Proyecto IA
UTFSM 2016-1

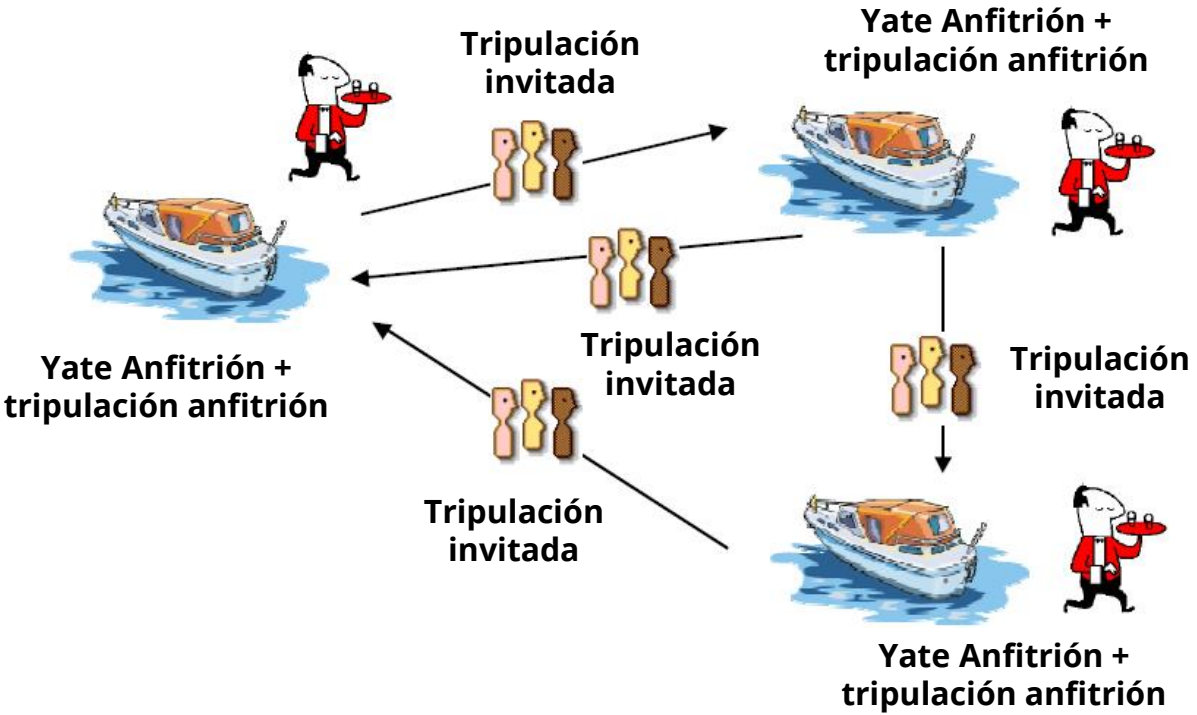
Progressive Party Problem

Backtracking + GBJ
Algoritmo Greedy + TS

Carlos Chesta
Eva Moya

El problema y sus restricciones

Progressive Party Problem



Restricciones

Restricción de capacidad: La capacidad de un yate **NO puede ser excedida.**

Restricción de visitas: Un yate invitado **NO** puede **visitar más de una vez** al mismo yate anfitrión en una misma fiesta.

Restricción de encuentros: Un par de grupos invitados **NO** pueden **encontrarse más de una vez** en una misma fiesta.

Representación

Representación de la solución

Ambos algoritmos

- **Matriz *B x T***

x_{it} : Número del yate anfitrión que visita la tripulación i en el tiempo t

$\forall i \in \{1, ..., B\}$
$\forall t \in \{1, ..., T\}$
$\forall x_{it} \in \{1, ..., H\}$

Diferencia:

- Para **BT+GBJ** *"B"* representa el **número total de yates**
- Para **Greedy + TS** *"B"* representa el **número total de yates invitados**

Espacio de Búsqueda

Ambos algoritmos

$$H^{B \cdot T}$$

Diferencia:

- Para **BT+GBJ** "B" representa el **número total de yates**
- Para **Greedy + TS** "B" representa el **número total de yates invitados**

Espacio de Búsqueda

Instancia: 42 en total, 29 invitados y 13 anfitriones en 6 tiempos

BT + GBJ

$$
\begin{aligned}
H^{B \cdot T} &= 13^{42 \cdot 6} \\
&= 13^{252} \\
&= 5{,}172 \cdot 10^{280}
\end{aligned}
$$

Greedy + TS

$$
\begin{aligned}
H^{B \cdot T} &= 13^{29 \cdot 6} \\
&= 13^{174} \\
&= 6{,}701 \cdot 10^{193}
\end{aligned}
$$

—

Implementación de Algoritmos

Backtraking (Implementado)

Función Asignar_Host (matriz inicial)

```
1:  Buscar fila, columna de una celda vacía
2:
3:  if no hay celda vacía then return True
4:
5:  for cada índice i del vector Host do
6:
7:      PosibleHost = vector Host [i]
8:
9:     if PossibleHost satisface restricciones en celda fila,columna
10:
11:     then asignar en celda y probar recursivamente en las demás
12:
13:    if recursión exitosa then return True
14:    else asignar ceda como vacía e intentar de nuevo
15:
16: end for
17:
18: if Ningún host dio resultado, return False
```

Backtraking + GBJ (No Implementado)

Función Asignar_Host (matriz inicial)

```
1: int h, i, jump
2: Buscar fila, columna de una celda vacía
3:
4: if no hay celda vacía then return True
5:
6: for cada índice i del vector Host do
7:
8:       PosibleHost = vector Host [i]
9:
10:     if PossibleHost satisface restricciones en celda fila,columna
11:
12:      then jump = asignar en celda y probar recursivamente en las demás
13: Stateif jump != matriz
14:
15:      then return jump
16:
17: end for
18:
19: merge(P,padres(matriz))
20: h = max(P)
21: delete (h,P)
22: if Ningún host dio resultado, Return (h)
```

**A theorical Evaluation of Slected Backtraking Algorithms de Grzegorz

Greedy

Algoritmo 2 Algoritmo Greedy

Entrada: Vectores de anfitriones e invitados y problema.
Salida: Solución inicial.

```
1:  Matriz ← Matriz vacía.
2:  para g ∈ Invitado hacer
3:      para t ∈ Tiempo hacer
4:          para h ∈ Anfitrión hacer
5:              x_{g,t} ← h
6:              si obtenerPenalización(matriz) es baja entonces
7:                  Se acepta la asignación de h a x_{g,t}
8:              si no
9:                  Se rechaza la asignación de h a x_{g,t}
10:             fin si
11:         fin para
12:     fin para
13: fin para
14: devolver Matriz
```

Tabú Search

Algoritmo 3 Algoritmo Tabú Search

Entrada: Vectores de anfitriones e invitados, problema, iteraciones, largo lista tabú, solución inicial (s_c).
Salida: Mejor solución encontrada.

```
1:  listaTabu← lista vacía
2:  s_best ← s_c
3:  i ← 0
4:  mientras i <iteraciones hacer
5:      s_v ← Seleccionar mejor solución vecina no tabú
6:      s_c ← s_v
7:      Actualizar lista tabú
8:      si s_c es mejor que s_best entonces
9:          s_best ← s_c
10:     fin si
11:     i ← i + 1
12: fin mientras
13: devolver s_best
```

Experimentación

Backtracking

Instancia	Periodos	#Yates	#Host	Datos de los yates	Tiempo ejecución
Propia 1	3	6	3	6,2;8,2;12,2;12,2;12,4;12,4	0,01467 seg.
Propia 2	4	10	5	6,2;8,2;12,2;12,2;12,4;12,4;12,4;10,1;10,2;10,2	10,0036 seg.
Propia 3	3	6	4	8,3;12,6;8,2;8,2;8,4;8,2;8,2;5,2;4,1;8,4	1,28213 seg.
PPP	6	42	13	6,2;8,2;12,2;12,2;12,4;12,4; 12,4;10,1;10,2;10,2;10,2; 10,3;8,4;8,2;8,3;12,6;8,2; 8,2;8,4;8,2;8,4;8,5;7,4;7,4; 7,2;7,2;7,4;7,5;6,2;6,4;6,2; 6,2;6,2;6,2;6,2;6,2;6,4;6,5; 9,7;0,2;0,3;0,4	113+ Seg.

Cuadro 4: Tabla de datos experimentales

Greedy + TS

Instancia	Yates totales	Tiempos totales	Anfitriones	Tiempo de ejecución [s]	Penalizaciones*
PPP_m1	6	3	3	0.000766	0
PPP_m2	10	4	5	0.004238	0
PPP1	42	6	13	113.553	6
PPP2	42	6	13	107.672	7
PPP3	42	6	13	121.209	12
PPP4	42	6	13	103.768	9
PPP5	42	6	13	127.998	7
PPP6	42	6	14	133.369	8

Parámetros TS: Largo lista tabú de 50 y 200 iteraciones

* Cada restricción que no se cumple en la matriz aumenta en 1 el contador de Penalizaciones

Experimentación Conjunta

Backtracking

Archivos	# Periodos	# Total Yates	# Yates Host	Tiempo de ejecución [s]
PPP_m1	3	6	3	0,001998
PPP_m2	4	10	5	0,005258
PPP	6	42	13	113+

Cuadro 5: Tabla de datos experimentales

Greedy + TS

Archivos	# Periodos	# Total Yates	# Yates Host	Tiempo de ejecución [s]	Penalizaciones*
PPP_m1	3	6	3	0.000766	0
PPP_m2	4	10	5	0.004238	0
PPP	42	6	13	113.553	6

*** Cada restricción que no se cumple en la matriz aumenta en 1 el contador de Penalizaciones**
Parámetros TS: Largo lista tabú de 50 y 200 iteraciones

Conclusiones

Greedy + TS es **levemente más rápido** que solo Backtracking en instancias pequeñas

Espacio de búsqueda es menor al ignorar yates Host en matriz

Representación y movimientos **adecuados**

El **tiempo de ejecución aumenta** exponencialmente según el tamaño del espacio de búsqueda

La **complejidad** de la técnica de resolución no implica mejores resultados