Inteligencia Artificial

Informe Final: Problema Progressive Party Problem

Carlos Chesta Rivas

24 de junio de 2016

Evaluación

Mejoras 1ra Entrega (10 %):	_____
Código Fuente (10 %):	_____
Representación (15 %):	_____
Descripción del algoritmo (20 %):	_____
Experimentos (10 %):	_____
Resultados (10 %):	_____
Conclusiones (20 %):	_____
Bibliografía (5 %):	_____
Nota Final (100):	_____

Resumen

Progressive Party Problem o PPP es un famoso problema de satisfacción de restricciones que consiste en cómo organizar una fiesta en un club de yates, donde hay dos categorías posibles de yates: yates anfitriones y yates invitados. La idea a grandes rasgos es que las tripulaciones de los yates invitados como unidad pueden visitar a diferentes yates anfitriones en cada periodo, mientras que la tripulación de un yate anfitrión debe quedarse en su lugar para atender a los invitados. La función objetivo es minimizar la cantidad de yates anfitriones, dado el costo que conlleva ser un anfitrión (comida, iluminación, etc). Este documento presenta distintas formulaciones y enfoques del problema así como sus ventajas y desventajas en la búsqueda de soluciones y los tiempos asociados a cada una de ellas.

1. Introducción

Organizar una fiesta progresiva no es una tarea fácil, dada la cantidad de anfitriones e invitados. Encontrar la cantidad óptima de anfitriones que satisfagan a todos los invitados de igual manera puede volverse una compleja tarea. En esta ocasión, se plantea el conocido problema *Progressive Party Problem* o conocido como PPP, por sus siglas en inglés, que consiste en una fiesta de un club de yates, el cual se identifican dos grupos: los yates que son anfitriones y los yates que son invitados. El problema natural de un organizador de este tipo de eventos es encontrar la cantidad adecuada de anfitriones para que los invitados sean atendidos de buena manera, considerando que no haya holgura de tiempo tanto para invitados sin estar siendo atendidos por un anfitrión como anfitriones sin estar atendiendo invitados.

Este tipo de problemas no sólo se limita a un club de yates. Puede aplicarse perfectamente, por ejemplo, a fiestas costumbristas donde hay una cierta cantidad de stands y otra cierta cantidad de público asistente que quiere conocer cada stand, como el caso de *Oktoberfest* que se celebra en Malloco, donde hay una cantidad fija (a determinar) de stands, cada uno con su producto, y otra cantidad conocida de asistentes, por lo que los organizadores deben encontrar el número de stands que permita proveer a los asistentes y a la vez minimizar los costos que conlleva cada uno.

Este paper está compuesto por la Definición del Problema donde se explica el problema a fondo; el Estado del Arte donde se aborda en qué está la literatura actualmente con respecto a este problema, análisis de algoritmos que buscan una solución, etc; el Modelo Matemático donde se plantea matemáticamente la formulación de PPP junto con sus respectivos espacios de búsqueda además de las restricciones características asociadas para llevar a cabo encontrar la(s) solución(es); finalmente las Conclusiones donde se concluyen los distintos algoritmos, dando un pequeño análisis al respecto, además de plantear algunos lineamientos para investigaciones futuras. El propósito del presente informe es presentar el problema en detalle y mostrar la representación, algoritmos usados, experimentos y resultados de la implementación empírica del PPP usando como algoritmos *Greedy + Tabu Search*

2. Definición del Problema

2.1. Origen del Problema

PPP fue planteado por el matemático Peter M. Hubbard[7], miembro del Departamento de Matemática de la Universidad de Southampton en el año 1995 que también es miembro de la Sea Wych Owners Association. El problema original plantea un caso en que en un club de yates hay una fiesta.

2.2. Explicación del Problema

El problema es el siguiente. Existen un total de n yates, cada uno con sus respectivas tripulaciones. Cada yate tiene una capacidad máxima de tripulación determinada, y también la cantidad de dicha tripulación es conocida de antemano. La idea es que miembros del club sociabilicen lo más posible, para eso se dividen en yates anfitriones y yates invitados. Las tripulaciones de los yates invitados visitan en turnos a yates anfitriones, es decir, los invitados pueden estar en un yate anfitrión por una cierta cantidad de tiempo, luego rotan a otro yate anfitrión. Mientras que estos últimos no pueden moverse de su yate ya que deben atender a los invitados. El número de rondas en que se mueven las tripulaciones invitadas es conocida.

El objetivo es encontrar un número mínimo de yates anfitriones que puedan satisfacer a la totalidad de tripulaciones invitadas, dado el costo que conlleva ser un yate anfitrión[7].

Como supuesto se considera que en todo momento la tripulación de cierto yate permanece unida, esto es, no se separan en ningún momento. También se tiene que una tripulación invitada no puede revisitar a un yate anfitrión. Dado lo anterior, se cumple que dos tripulaciones no pueden juntarse más de una vez. Del total de yates, hay una cantidad fija de barcos que serán designados como anfitriones por lo que los restantes serán los yates invitados.

2.3. Posibles variantes

También es posible definir ciertas variaciones del problema original añadiendo ciertas restricciones que se dan en el mundo real[4][7] tal como que se definen los primeros m, con $m < n$, botes como anfitriones compuestos por el organizador y otros $(m-1)$ compuestos por padres con sus hijos, pero los padres se quedan en los yates atendiendo a los invitados mientras que los hijos

se separan visitando otros yates por lo que dicho conjunto de tripulación se puede considerar como un yate invitado virtual de capacidad cero.

2.4. Problemas Relacionados

Dado que *Progressive Party Problem* es catalogado un problema combinatorio[3] del tipo de asignación, existe una serie de otros problemas conocidos en la literatura que también son catalogados de este tipo. Un clásico problema es el Nure Scheduling Problem que consiste en asignar una mínima cantidad de enfermeras tomando en cuenta los turnos que disponen y las vacaciones que poseen pero que puedan satisfacer todas las necesidades del hospital en que trabajen, de modo que no hayan enfermeras en modo ocioso ni tampoco que falten enfermeras.

3. Estado del Arte

3.1. Origen del problema

El planteamiento de este problema remonta en el año 1995 por P. Hubbard, pero formalizada su investigación en el paper de B. Smith et al [7]. En éste, Smith plantea dos posibles enfoques para resolver este problema: con Programación Lineal Entera (ILP, por sus siglas en inglés) y por el método de Satisfacción de Restricciones (CSP, por sus siglas en inglés).

En esta investigación se toma en cuenta el caso de prueba de 39 yates en total, con una tripulación que varía entre 1 y 7 pasajeros por yate, los cuales los yates invitados rotan entre los yates anfitriones en una cantidad de rondas de $T = 6$ con 30 minutos por ronda. Además, se considera que el primer yate corresponde al organizador con una capacidad de 6 personas, luego dos yates más que su tripulación son padres con niños, por lo que los padres se quedan en sus yates atendiendo a invitados, mientras que los niños se separan de sus padres convirtiéndose así en 3 yates virtuales con capacidad de cero. Por lo tanto, para el análisis se considera un total de 42 yates.

3.2. Resolución

P. Hubbard plantea el problema inicialmente usando IPL[7]. Define dos variables binarias: $\delta_i = 1$ ssi el yate i es anfitrión y $\gamma_{ikt} = 1$ si el yate k es invitado del yate i en el periodo t. La función objetivo es minimizar la cantidad de yates anfitriones.

Usando este primer enfoque se tiene que con 12 yates anfitriones es posible atender al resto de los 30 invitados. Sin embargo, no fue posible determinar si para 13 yates anfitriones también corresponde a una solución. Para ello se planteó como heurística ordenar en forma descendiente los yates según su capacidad máxima. De esta forma se obtiene que es posible obtener 13 yates anfitriones, siendo los primeros 13 yates del total los que les corresponde el papel de anfitrión.

También se encontró una solución con 14 anfitriones pero se tuvo que relajar algunas restricciones y especificar que los primeros 14 yates tuvieran que ser anfitriones. Para este caso la solución fue encontrada en 598 segundos. No obstante, para soluciones mayores a 14 yates fue imposible determinar una solución ya que luego de 189 horas en ejecución tuvo que abortarse.

Luego, a modo de comparación, Smith plantea otro enfoque para abordar el problema: La resolución mediante CSP, el cual fue implementado usando ILOG Solver[1], que es una herramienta de programación de restricciones usando librerías en C++. Esta herramienta procesa el problema usando un backtracking simple aunque también es posible realizarlo usando *full lookahead*. H.P. Williams[9] llega a conclusiones similares que el de Smith. Señala que el modelo de programación lineal entera es muy largo de resolver, en cambio con programación de restricciones se resuelve más rápido.

Una ventaja de este enfoque fue que la formulación como CSP fue mucho más compacta ya que el modelado queda directamente centrado en sus restricciones por lo que es más fácil abordar

soluciones que satisfagan estas restricciones. La idea es que usando la técnica de *full lookahead* las restricciones son usadas para identificar los efectos de futuras instancias de las variables y si existe un valor en el dominio de una futura variable que entre en conflicto con la asignación actual es temporalmente eliminada. Bajo esta premisa es posible reducir los tiempos. Producto de lo señalado en lo anterior, es de suma importancia definir heurísticas que permitan un mejor filtrado de instancias que se sabrán que no serán parte de las soluciones. En este caso conviene usar una basado en el principio "fail first"(fallo primero), el cual consiste en elegir una variable que sea difícil de asignar, esto es, escogiendo variables que tengan un dominio pequeño[7]. Este modelo está detallado en la sección 4.3.

Smith[7] señala que el problema fue testeado usando CSP, pero usando versiones más pequeñas del problema, es decir, con un número reducido de yates y anfitriones. Señala que, usando una IPX SPARCstation, pudo resolverlo bastante rápido (con una cantidad considerable de pequeños errores, fáciles de corregir usando backtracking). No obstante, usando la versión completa del problema no fue posible resolverlo usando CSP. Para ello plantea una nueva heurística que es analizar de forma separada cada periodo de tiempo $t \in T$ del problema, pero almacenando las soluciones internas de cada periodo. Como resultado obtuvo las soluciones bastante rápido. Es por esto que concluye que usando CSP por sí solo no tiene utilidad alguna ya que no encuentra soluciones al problema general, pero si se interviene manualmente entonces el rendimiento aumenta considerablemente con respecto a la búsqueda de soluciones usando ILP.

J. N. Hooker[3] propone la resolución del problema usando Programación Lineal/Lógica Mixta (MLLP, por sus siglas en inglés) y hace la comparación con el modelo de Programación Entera Lineal Mixta (MILP, por sus siglas en inglés) señalado por [4]. La idea es aprovechar el enfoque que posee MLLP de optimización de problemas que posean elementos discretos y continuos. MLLP toma como base la premisa hecha por MILP pero además agregando enfoques del tipo lógicas como implicancias, entre otros.

Usando una versión reducida del problema, Hooker en sus resultados señala que resolviendo el problema con el enfoque MLLP obtiene resultados muy buenos en comparación a la resolución del problema como MILP. A modo de resumen se presenta la siguiente tabla con sus resultados[3].

		Segundos	
Botes	Periodos	MLLP	MILP
5	2	0.006	0.173
6	3	0.007	0.445
7	3	0.007	1.014
8	4	0.008	1.525
10	4	0.011	2.161

Cuadro 1: Comparación de tiempos de computación para PPP reducido usando MILP y MLLP

Tanto Hooker[3] como G. Ottosson[6] señalan que usando la técnica de MLLP en el problema PPP se obtienen resultados grandiosos.

J. Walser [8] y P. Galinier[2] proponen la resolución del PPP no usando técnincas de resolución de búsqueda completa, sino que proponen resolverlo usando técnicas de búsqueda local con algoritmos como *Tabu Search*, *Algoritmo de Metrópolis* que es una versión simplificada del algoritmo *Simulated Annealing*[5]. En sus resultados señalan que este método de resolución es bastante acertiva ya que usando simplemente *Tabu Search* pudieron resolver el problema hasta con $T = 9$ periodos de tiempo en tan solo del orden de segundos. Por lo que señalan que usando búsqueda local podría ser considerado una de las mejores formas para obtener resultados en PPP en poco tiempo[2].

4. Modelo Matemático

4.1. Programación Lineal Entera

El modelo planteado por Peter Hurbbard[7] fue el primer modelo en abordar este tipo de problemas. Se tiene lo siguiente:

Variables

$$\delta_i = \begin{cases} 1 & \text{si el yate } i \text{ es anfitrión} \\ 0 & \text{en otro caso} \end{cases} \qquad \forall i = 1..n$$

$$\gamma_{ikt} = \begin{cases} 1 & \text{si el yate } k \text{ es invitado del yate } i \text{ en el periodo t} \\ 0 & \text{en otro caso} \end{cases} \qquad \forall i = 1..n, \forall k = 1..n, \forall t = 1..T$$

Parámetros

B**:** número total de yates

c_i**:** tripulación del yate i $\quad \forall i = 1..n$

K_i**:** capacidad total del yate i $\quad \forall i = 1..n$

T**:** Cantidad de rondas

Función Objetivo

$$\text{Min} \sum_i \delta_i \qquad \text{Minimizar la cantidad de yates anfitriones.}$$

Restricciones

Un bote puede sólo puede ser visitado si es anfitrión (CD)

$$\gamma_{ikt} - \delta_i \leq 0 \qquad \forall i, k, t; i \neq k$$

La capacidad de los yates anfitriones no puede ser excedida (CCAP)

$$\sum_{k, k \neq i} c_k \gamma_{ikt} \leq K_i - c_i \qquad \forall i, t$$

Cada tripulación debe ser anfitrión o invitado (GA)

$$\delta_k + \sum_{i, \neq k} \gamma_{ikt} = 1 \qquad \forall k, t$$

Una tripulación invitada no puede visitar a un anfitrión más de una vez (GB)

$$\sum_i \gamma_{ijt} \leq 1 \qquad \forall i, k; i \neq k$$

Cualquier par de tripulación pueden juntarse a lo más una vez (W)

$$\gamma_{ikt} + \gamma_{ilt} + \gamma_{jks} + \gamma_{jls} \leq 3 \qquad \forall i, j, k, l, t, s$$
$$i \neq j; i \neq k; k < l; i \neq l; k \neq j; j \neq l; s \neq t$$

Esta primera instancia de la formulación como ILP posee un espacio de búsqueda de 2^{B^2T}

Sin embargo, debido a la dificultad de resolver dicho modelo, Smith et al.[7] propuso una reformulación del problema agregando una nueva variable y modificando la restricción W en tres nuevas restricciones: S, V, Y. Para ello formuló lo siguiente:

Variables

$$\delta_i = \begin{cases} 1 & \text{si el yate } i \text{ es anfitrión} \\ 0 & \text{en otro caso} \end{cases} \qquad \forall i = 1..n$$

$$\gamma_{ikt} = \begin{cases} 1 & \text{si el yate } k \text{ es invitado del yate } i \text{ en el periodo t} \\ 0 & \text{en otro caso} \end{cases} \qquad \forall i = 1..n, \forall k = 1..n, \forall t = 1..T$$

$$x_{iklt} = \begin{cases} 1 & \text{si las tripulaciones } k \text{ y } l \text{ se juntan en el yate } i \text{ en el periodo t} \\ 0 & \text{en otro caso} \end{cases}$$

Parámetros

B: número total de yates

c_i: tripulación del yate i $\quad \forall i = 1..n$

K_i: capacidad total del yate i $\quad \forall i = 1..n$

T: Cantidad de rondas

Función Objetivo

$$\text{Min} \sum_{i \in I} \delta_i \qquad \text{Minimizar la cantidad de yates anfitriones.}$$

Restricciones

Un bote puede sólo puede ser visitado si es anfitrión (CD)

$$\gamma_{ikt} - \delta_i \leq 0 \qquad \forall i, k, t; i \neq k$$

La capacidad de los yates anfitriones no puede ser excedida (CCAP)

$$\sum_{k, k \neq i} c_k \gamma_{ikt} \leq K_i - c_i \qquad \forall i, t$$

Cada tripulación debe ser anfitrión o invitado (GA)

$$\delta_k + \sum_{i, \neq k} \gamma_{ikt} = 1 \qquad \forall k, t$$

Una tripulación invitada no puede visitar a un anfitrión más de una vez (GB)

$$\sum_{i} \gamma_{ijt} \leq 1 \qquad \forall i, k; i \neq k$$

Cualquier par de tripulación pueden juntarse a lo más una vez (S)

$$2x_{iklt} - \gamma_{ikt} - \gamma_{ilt} \leq 0 \qquad \forall i, k, l, t; k < l; i \neq k; i \neq l$$

Cualquier par de tripulación pueden juntarse a lo más una vez (V)

$$\gamma_{ikt} + \gamma_{ilt} - x_{iklt} \leq 1 \qquad \forall i, k, l, t; k < l; i \neq k; i \neq l$$

Cualquier par de tripulación pueden juntarse a lo más una vez (Y)

$$\sum_{t} \sum_{l, l>k} x_{iklt} \leq 1 \qquad \forall i, k$$

Con este nuevo planteamiento de ILP, se tiene un espacio de búsqueda de 2^{B^2T}, por lo que se mantiene con respecto al modelo planteado inicialmente. Sin embargo, con esta instancia del problema se reduce la complejidad total de número de filas de $O(B^4T^2)$ a $O(B^3T)$, lo que sin duda es una reducción considerable.

4.2. Programación Lineal/Lógica Mixta

Años después, en 1999, J. N. Hooker[3] propuso una formulación basada en Programación Lineal/Lógica Mixta (MLLP, por sus siglas en inglés).

Variables

$$z_i = \begin{cases} 1 & \text{si el yate } i \text{ es anfitrión} \\ 0 & \text{en otro caso} \end{cases} \qquad \forall i = 1..n \text{Yate anfitrión}$$

$$\delta_i = \begin{cases} 1 & \text{si el yate } i \text{ es anfitrión} \\ 0 & \text{en otro caso} \end{cases} \qquad \forall i = 1..n$$

h_{it} = Yate anfitrión que está siendo visitado por la tripulación i en el periodo t

$$\forall i \in I, t = 0..T, h_{it} \in I$$

$$m_{ijt} = \begin{cases} 1 & \text{si las tripulaciones } i \text{ y } j \text{ se visitan en el periodo } t \\ 0 & \text{en otro caso} \end{cases} \qquad \forall i, j \in I, t = 0..T$$

Parámetros

I: Conjunto de yates

c_i: tripulación del yate i $\quad \forall i = 1..n$

K_i: capacidad total del yate i $\quad \forall i = 1..n$

T: Cantidad de rondas

Función Objetivo

$$\text{Min} \sum_{i \in I} z_i \qquad \text{Minimizar la cantidad de yates anfitriones.}$$

Sujeto a

Definición de v_{ijt}

$$v_{ijt} \equiv (h_{ij} = j), i, j \in I, t \in T$$

Las tripulaciones invitadas deben visitar un diferente anfitrión en cada periodo de tiempo

$$\delta_i \vee h_{i1} \neq h_{i2} \neq ... \neq h_{i|T|}, i \in I$$

Una tripulación debe permanecer en su yate ssi es anfitrión

$$\delta_i \equiv (h_{it} = i), i \in I, t \in T$$

Las tripulaciones a bordo deben ser menor que la capacidad del yate

$$\sum_{i \in I, i \neq j} c_i v_{ijt} \leq K_j - c_j, i, j \in I, t \in T$$

Si tripulaciones i y j están ambas visitando tripulaciones, entonces $m_{ijt} = 1$ o $h_{it} \neq h_{jt}$

$$\delta_i \vee \delta_j \vee m_{ijt} \vee (h_{it} \neq h_{jt}); i, j \in I, i < j, t \in T$$

Un par de tripulaciones no pueden juntarse más de una vez

$$\sum_{t \in T} m_{ijt} \leq 1; i, j \in I, i < j, h_{ij} \in \{1, ..., \mid I \mid\}$$

Esta formulación con MLLP posee un espacio de búsqueda de $2^{I^2 T}$. Esto se obtiene debido a que en la variable m_{ijt} posee dos posibles valores para cada instancia de I (producto de i), nuevamente I (producto de j) y de T (producto de t).

4.3. CSP

Smith et.al[7] planteó el mismo problema, pero como CSP que busca encontrar soluciones que satisfagan las restricciones usando el siguiente modelo matemático, con el fin de que quedase mucho más compacto

Variables

h_{it} = Yate anfitrión que está siendo visitado por la tripulación i en el periodo t $\qquad h_{it} \in \{1, ..., H\}$

$$v_{ijt} = \begin{cases} 1 & \text{si el yate invitado } i \text{ visita al anfitrión } j \text{ en periodo } t \\ 0 & \text{en otro caso} \end{cases} \qquad i \in G, j \in H, t = 0..T$$

$$m_{klt} = \begin{cases} 1 & \text{si la tripulación } k \text{ y } l \text{ se conocen en periodo } t \\ 0 & \text{en otro caso} \end{cases} \qquad (k \in G, l \in H) \vee (l \in G, k \in H), t = 0..T$$

Parámetros

H**:** Conjunto de yates anfitriones

G**:** Conjunto de yates invitados

T**:** Cantidad de rondas o periodos

c_i**:** Tamaño de la tripulación i

C_j**:** Capacidad del anfitrión j

Sujeto a

Tripulaciones invitadas no pueden visitar más de una vez a un anfitrión

$$h_{i1}, hi2, ..., h_{iT} \text{ son todos diferentes } \forall i$$

Una tripulación invitada debe ser atendida siempre por un anfitrión

Satisfecha implícitamente por la definición de variables.

Una tripulación invitada sólo puede ser asignada a un anfitrión por periodo

Satisfecha implícitamente por la definición de variables.

La capacidad del yate no puede ser excedida

$$\sum_i c_i v_{ijt} \leq C_j \qquad \forall j, t$$

Un par de tripulaciones no pueden juntarse dos veces

$$\sum_i m_{klt} \leq 1 \forall k, l; k < l$$

Este método da un espacio de búsqueda de 2^{GHT} ya que para G, H, T valores hay dos posibles opciones (0 y 1).

5. Representación

Para la implementación del *Progressive Party Problem* se escogió usar una formulación como un problema de satisfacción de restricciones (CSP) para ello como representación escogida es una matriz de enteros.

Sea G la cantidad de yates anfitriones y T el número de periodos. Se tiene una matriz GxT, donde cada elemento de dicha matriz $x_{g,t}$ es un entero de 1 a H -con H la cantidad de anfitriones- y representa al yate anfitrión que atiende a la tripulación del yate g en el tiempo t, con $1 \leq g \leq G, 1 \leq t \leq T$. Un ejemplo de esta representación es la siguiente:

	T_1	T_2
G_1	4	5
G_2	5	4
G_3	6	4

Cuadro 2: Ejemplo de representación.

En el cuadro 2 se tiene que en el tiempo T_1 la tripulación del yate G_1 está en el yate anfitrión 4.

Dado que es un problema de satisfacción de restricciones tiene que haber una función de costo que determine qué solución es mejor que otra. Para ello la función de costo para este caso se define como el número de restricciones incumplidas. Para cada restricción C se define una penalización $f_C(s)$ que toma como valor 0 o 1 si la solución s cumple la restricción C o no, respectivamente. Por lo tanto, la solución óptima s_{best} es aquella en que su solución tenga una función de costo:

$$\sum_{C} f_C(s_{best}) = 0 \tag{1}$$

El conjunto C queda definido por las tres siguientes restricciones:

1. Restricción 1: Relacionado con los movimientos de los invitados a diferentes anfitriones en cada periodo de tiempo. Para cada g $(1 \leq g \leq G)$ se debe cumplir que:
$$x_{g,1} \neq x_{g,2} \neq x_{g,1} \neq ... \neq x_{g,T} \tag{2}$$
2. Restricción 2: Relacionado con que para cada par de tripulaciones invitadas sólo pueden juntarse a lo más una vez. Para cada par (g_i, g_j) con $1 \leq i, j \leq G$ y $i \neq j$:
$$|\{t, 1 \leq t \leq T | x_{g_1,t} = x_{g_2,t}| \leq 1 \tag{3}$$
3. Restricción 3: La capacidad de los yates debe respetarse. Sea $c(y)$ la cantidad de personas de la tripulación y y $K(h)$ la capacidad del yate anfitrión h, debe cumplirse que:
$$\sum_{1 \leq g \leq G, x_{g,t}=h} c(g) \leq K(h) \tag{4}$$

Por lo tanto, se tiene que para cada una de las restricciones las penalizaciones son las siguientes.

1. Restricción 1: Se define como la cantidad de pares de (x_{g,t_i}, x_{g,t_j}) que tengan el mismo valor.
2. Restricción 2: Como cada par de tripulaciones invitadas pueden juntarse a lo más una vez, se define la penalización como la cantidad total de veces que se han juntado menos uno (que es aquella permitida).
$$f_C(s) = \sum (x_{g_1,t} = x_{g_2,t}) - 1 \tag{5}$$
3. Restricción 3: Sea
$$\sigma_{h,t}(s) = \begin{cases} 1 & \text{si } K(h) - \sum_{1 \leq g \leq G, x_{g,t}=h} c(g) < 0 \\ 0 & \text{en otro caso} \end{cases} \tag{6}$$
Por lo tanto, la penalización para esta restricción está definida como:
$$\sum_{s} \sigma_{h,t}(s) \tag{7}$$

Finalmente, la función de costo estará constituida como la suma de penalizaciones totales para una solución s del conjunto de soluciones S. Esto es:

$$\forall s \in S, f(s) = \sum_{i} f_{C_i}(s) \tag{8}$$

El movimiento elegido consiste en cambiar el valor del yate anfitrión para una tripulación invitada dada en un tiempo dado. Esto es, cambiar el valor $x_{g,t}$ por otro $v_{g,t}$ que pertenezca al conjunto de yates anfitriones.

6. Descripción del algoritmo

Los algoritmos asignados para la implementación fueron *Greedy + Tabu Search*. Para ello se lee el archivo de texto de la instancia del problema y, dependiendo del tipo de instancia, se generan dos vectores: uno con los yates anfitriones y otro con los yates invitados. El contenido del vector de yates anfitriones depende exclusivamente del tipo de instancia que se desarrollará: para instancias CSPLib se escogió a aquellos yates que tengan la mayor capacidad como anfitriones, mientras que para instancias no CSPLib se lee el archivo de configuración.

Luego, se usa el algoritmo constructivo *Greedy* para generar una solución inicial con un buen primer acercamiento a un óptimo, ya que este algoritmo va ingresando valores tales que se genera el mayor beneficio posible (en este caso busca minimizar las penalizaciones). Luego de obtener la matriz solución inicial se aplica el algoritmo reparativo *Tabu Search* con el fin de explotar la solución hasta encontrar aquella solución que minimice la función de costo, siendo la solución global con función de costo 0. Finalmente, una vez terminado de iterar en *Tabu Search* se escribe el archivo de salida correspondiente.

Para instancias CSPLib, además de escoger los yates anfitriones con la heurística de la mayor capacidad, se vuelve a iterar hasta encontrar una solución global (penalizaciones igual a 0) pero esta vez con una cantidad de yates anfitriones aumentado en uno.

En resumen, el pseudocódigo del algoritmo implementado es el siguiente:

Algoritmo 1 Resolución de PPP usando Greedy + Tabu Search

Entrada: Nombre de la instancia, cantidad de iteraciones y largo lista tabú.
Salida: Archivo generado con la mejor solución encontrada.

```
1: si problema es una instancia de CSPLib entonces
2:    δ ← 0
3:    mientras encontroOptimo = falso  hacer
4:       problema ← Leer archivo de la instancia CSPLib.
5:       Cantidad Anfitriones ← cantidadAnfitriones(problema) + δ.
6:       Generar vectores de anfitriones y invitados.
7:       Solución Inicial ← Greedy(vector anfitrion, vector invitado, problema)
8:       Solución Final ← Tabu Search(vectores anfitrion e invitado, problema, Largo lista Tabu,
         Iteraciones)
9:       si cantidad de penalizaciones ≠ 0 entonces
10:         δ ← δ + 1
11:         ir a 3.
12:      si no
13:         Escribir Archivo de salida
14:         encontroOptimo = cierto
15:      fin si
16:   fin mientras
17: fin si
18: si problema es el PPP completo entonces
19:   problema ← Leer archivo PPP.
20:   Generar vectores de anfitriones y invitados a partir del archivo de configuración
21:   Solución Inicial ← Greedy(vector anfitrion, vector invitado, problema)
22:   Solución Final ← Tabu Search(vectores anfitrion e invitado, problema, Largo lista Tabu,
      Iteraciones)
23:   Escribir Archivo de salida
24: fin si
```

Dentro del algoritmo *Greedy* se tiene lo siguiente en grandes rasgos.

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
```

Por otro lado, se tiene el algoritmo *Tabu Search* como:

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 leftarrow lista vacía
2: s_best leftarrow 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
```

Para el caso de instancias CSPLib, se escogió como heurística que los yates anfitriones elegidos serán aquellos que tengan la mayor capacidad.

7. Experimentos

La implementación fue realizada dentro de una máquina virtual del programa Virtual Box con sistema operativo Ubuntu con una capacidad de procesamiento de hasta el 100 % de la máquina anfitriona (Intel i7-5500U de 2.40GHz) usando hasta dos núcleos de CPU, con una memoria RAM asignada de 5046 MB. El código elegido para la implementación de resultados fue C++. Además los parámetros de todas las funciones utilizadas dentro del código fueron pasadas por valor (copia directa de las variables) y no por referencia (punteros directos a los espacios de memoria de éstos).

Todas las pruebas fueron hechas con los mismos parámetros para una homogeneidad y fiabilidad de los resultados, esto es que para cada instancia fue hecha con 200 iteraciones y un largo de lista de tabú de 50 elementos para el algoritmo de *Tabú Search*.

8. Resultados

Para instancias CSPLib los resultados son:

Instancia	Yates totales	Tiempos	Anfitriones óptimos	Tiempo de ejecución [s]	Penalizaciones
Ian01	5	2	3	0.000461	0
Ian02	6	5	5	0.000759	0
Ian03	7	6	6	0.001285	0
Ian04	8	6	6	0.017887	0
Ian05	9	5	5	0.005376	0
Ian06	10	6	6	0.005586	0
Ian07	11	5	6	0.008105	0
Ian08	12	5	6	0.013053	0
Ian09	13	5	6	0.0142	0
Ian10	14	5	5	0.033553	0

Cuadro 3: Resultados para instancias CSPLib

Del cuadro 3 se observa que para todos ellos se encuentra el óptimo global (con 0 penalizaciones), y para la mayoría la cantidad óptima de yates anfitriones es la misma cantidad de tiempos totales asignados para dicha instancia, salvo en los casos de las instancias CSPLib Ian01, CSPLib Ian07, CSPLib Ian08 y CSPLib Ian09, donde el óptimo global fue encontrado para una cantidad superior de yates (específicamente un yate más que la cantidad de tiempo asignado).

Un ejemplo del archivo resultante de una instancia CSPLib, específicamente el CSPLib Ian01, es:

```
T = 1
----------------------
1 -> 3
2 -> 3
3 -> A [2+2/ 12]
4 -> A
5 -> A

T = 2
----------------------
1 -> 4
2 -> 5
3 -> A
4 -> A [2/ 12]
5 -> A [2/ 12]

Botes anfitriones óptimos: 3.
Tiempo de ejecución: 0.000461 [s].
Penalizaciones: 0.
```

Luego, para instancias PPP, es decir, para el problema completo y reducido donde los anfitriones se conocen de antemano se tienen los siguientes resultados:

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

Cuadro 4: Resultados para instancias PPP

Observar que para las dos primera instancias se llegó a soluciones globales ya que su penalización es de 0. No obstante, estas instancias son versiones bien reducidas del problema por lo que el cómputo de éstas fue bastante rápido de encontrar.

Sin embargo, para el problema completo de 42 yates totales no se pudo llegar a un óptimo global ya que de las 6 posibles configuraciones de yates anfitriones cuentan con penalizaciones que van desde 6 a 12 penalizaciones. Además hay que notar que el tiempo de ejecución subió bastante en relación a las demás instancias incluso superando los dos minutos de cómputo por configuración.

Como medida para mejorar los resultados para las instancias $PPPi$ se propone usar como primera medida la heurística de asignación de yates por capacidad, y, en caso de que esto siga dando resultados poco favorables, aumentar las iteraciones y que dinámicamente se aumente en uno la cantidad de yates anfitriones hasta que el resultado tenga 0 penalizaciones totales, como se hace en el caso de instancias CSPLib. Finalmente una buena medida también es variar el tamaño dinámicamente de la lista tabú: Se comienza con una lista larga para explorar más que se puede ir reduciendo de modo de explorar menos y enfocarse en la explotación.

Cabe mencionar nuevamente que todos estos resultados fueron realizados con 200 iteraciones y un largo de 50 en la lista Tabú.

9. Conclusiones

Como se ha señalado anteriormente, para el problema *Progressive Party Problem* existe más de una formulación y enfoque, cada uno con sus ventajas y desventajas, pero todas ellas convergen a los mismos resultados salvo en algunas situaciones. La diferencia recae en el costo tanto de tiempo como de cómputo asociado a esta convergencia ya que como se vio una formulación del tipo Programación Lineal Entera falla si no se tienen los parámetros adecuados. La primera formulación de ILP falla para botes anfitriones mayores que 12, a no ser que se consideren ciertas heurísticas iniciales. La razón de la dificultad de encontrar soluciones a este tipo de modelo es producto de que es demasiado grande para ser resuelto por lo que no hay suficiente tiempo para encontrar soluciones (tan solo para el caso de 15 yates anfitriones se tuvo que abortar la operación luego de 189 horas en ejecución[7]). Es por esto que una buena formulación del modelo es importantísimo para reducir la mayor cantidad de recursos para encontrar soluciones. Otra buena estrategia para abordar este tipo de problemas es definir y analizar posibles heurísticas que se pueden aplicar al problema. Por ejemplo en este caso conviene ordenar en forma descendiente los yates según su capacidad y definir los primeros h yates como anfitriones. Esto también sirve mucho con la formulación del problema del tipo CSP ya que de por sí representa mucho más directamente el problema en comparación a la formulación del tipo ILP.

Una idea a futuro que sería bien interesante es investigar y generar nuevos marcos de modelamiento que sean mezclas entre los existentes. De esta manera poder combinar lo mejor de dos mundos de modelamiento. Si bien hay semejanzas naturales entre, por ejemplo, modelos del tipo

Programación Lineal con modelos del tipo Satisfacción de Restricciones sería interesante que a partir de estos generar un nuevo enfoque que permita encontrar soluciones a un menor costo. Actualmente ya hay tipos de modelamiento que mezclan elementos de uno y de otro, como el caso de MLLP (Programación Lineal/Lógica Mixta) pero eso no significa que no hayan otros por descubrir/inventar.

Como conclusión secundaria es posible decir que problemas del tipo combinatorio, como lo es PPP, tienen una gran utilidad para fabricantes de software de optimización ya que a través de este tipo de problemas donde los modelos son bien extensos y grandes permiten evaluar el rendimiento de diversos programas y hacer una especie de benchmark entre ellos para una misma formulación de un problema.

Finalmente, luego de la realización de la implementación es posible concluir que los resultados obtenidos si bien pueden parecerse unos con otros dependen exclusivamente de la técnica a utilizar ya que cada técnica tiene sus pro y sus contra: unos pueden ser bien rápidos, pero costosos computacionalmente o pesados; mientras que otros pueden no dar buenos resultados pero los arrojan en un tiempo reducido; etc. Se concluye además que el tiempo de ejecución, al menos para *Greedy + Tabu Search* sube exponencialmente a medida que el problema crece, esto es aumentando la cantidad de yates totales. Para instancias con yates que no superan el total de 15 se encontró soluciones bastante rápidos, pero cuando el problema crece a 42 yates el tiempo aumentó considerablemente. El factor del tiempo también depende de la cantidad de iteraciones que se setean los algoritmos que para este caso fue de 200 por lo que para mejores resultados se tiene que aumentar la cantidad de iteraciones a costa del tiempo de ejecución, por lo que habría que preguntarse: ¿Cuánto tiempo se está dispuesto a gastar por encontrar una buena solución?.

10. Bibliografía

Referencias

[1] Jean francois Puget. A c++ implementation of clp. 1994.

[2] Philippe Galinier and Jin-Kao Hao. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, chapter Solving the Progressive Party Problem by Local Search, pages 419–432. Springer US, Boston, MA, 1999.

[3] J.N. Hooker and M.A. Osorio. Mixed logical-linear programming. *Discrete Applied Mathematics*, 96–97:395 – 442, 1999.

[4] Erwin Kalvelagen. On solving the progressive party problem as a {MIP}. *Computers & Operations Research*, 30(11):1713 – 1726, 2003.

[5] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[6] Greger Ottosson. *Integration of constraint programming and integer programming for combinatorial optimization*. Institutionen för informationsteknologi, Univ.[distributör], 1999.

[7] Brailsford Sally C. Hubbard Peter M. Williams H. Paul Smith, Barbara M. The progressive party problem: Integer linear programming and constraint programming compared. *Constraints*, 1(1):119–138, 1996.

[8] Joachim P. Walser. Solving linear pseudo-boolean constraint problems with local search. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence*, AAAI'97/IAAI'97, pages 269–274. AAAI Press, 1997.

[9] H. P. Williams. The elimination of integer variables. *The Journal of the Operational Research Society*, 43(5):387–393, 1992.