. En este algoritmo no se utiliza la estructura de datos explorada, por su relativa simpleza. Luego se inicia un bucle hasta que se revisen todos los nodos o en su caso se encuentre la solución, para lo cual se pregunta si la frontera está vacía y si es el caso se devuelve fallo. Si existen elementos se procede a retirar el primer elemento de la frontera y se lo asigna a nodo hoja, posteriormente se realiza la pregunta si el nodo-hoja es el objetivo, si ese es el caso se devuelve el nodo y termina la función, en caso contrario se expanden los nodos vinculados al nodo hoja actual, para posteriormente agregarlos a la lista frontera y continuar con la lógica del bucle.

Este es un pseudocódigo base para ir construyendo los algoritmos más avanzados, esa es la razón de su simpleza y la importancia de su plena comprensión.

1.8 BUSQUEDA EN GRAFO

El anterior pseudocódigo no sirve para grafos, para lo cual se realizan casi los mismos procedimientos, adicionando algunos pasos que se encuentran subrayados en el pseudocódigo. Los elementos modificados o adicionados son los siguientes: Se agrega una lista para almacenar los elementos explorados, la cuál almacenara los nodos que ya se exploraron para evitar repetir la búsqueda en nodos ya explorados, es por eso que cada nuevo nodo es agregado a explorada, para posteriormente realizar una verificación más compleja al preguntar si cada nodo de los nodos expandidos no está en la frontera y además no está en explorada, para ser agregado a frontera y proceder con el procedimiento de verificación si alguno de estos nodos es la solución.

```
función BÚSQUEDA-GRAFO(problema) devuelve nodo o fallo
  datos: frontera : lista de nodos hoja (vacía).
         explorada: conjunto de estados ya explorados (vacío).
  frontera.AGREGAR(problema.ESTADO-INICIAL)
  repetir
    si frontera.ESTÁ-VACÍA() entonces
      devolver fallo
    nodo-hoja ← frontera.POP() # Según tipo.
    si problema.ES-OBJETIVO(nodo-hoja) entonces
      devolver nodo-hoja
    explorada.AGREGAR(nodo-hoja.ESTADO)
    nodos ← nodo-hoja.EXPANDIR(problema)
    por cada nodo en nodos hacer
      si nodo.ESTADO no está en frontera.ESTADOS()
      y nodo.ESTADO no está en explorada:
        frontera.AGREGAR(nodo)
```