Inteligencia Artificial

Estado del Arte: Progressive Party Problem

Victor Andres Roberto Gonzalez Rodriguez

28 de mayo de 2014

Evaluación

Resumen (5 %):	_____
Introducción (5 %):	_____
Definición del Problema (10 %):	_____
Estado del Arte (35 %):	_____
Modelo Matemático (20 %):	_____
Conclusiones (20 %):	_____
Bibliografía (5 %):	_____
Nota Final (100 %):	_____

Resumen

El Progressive Party Problem (PPP), es un complejo problema de optimización combinatorial sujeto a restricciones propuesto en el año 1995. El objetivo del problema consiste en una fiesta de yates, donde los invitados deben ser capaces de recorrer todos los yates anfitriones, cambiandose de yate cada cierto tiempo y cumpliendo ciertas restricciones. Hasta el momento, se han utilizado tres métodos para tratar de resolver el problema: mediante Programación Lineal Entera o Mixta, mediante Programación con Restricciones, y mediante Búsqueda Local (esta última con distintas variaciones). Los resultados al tratar de resolver este problema muestran que la Programación Lineal Entera o Mixta tiene un mal desempeño comparado con la Programación con Restricciones o con Busqueda Local. Finalmente, se presenta un modelo matemático que describe el problema de manera estandar.

1. Introducción

En el presente documento se analizará en detalle el Problema de la Fiesta Progresiva, o Progressive Party Problem (PPP), su historia, el estado del arte y como han evolucionado los métodos resolutivos para este problema.

Antes de continuar, es necesario detallar qué es el Progressive Party Problem. El problema consiste basicamente en lo siguiente: se tiene una fiesta de yates donde cada anfitrión (entiendase el dueño del bote), dispone de su bote de manera que cada asistente a la fiesta pueda pasar por todos los botes, y además, los asistentes van cambiandose de yate cada cierto tiempo (normalmente cada 30 minutos). Todo esto se debe lograr sin superar la capacidad máxima de pasajeros, y sin visitar el mismo bote más de una vez.

En esencia, en esa dinámica se modela el Progressive Party Problem, la cual es un problema combinatorio muy complejo.

En este documento se detallará a cabalidad los detalles del problema, tales como sus variables, las restricciones y el objetivo, de manera que se pueda apreciar de manera estandarizada el problema más allá de las modificaciones que proponen distintos autores. Además, veremos en qué posición se encuentra la ciencia de la computación en la actualidad para resolver este problema. Finalmente, se presentará un modelo matemático que representará todas sus variables, restricciones y cualquier función que se estime necesaria.

Con este documento el lector podrá quedar completamente interiorizado y actualizado del Progressive Party Problem.

2. Definición del Problema

El Progressive Party Problem se formula de la siguiente manera: considérese una fiesta durante una reunión de yates. Existen n botes y sus tripulaciones. Un número de determinado de botes es escogido para ser bote anfitrión: estos serán los botes donde se realizará la fiesta, donde otras tripulaciones visitarán al bote en intervalos de tiempo $t = 1..T$ de media hora cada uno. El número total de periodos T viene dado. Las *tripulaciones visitantes* se mueven de un bote anfitrión a otro, y no pueden visitar un bote más de una vez. Los botes tienen una cierta capacidad limitada de tripulación a bordo, la cual no debe ser superada: esta es la capacidad de visitas de un bote anfitrión. Además, las *tripulaciones visitantes* no pueden encontrarse más de una vez con la misma *tripulación visitante*.

Lo que se busca es una calendarización que minimice la cantidad de botes anfitriones. A partir de lo recopilado en [1, 3, 9, 2], el problema se define[1] como:

2.1. Parámetros

- T (numero de periodos), el cual define cuantos intervalos de tiempos se manejarán en el problema.
- n (cantidad de botes), el cual dice cuantos botes existen disponibles para realizar la fiesta.[2]
- w_i (tamaño de la tripulación del bote), el cual indica cuantos tripulantes quedan fijos en el bote i.
- p_i (capacidad de tripulación del bote), el cual define la capacidad máxima de personas que puede soportar un bote i sin hundirse.

2.2. Objetivos

- Todas las tripulaciones visitantes deben visitar todos los yates.
- Promover que los visitantes tengan la mayor cantidad de interacción social.
- **Minimizar la cantidad de botes anfitriones.**
- Satisfacer las necesidades y restricciones de la fiesta.

[1] En ningun lado existe un modelo estándar de PPP, pero en todas las publicaciones se respeta la formulación hecha por la publicación original [1], la cual es respetada en la definición utilizada en este documento.

[2] Recordar que se busca minimizar la cantidad de botes que se utilizarán finalmente, por lo que es posible que no se utilizen todos los botes disponibles como anfitriones.

Todos estos objetivos serán pulidos y ajustados a un listado de requerimientos que buscan que la fiesta se produzca sin accidentes (respetar limites de capacidad), que se mantenga socialmente activa (evitando que las tripulaciones visitantes se encuentren más de una vez), entre otros requerimientos.

2.3. Restricciones

Las restricciones generales de este problema son bastante simples y directas. Pero más adelante se podrá ver que existen varias representaciones para estas restricciones, aunque todas buscan cubrir los mismos requerimientos.

- Las fiestas solo se pueden realizar en los botes anfitriones.
- La capacidad de un bote no puede ser excedida.
- Las tripulaciones no pueden estar sin hacer nada: o estan visitando botes anfitriones, o bien ellos mismos son anfitriones.
- Las tripulaciones no pueden visitar el mismo bote mas de una vez.
- Las tripulaciones visitantes no pueden cruzarse mas de una vez.

2.4. Problemas Similares

El Progressive Party Problem es considerado un problema particular de Timetabling (programación de horarios), por lo que problemas que buscan calendarizar eventos y optimizarlos bajo algun criterio, son considerados problemas similares.

2.5. Variantes

En el articulo [1] y [3] se muestran algunas variantes que buscan reducir el numero de ecuaciones (y por ende el espacio de búsqueda), del problema. Algunas de esos cambios son:

- Se introduce una variable binaria adicional que indica si una tripulación j y j' visitan a un bote i en el mismo intervalo de tiempo determinado.
- Se introduce una variable entera que indica si un bote fue visitado por una tripulación en un instante determinado (variable con penalizaciones).

3. Estado del Arte

El *"Progressive Party Problem"* (PPP) fue postulado por primera vez y resuelto heurísticamente por Peter Hubbard, el cual es miembro del *"Sea Wych Owners Association"* y del Departamento de Matemáticas de la Universidad de Southhapton. Peter Hubbard era el organizador de una reunión de yates, donde debía, entre muchas otras cosas, organizar una fiesta para los participantes de los yates sobre los yates. [7]

P. Hubbard logró dar con la solución al problema heurísticamente (buscando manualmente), pero pensó que sería interesante ver si es que se podría encontrar una mejor solución y más optima. Le sugirió este problema a H. Paul Williams, el cual formuló el problema bajo *Programación Entera Mixta*, mientras que su colega Sally Brailsford trataba de resolver el problema con software comercial de Programación Entera Mixta. Por otro lado, Barbara Smith usó técnicas de *Programación con Restricciones* (es decir, se formuló como CSP). Estos dos enfoques dieron paso al primer articulo publicado sobre el Progressive Party Problem. [1]

La publicación de P. Hubbard y sus colegas en 1995 [1] (la cual fue incluida posteriormente en otra publicacion en 1996), se enfocó principalmente en detallar las diferencias empíricas que se obtuvieron al desarrollar la solución del PPP en Programación Entera (PE) y en Programación con Restricciones (PR). El resultado fue desastrozo para la PE, debido a que este tipo de problemas de Timetabling eran problemas comunmente resueltos bajo ese paradigma de programación, pero que sin embargo fue bastante alentador para la PR. Esta situación fue lo que inició la interesante "guerra" entre la Programación Entera y la Programación con Restricciones en torno al PPP.

En 1997-1998, un nuevo autor [8, 9], postula que las nuevas técnicas implementadas en la Programación Entera Mixta (MIP) son mucho mejores, por lo que utiliza MIP's para resolver el problema del PPP, lo cual antes había sido desastrozo.

Ya recién en 1999 se utilizó un nuevo acercamiento al PPP, esta vez utilizando Local Search (Tabu Search, Simulated Annealing), las cuales eran eficientes, pero no siempre óptimas[3].

De ahí en adelante, la literatura sobre el tema decayó, pero distintas implementaciones en distintos programas, metodos y representaciónes han tenido publicaciones. [5, 6]

3.1. Métodos Utilizados Para Resolverlo

Para resolver el PPP, se menciona repetidamente en varios de los articulos que se utiliza Programación Entera o Mixta, para los cuales se utiliza el software GAMS[4], o tambien se menciona el uso de Programación con Restricciones[1]. Pero en los articulos más recientes, se hace uso explícito de métodos de búsqueda local, como Tabu Search y Simulated Annealing.

3.1.1. Programación Lineal Entera o Mixta

Es un modelo de programación que contiene restricciones y una función objetivo. En el caso de la Programación Entera, todas las variables deben ser enteras, en cambio para la Programación Mixta, las variables pueden tomar valores enteros como tambien binarios.

Este modelo de programación resulta bastante útil en especial para los problemas de tipo Timetabling, ya que las restricciones y la función objetivo de este tipo de problemas tienden a ser altamente representables mediante formulas matemáticas sencillas.

Software como GAMS, son altamente utilizados en la resolución de este tipo de problemas.

3.1.2. Programación con Restricciones

La programación con restricciones, es otro modelo de programación donde las variables se representan mediante las restricciones que las relacionan. En este tipo de paradigma, las restricciones de los modelos difieren de los modelos tradicionales de programación, ya que en vez de ejecutarse pasos o procedimientos, la Programación con Restricciones especifica las propiedades de las posibles soluciones. Este enfoque hace que la Programación con Restricciones sea parte del mundo de la Programación Declarativa.

Un software bastante conocido en el mundo académico que utiliza este modelo de programación es Prolog, pero en los estudios del PPP se menciona CHIP.

3.1.3. Simulated Annealing

Simulated Annealing es un procedimiento de búsqueda local para explorar el espacio de soluciones más allá del óptimo local, el cual busca reducir el problema de los algoritmos de búsqueda local, aceptando soluciones de "peor calidad".

Al igual que otros algoritmos de búsqueda local, Simulated Annealing necesita una solución inicial, y comienza explorando por el vecindario de todos los optimos locales. A medida que el

[3] Este es un problema común al utilizar estos métodos

[4] `http://www.gams.com/`

tiempo pasa, una función de temperatura va disminuyendo, lo cual obliga al algoritmo a explotar más a los óptimos locales en vez de explorar.

Este método necesita definir una representación de los datos, un movimiento para explotar, una función de evaluación que indica que tan buena es una solución, una temperatura inicial y su respectiva función de decaimiento, y el número de iteraciones máximas para el algoritmo.

3.1.4. Metropolis

El algoritmo Metrópolis es una version simplificada del Simulated Annealing, donde su principal diferencia radica en que en vez de tener una función de decaimento para la temperatura, la temperatura se mantiene constante durante toda la ejecución.

El algoritmo avanza buscando en su vecindario al azar, y luego acepta o no la solución dependiendo de una función probabilística. Es decir, igual que el funcionamiento de Simulated Annealing.

3.1.5. Tabu Search

Tabu Search es un algoritmo de búsqueda local, que busca explotar y explorar el espacio de búsqueda, de acuerdo a una estructura de memoria adaptativa y flexible.

Al igual que Simulated Annealing, Tabu Search necesita una solución inicial para comenzar a explorar, y va almacenando en memoria la lista tabú, para evitar que se produzcan ciclos, y además, puede aceptar soluciones peores a la actual, pero siempre mantiene en memoria a la mejor solución.

Este método necesita definir una representación de los datos, un movimiento para explotar, una función de evaluación que indica que tan buena es una solución, el tamaño de la lista tabú, y el número de iteraciones máximas para el algoritmo.

3.2. Heurísticas y Metaheurísticas

En los distintos articulos revisados, se pueden apreciar distintas técnicas de algoritmos de heurísticas, las cuales buscan ayudar el procesamiento de soluciones para el problema.

En la formulación de Búsqueda Local [2, 8], se utilizan técnicas que buscan optimizar los movimientos realizados para explorar en el espacio de búsqueda. En el caso de la Búsqueda Local se utiliza la heurística *Min-Conflict*[2].

En la heurística Min-Conflict, al tener un set de variables (que en el caso de la Búsqueda Local es el vecindario de una solución), se va a tratar de avanzar a la solución que tenga la menor cantidad de violaciones a las restricciones del problema[5], con el fin de priorizar la *reparación* de las soluciones encontradas[4].

Para los artículos donde se utiliza Programación Lineal o Programación con Restricciones, las heurísticas no juegan un papel tan relevante, como lo es para la Búsqueda Local. Esto es debido a que la Programación Lineal trabaja con las variables definidas directamente.

A pesar de esto, se menciona que para la publicación original [1], se pre-procesa el problema mediante el software GAMS, para luego ser resuelto por el software CPLEX 7.0. Esto se hace para aplicar las heurísticas utilizadas en GAMS (las cuales se basan en los algoritmos branch-and-bound), y así simplificar la tarea que debe realizar CPLEX 7.0 para resolver el problema "reducido". Y para el caso de la Programación con Restricciones, simplemente se aplica un ordenamiento de acuerdo al dominio de las variables[6].

[5]Recordar que técnicas como Tabú-Search y Simulated Annealing, pueden revisar espacios infactibles de soluciones, con el fin de salirse de óptimos locales.

[6]Es importante recordar que para este artículo, el algoritmo más eficiente fue el utilizado en Programación con Restricciones

En el artículo [3] se menciona el uso de una heurística específica para el problema, el cual se basa en "etapas de tiempo"(Time-Stage), el cual básicamente hace un pre-procesamiento del problema para $t = 1$. Con eso se pueden definir h_i y $x_{i,j,1}$. Luego se resuelve para $t = 1, 2$, y se puede definir $x_{i,j,s}$, y así sucesivamente.

3.3. Resultados

Considerando los 3 enfoques mencionados en los artículos, se puede ver que claramente el mejor método utilizado es el de Búsqueda Local en cuanto a los tiempos de ejecución. Pero sus representaciones son mucho más elaboradas, refinadas y complicadas, debido a que se introducen restricciones con castigos, lo cual aumenta la eficiencia, pero destruye la semántica original del problema[7].

A partir de lo que se ha recopilado, los artículos [2, 3] han hecho una buena recopilación de las diferencias al ejecutar los distintos métodos. Es por esto, que este documento utilizaremos la información referenciada en los artículos mencionados, con el fin de simplificar la entrega de los datos.

En la publicación original [1], se menciona que la Programación Lineal fue un total fracaso, y la única vez que sus modelos arrojaron un resultado, al corroborarlo el sistema no pudo lograrlo, ya que después de 189 horas, ya que el problema no se pudo corroborar. Además, la cantidad de variables y filas de ecuaciones generadas al ejecutar los algoritmos, estaban al borde de la capacidad de los computadores de la época utilizados en el estudio.

Sin embargo, los resultados fueron mas optimistas al utilizar Programación con Restricciones, ya que logró resolver en pocos segundos dos de los problemas que Programación Lineal no pudo resolver. Sin embargo, para instancias mas grandes, el método falló.

En el artículo [3], se logra aplicar una nueva representación al problema, y esta vez, con computadores más potentes y mejores pre-procesadores, se logró dar con soluciones optimas al cabo de pocos segundos, pero instancias superiores no puedieron ser corroboradas debido a la heurística utilizada.

En la tabla a continuación se puede ver tabulada los resultados al resolver el programa con Programación Entera Mixta (MIP):

time stage	nº equations	nº variables	nº non-zero elem.	discrete vars.	gen. time	sol. time
1	38830	2620	114130	1758	0,73	1,62
...	...	...	...	...	...	...
(7)	245470	77500	322876	1722	3,42	5,50

Cuadro 1: Resultado simplificado usando la heurística Time-Stage

En la tabla podemos ver que a la séptima iteración se logró hallar una solución óptima. Pero debido al funcionamiento de la heurística (la cual fue explicada anteriormente), el método no puede seguir comprobando para $t > 7, \forall t \in [1, T]$, lo cual es un inconveniente si se quiere establecer un numero fijo de botes anfitriones.

Para el caso de la Búsqueda Local, el artículo [2] de una buena comparación de todos los métodos utilizados.

[7] Lo cual no es malo, pero su formulación se aleja mucho de lo original.

En el artículo se compara la Programación Lineal y la Programación con Restricciones usada en [1], incluyendo la segunda formulación postulada para la Programación con Restricciones propuesta en el mismo artículo. Los resultados podemos observarlos tabulados a continuación:

problem	ILP	CP1	CP2	LS
P_6	fail	27 min.	a few sec.	< 1 s.
P_7	fail	28 min.	a few sec.	< 1 s.
P_8	fail	fail	a few sec.	1 s.
P_9	fail	fail	hours	4 s.
P_10	fail	fail	fail	fail

Cuadro 2: Resultado de las ejecuciones de los distintos métodos

P_i indica que el problema está instanciado para i períodos de tiempo.

Podemos observar que los resultados son consecuentes con lo presentado en [1], dado que la Programación Entera falla en todas las instancias, mientras que la Programación con Restricciones lo resuelve para instancias pequeñas. Lo interesante de este resultado es que la Búsqueda Local puede encontrar resultados para instancias mucho mas grandes en tiempos muy pequeños comparado con las demás técnicas.

A pesar de esto, el método falla cuando se le instancia con 10 períodos de tiempo, lo cual no concluye si es que el método logró encontrar solución para esa instancia o no.

Podemos afirmar con seguridad, que métodos de Búsqueda Local son considerablemente mejores que métodos que buscan en los espacios de búsqueda completos, como lo hacen la Programación Lineal (aunque se desconocen sus métodos).

4. Modelo Matemático

Existen varias formas de plantar este modelo, pero se busca preservar la simplicidad y semántica que respeten la originalidad del problema.

Es por eso que las representaciones y restricciones aquí definidas, están basadas en la publicación original del Progressive Party Problem [1], complementada con la información disponible en [5], y respaldada por la publicación [3].

4.1. Datos y Constantes

- T = intervalos de tiempo definidos para el problema.
- n = cantidad de botes que participarán de la fiesta.
- w_i = tamaño de la tripulación del bote i.
- p_i = cantidad máxima de personas que soporta el bote i.
- $g_i := max\{p_i - w_i, 0\}$ cantidad máxima de invitados que puede recibir el bote anfitrión i.

4.2. Variables

Las variables definidas a continuación son variables que se pueden encontrar a lo largo de todos las publicaciones, y se pueden considerar un modelo matemático base para el PPP.

$$x_{i,j,t} = \begin{cases} 1 & \text{si la tripulación } j \text{ visita el bote } i \text{ en el instante } t, \\ 0 & \text{si no lo hace.} \end{cases} \tag{1}$$

$$h_i = \begin{cases} 1 & \text{si el bote } i \text{ es un bote anfitrión,} \\ 0 & \text{si no lo es.} \end{cases} \tag{2}$$

A continuación se define una variable introducida en el articulo [3], debido a que esta formulación de la variable permite una semántica mucho mas cercana a la propuesta por la publicación original, ademas de evitar lo inconveniente de agregar variables con castigo, como se hace en el artículo donde se utiliza Búsqueda Local [2]. Esto se puede corroborar al mirar que otro autor que utiliza la Búsqueda Local, prefiere la representación sin variables con castigos [8].

$$m_{j,j',t} = \begin{cases} 1 & \text{si la tripulación } j \text{ y } j' \text{ se encuentran en el mismo bote en el instante } t, \\ 0 & \text{si no lo hacen.} \end{cases} \tag{3}$$

4.3. Función Objetivo

La función objetivo de este problema es minimizar la cantidad de botes anfitriones. A pesar de esto, en la literatura se puede apreciar que este valor (la cantidad de botes anfitriones), se define de manera fija como una constante [3, 1, 8, 2], con el fin de reducir la complejidad del problema, y así convertir a la función objetivo en una restricción más.

$$\min z = \sum h_i \quad \forall i \in [1, n], i \neq j \tag{4}$$

4.4. Restricciones

Como se ha mencionado anteriormente, las restricciones acá representadas buscan mostrar la mayor cantidad de semántica fiel al artículo original.

La primera restricción es que un bote puede ser visitado solo si el bote visitado es un bote anfitrión:

$$x_{i,j,t} \leq h_i \quad , \forall i, j \in [1, n] \quad \forall t \in [1, T] \tag{5}$$

La otra restricción utilizada en el modelo, es la restricción de que no se puede superar la capacidad máxima de pasajeros invitados en un bote. Esta se define como:

$$\sum_{j|j \neq i} w_j x_{i,j,t} \leq g_i \quad , \forall i, j \in [1, n] \quad \forall t \in [1, T] \tag{6}$$

La siguiente restricción permite asegurar que la tripulación de un bote anfitrión no puede abandonar su yate, ya que ellos son los anfitriones. Esto se define de la siguiente manera:

$$h_i + \sum_{i|i \neq j} x_{i,j,t} = 1 \quad , \forall i, j \in [1, n] \quad \forall t \in [1, T] \tag{7}$$

Existe otra restricción relacionada con los movimientos que pueden realizar las tripulaciones. Se requiere que una tripulación invitada no visite un mismo bote más de una vez durante la fiesta, aunque no se requiere que visite todos los botes de las fiesta. Esto se puede leer como:

$$\sum_t x_{i,j,t} \leq 1 \quad ,\forall i,j \in [1,n], i \neq j \quad \forall t \in [1,T] \tag{8}$$

Las siguientes restricciones buscan modelar las interacciones que tienen las tripulaciones visitantes durante la fiesta.

Se mencionó anteriormente, que el modelo que se representa acá es el que se puede ver en [3]. En este modelo se introduce la variable binaria $m_{j,j',t}$ del encuentro de la tripulación j con la tripulación j' en el tiempo t. La restricción que define esta variable indica que si dos tripulaciones se han encontrado en el mismo lugar al mismo tiempo, es porque estuvieron en el mismo bote al mismo tiempo:

$$m_{j,j',t} \geq x_{i,j,t} + x_{i,j',t} - 1 \quad ,\forall (j,j',t) | j < j' \quad \forall t \in [1,T] \tag{9}$$

Ahora que la variable $m_{j,j',t}$ se encuentra debidamente definida bajo la restricción antes mencionada, podemos modelar la restricción que nos pide que las tripulaciones visitantes no se puede encontrar más de una vez con otras tripulaciones visitantes durante la fiesta. Esto se puede ver como:

$$\sum_t m_{j,j',t} \leq 1 \quad ,\forall j < j' \quad \forall t \in [1,T] \tag{10}$$

5. Conclusiones

- El Progressive Party Problem describe un problema del tipo Time Tabling que fue planteado casi fortuitamente por P. M. Hubbard, donde los resultados fueron desconcertantes inicialmente, debido a que los métodos para resolver ese tipo de problemas no arrojó los resultados esperados. De esto último se desprende que la Programación Lineal no es el método más óptimo para resolver el Progressive Party Problem.
- Entre los métodos para resolver el Progressive Party Problem que se vieron en este documento, como por Programación Lineal, Programación con Restricciones y Búsqueda Local, se logró comprobar que el enfoque de Búsqueda Local es el más eficiente en general para las distintas instancias del problema.
- Las variantes encontradas para el problema, no son de mayor incumbencia, ya que los objetivos no se alejan de los originales, pero sus representaciones sí. Esto último se ve reflejado negativamente en la semántica de las restricciones para los distintos métodos que usan los autores. Y es más, en una representación utilizada en la Programación Lineal, elimina las restricciones cargadas de una semántica alejada de la original.
- El modelo matemático utilizado para representar el problema es el más sencillo y el más simple en términos semánticos, lo cual facilita la lectura.
- Finalmente, se puede concluir que el Progressive Party Problem es un problema combinatorio altamente complejo, el cual tiene una inclinación a resolverse más facilmente mediante técnicas incompletas por sobre las completas. Su representación sencilla es engañadora, ya que su complejidad tiende a ser alta, pero distintas heurísticas ayudan a facilitar el proceso de encontrar soluciones. Es por esto último que adaptar distintas heurísticas a este problema es algo necesario, ya que el único límite parece ser el ingenio humano.

6. Bibliografía

Referencias

[1] B.M.Smith, S.C.Brailsford, P.M.Hubbard, and H.P.Williams. The progressive party problem: Integer linear programming and constraint programming compared. *Constraints*, 1:119–138, 1995.

[2] Philippe Galinier and Jin-Kao Hao. Solving the progressive party problem by local search. In Stefan Voß, Silvano Martello, IbrahimH. Osman, and Catherine Roucairol, editors, *Meta-Heuristics*, pages 419–432. Springer US, 1999.

[3] Erwin Kalvelagen. On solving the 'progressive party problem as a mip. `http://amsterdamoptimization.com/pdf/ppp.pdf`, 2002. (Ultima visita el 26/05/2014).

[4] Steven Minton, Mark D. Johnston, Andrew B. Philips, , and Philip Laird. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Eighth National Conference on Artificial Intelligence (AAAI-90)*, 1990.

[5] Helmut Simonis. Customizing search (progressive party problem). `http://4c.ucc.ie/~hsimonis/ELearning/party/handout.pdf`, 2009.

[6] Helmut Simonis. Progress on the progressive party problem. In Willem-Jan Hoeve and JohnN. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 5547 of *Lecture Notes in Computer Science*, pages 328–329. Springer Berlin Heidelberg, 2009.

[7] Joachim Paul Walser. The progressive party problem. `http://www.ps.uni-saarland.de/~walser/ppp/ppp.html`, 1997. (Ultima visita el 26/05/2014).

[8] Joachim Paul Walser. Solving linear pseudo-boolean constraint problems with local search. In *FOURTEENTH NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, 1997.

[9] Joachim Paul Walser. *Domain-Independent Local Search for Linear Integer Optimization*. PhD thesis, Universität des Saarlandes, 1998.