Inteligencia Artificial

Estado del Arte: *The Social Golfer Problem*

Cristian Soriano

2 de julio de 2024

Evaluación

Mejoras 2da Entrega (10 %):	_____
Código Fuente (10 %):	_____
Representación (5 %):	_____
Descripción del algoritmo (15 %):	_____
Experimentos (10 %):	_____
Resultados (40 %):	_____
Conclusiones (10 %):	_____
Nota Final (100):	_____

Resumen

The Social Golfer Problem es un problema famoso en el campo de la optimización combinatoria. Este problema es esencial para todo lo que es organización de torneos tipo Round-Robin ya que en su forma más general trata de organizar mini torneos por la máxima cantidad de semanas posibles tal que ningún jugador se tope con otro más de una vez. En este informe se van a explorar los distintos enfoques que se le han dado al problema, tanto en modelos como en las técnicas que se han usado para resolverlo. El objetivo final es formar un punto de vista en base a la información recolectada para luego enfocarse netamente en probar distintas técnicas para encontrar soluciones guiándonos por lo que se tiene actualmente.

1. Introducción

En este informe se hará una revisión y análisis del estado del arte de *Social Golfer Problem* (abreviado SGP a lo largo del trabajo), un problema de optimización combinatoria que ha generado bastante discusión en términos de rupturas de simetrías en problemas similares. La tarea principal es organizar una cierta cantidad gp de golfistas en g grupos de p jugadores durante w semanas; generalmente queremos extender el torneo lo más posible por lo que se busca maximizar w. Partiremos con la definición del problema (2), donde se da un contexto profundo de cómo se debería abordar el problema dado su origen, problemas similares que son instancias de SGP y variantes/generalizaciones que se han postulado en diversas investigaciones. En el estado del arte (3) veremos los distintos enfoques que se han dado tanto en modelos y

métodos de resolución, haciendo comparaciones en rendimiento según lo que se tiene a la fecha. En este informe también se exploran algunos modelos propuestos que han mostrado resultados interesantes dada su complejidad, vistos en 4, haciendo comparaciones según lo que se encontró en el estado del arte. En 5 empezamos a armar una metodología para plantear nuestro propio algoritmo para resolver el problema. Luego en 6 definimos los distintos tipos de construcciones que se quieren aplicar, en conjunto con un componente de búsqueda local con el algoritmo Hill Climbing. Para poner a prueba la metodología planteada en 7 definimos experimentos cruciales para medir el rendimiento de nuestro algoritmo y saber qué parámetros son ideales para este para luego en 8 mostrar los resultados obtenidos de los experimentos. Finalmente haremos notar las conclusiones (9) relevantes que se pudieron sacar de este estudio, dando también posible trabajo a futuro por realizar.

2. Definición del Problema

En [1] se define al SGP como un problema de optimización combinatoria que surge de una pregunta publicada en `sci.op-research` en mayo de 1998: *32 golfistas juegan golf una vez a la semana, siempre en grupos de a 4. ¿Por cuántas semanas pueden hacer esto tal que dos golfistas no jueguen en el mismo grupo más de una vez?*

En el mismo libro se plantea una generalización de SGP con el siguiente problema de decisión: ¿Es posible programar $g \times p$ golfistas en g grupos de p jugadores por w semanas tal que dos golfistas no jueguen en el mismo grupo más de una vez?
Una instancia de SGP se denota mediante una tripleta $g - p - w$. La instancia original del problema se denotaría como $8 - 4 - w$, donde w sería la cantidad de semanas donde se cumplen las condiciones. En adición a todo esto, también se puede considerar a SGP como un problema de optimización discreta que maximiza el número de semanas w^* para g y p dados, donde $w^* \leq \frac{g \cdot p - 1}{p - 1}$, es decir, la cantidad de semanas óptima no sobrepasa la cantidad de parejas únicas que pueden formarse en todos los grupos [2].

El problema en sí es altamente combinatorio y simétrico por lo que no es sorprendente que haya llamado la atención por parte de la comunidad de programación con restricciones. Ha planteado problemas interesantes dentro del modelado y el quiebre de la simetría, volviéndose uno de los puntos de referencia estándar para evaluar esquemas que rompen la simetría de un problema [3].

Instancias conocidas de SGP son los *Cuadrados de Euler* y el problema de *Kirkman*. El primero, también conocido como el problema de los 36 oficiales, es la instancia $6 - 6 - 4$ de SGP, conocida por ser imposible [1]. El problema es el siguiente: *Seis oficiales de 6 rangos distintos son elegidos de seis regimientos distintos. Se requiere arreglarlos en un cuadrado sólido tal que ningún oficial del mismo rango o del mismo regimiento se encuentre en la misma fila o columna.* Es un equivalente a arreglar 36 pares de enteros, cada uno menor o igual a seis, en un arreglo cuadrado tal que el primer (o segundo) número de los pares en cualquier fila o columna sean todos distintos, y que ninguno de 2 pares sean idénticos [4].

El segundo problema, también conocido como el problema de las quince alumnas, es la instancia $5 - 3 - 7$ de SGP, cuya descripción es la siguiente: *Encontrar un calendario semanal para quince alumnas que salen a caminar a diario en cinco filas de tres alumnas cada una, de tal manera que dos chicas no deben caminar en la misma fila más de una vez por semana.* Una forma equivalente a la última parte es que *cualquier alumna debe caminar exactamente una vez en la misma fila con cada una de las otras chicas.* Es conocido por ser uno de los problemas más importantes, celebrado y fascinante en el campo de las combinatorias y las matemáticas recreativas [5].

Otra instancia de SGP es *The Social Doubles Tournament Problem* (SDTP), el problema consiste en crear un torneo tipo round robin (cada participante se encuentra con los demás) donde los equipos, compuestos por pares de jugadores, no están fijados a priori pero tienen que

ser generados mientras se crean los rounds y donde cada jugador debe tener a todos los otros jugadores como compañero y como oponente. Además se pide que siempre que 4 jugadores se encuentren, hacer que se sigan encontrando durante tres partidos consecutivos en un mini torneo donde los jugadores cambian compañeros. En [6] se encontró que, para $4n$ jugadores un torneo round robin prevé $4n-1$ rondas. Como cada cuádrupla determina 3 rounds para cada jugador, el número de rondas disjuntas totales es $\frac{4n-1}{3}$ y debe ser un número entero. Debido a esto, SDTP permite que existan soluciones factibles para $4n$ jugadores y se denota que SDTP_{4n} es equivalente a la instancia $n-4-\frac{4n-1}{3}$ de SGP.

SGP y problemas similares tienen varias aplicaciones prácticas, tales como desarrollos de frameworks para asignación de grupos, programación de grupos para salones de espera en zoom e incluso el estudio de la agrupación de torneos (como round-robin). Por ejemplo, en [7] se exploran soluciones a este problema con el uso de el sistema de numeros binarios en el torneo round-robin para asignar grupos de estudiantes en laboratorios y trabajos en grupo a lo largo de un semestre de un curso de química. Por otro lado, en [8] se exploraron metodos para hacer las clases online durante la pandemia más didacticas, empleando uso de los salones de espera que ofrecía Zoom y Microsoft Teams para desarrollar actividades en grupo, donde se desea mezclar los grupos de tal manera que dos estudiantes no tuvieran el mismo grupo en más de una ronda, donde se uso una variación de SGP llamada *Social Golfer Problem with Adjacent Group Sizes* (SGA), donde se busca asignar v participantes a grupos con tamaños de K (donde K es un conjunto de enteros positivos de tamaño a lo más 2, si $|K|=2$ entonces $K=\{k,k+1\}$ para algún k) en r rondas de tal manera que cada participante aparece en todas las rondas, todos los pares de participantes aparecen a lo más en un grupo y todas las rondas tienen el mismo conjunto de tamaños de grupos; es decir, se quiere encontrar un conjunto de rondas donde los tamaños de todos los grupos no necesariamente es el mismo. En otros casos se usan aplicaciones de otras variaciones de SGP, como en [9], donde se usa *Traveling Social Golfer Problem* (TSGP), una generalización de SGP, para asignar grupos en la Volleyball Nations League, donde las dieciséis mejores naciones por genero compiten por un trofeo en un torneo que dura 6 semanas. En TSGP, definido en el mismo artículo, todos los grupos deben jugar en diferentes lugares, donde el objetivo es crear un calendario que minimiza la desigualdad (o injusticia) generada por los distintos tiempos de transporte entre las locaciones para cada golfista.

Ahora que tenemos un entendimiento base de SGP, podemos pasar a analizar distintas formulaciones del problema y cuales han sido las tendencias para encontrar soluciones a distintas instancias de éste.

3. Estado del Arte

A lo largo de esta investigación nos hemos fijado que se han tomado muchos puntos de vista distintos para modelar y resolver el problema, desde propuestas de heurísticas greedy para obtener mejores resultados, hasta la ruptura de simetrías con el uso de teoría de conjuntos y con la suma de otras restricciones, mejorando el rendimiento a la hora de buscar soluciones.

En [10] se propone una mejora a lo que se propone en [3]. Se introduce el concepto de grado de libertad de los jugadores, que, informalmente, denota con cuántos jugadores aún se puede emparejar al jugador. Esta métrica permite la mejora de dos métodos: bactracking completo y búsqueda local. Para el primer método, se plantea una heurística greedy que puede ser usada para guiar la búsqueda mediante backtracking para SGP. Si se asume que los jugadores por equipo p es par, la tarea de asignar jugadores en grupos en cada semana es equivalente a organizar pares de jugadores dentro de grupos. De este modo, se visitan las semanas una tras otra y en cada semana se recorren los grupos en su orden natural. Por cada par de posiciones adyacentes en un grupo, se necesita seleccionar un par de jugadores que aún quedan por agrupar en la semana actual. Aquí se ocupa el grado de libertad para guiar la selección del par que tiene la mínima libertad con respecto a la configuración parcial actual. La intención detrás de esto es

que si un par de jugadores esta cercano a quedarse sin potenciales parejas, entonces deben estar en el mismo grupo mientras sea posible. Si un grupo no puede ser completado, o se genera un conflicto, ocurre backtracking: se deshace la elección más reciente de jugadores y se selecciona un par con el mayor grado de libertad. Si p es impar, se emparejan todos los jugadores posibles y después se rellena la posición que falta con cualquier jugador que sea compatible con el resto de jugadores que estén en el grupo. Otra opción sería usar tripletas y conjuntos de jugadores más grandes. La implementación de backtracking es tal que un patrón puede denotar de qué manera se considera a cada grupo por individual. Por ejemplo el patrón $3-2-2-1$ puede ser aplicado a instancias $g-8-w$ y significa que por cada grupo, se procede de la siguiente manera: primero se programa una tripleta de jugadores; en los lugares que quedan, se colocan dos pares inmediatamente después (cada uno elegido según la libertad mínima) y cualquier jugador que aún pueda jugar con todos en el grupo se pone en la posición final. En la figura 1 se muestra la tabla con los resultados para las distintas instancias y patrones obtenidos con este método, por ejemplo, se resuelve el problema de Kirkman de manera óptima. También se resuelve el SGP original por 9 semanas con un patrón $2-2$ (dos pares) y 4 (considerando cuartetos).

Table 1 Complete backtracking with greedy heuristic on selected instances

Instance	Pattern	Time	Instance	Pattern	Time
5–3–7	3	0.51 s	6–6–3	3–3	0.29 s
8–4–9	2–2	0.08 s	6–6–3	4–2	1.54 s
8–4–9	4	1.58 s	9–9–3	3–3–3	14.4 s

Figura 1: Tabla con los resultados de la heurística con backtracking

En el paper se usa la misma heurística greedy para generar configuraciones iniciales buenas para un enfoque de búsqueda local. Primero se usa una forma dual de la heurística definida, es decir, no garantiza encontrar la solución óptima pero si una cercana a ser óptima, en adición con un componente aleatorio que permite agregar pequeñas perturbaciones a las configuraciones iniciales. La heurística visita las semanas una tras otra. Una única semana se produce de la siguiente manera: Los grupos de esta semana se recorren uno tras otro. Al igual que antes, por cada par de posiciones adyacentes en un grupo, la heurística debe seleccionar un par de jugadores que aún deban ser asignados en la semana actual. Selecciona el par que tiene la libertad máxima con respecto a la configuración parcial actual. Hay un parámetro γ, con $0 \leq \gamma \leq 1$, que es usado para randomizar la heurística: En el caso que hayan empates, se realiza una elección aleatoria entre los pares de jugadores que tienen el máximo grado de libertad con probabilidad γ. Con probabilidad $1-\gamma$, los pares se consideran como ordenados, con el jugador con numeración más baja siendo primero, y se selecciona el par que es lexicográficamente más pequeño (el que tenga la letra que venga primero en el alfabeto). Después que un par de jugadores es seleccionado y asignado a un grupo, se substrae un número grande en forma de penalidad de la libertad de ese grupo en semanas siguientes, con esto se desalienta a que ese par sea seleccionado nuevamente en un grupo diferente. En otros casos, la heurística no presta atención a potenciales conflictos en un grupo, y nunca deshace una elección de pares. En el caso en que p es impar, la heurística aún puede trabajar con pares de jugadores a excepción del último jugador en cada grupo. Con probabilidad γ, ese jugador es seleccionado de manera aleatoria de todos los jugadores que aún deben ser asignados en esa semana. Con probabilidad $1-\gamma$, el jugador cuya numeración es la más pequeña es seleccionado. Al igual que antes, la heurística es generalizada desde pares a conjuntos más grandes de jugadores, y nuevamente hay un intercambio entre maximizar la libertad de los grupos y la eficiencia. La intuición detrás de maximizar la libertad entre jugadores de un grupo es para hacer espacio a buenos movimientos locales, que serán discutidos en lo que sigue. Se apuntó a mantener el componente de búsqueda local lo más simple posible. Se usó el enfoque de un algoritmo memético, definido en [11], el cuál siguiendo la filosofía de la programación

evolutiva, ocupa únicamente mutación como forma de diversificar la búsqueda, lo que alivia la necesidad de realizar rupturas de simetría; esto en conjunto con una base de búsqueda tabú. Se elimino la componente de reinicio debido a que se encontró innecesaria en experimentos: Incluso después de omitir la componente de reinicio, la búsqueda local implementada excedió los enfoques de búsqueda local reportados en la literatura. Sea que C denota una configuración. La tripleta (i, j, k) es una posición conflicto sí y solo sí dondequiera que un jugador esta en el mismo grupo con otro jugador más de una vez. Sea que $f(C)$ denote el número de posiciones conflicto en C. El rango de $f(C)$ es $[0, gpw]$. Un *swap* que afecte a la posición (i, j, k) significa el intercambio de los valores de G_{ijk} con cualquier otro valor de alguna variable $G_{ij'k'}$, con $j' \neq j$ y $k' \neq k$, donde G_{ijk} denota qué jugador juega en la semana i, grupo j y posición k, con $0 \leq G_{ijk} \leq n = gp$. El movimiento es bueno ya que no rompe restricciones importantes del modelo usado. Una iteración de esta búsqueda local procede de la siguiente manera: Primero se determina $f = f(C)$. Si $f = 0$, se encontró una solución y estamos listos. En caso contrario, de todos los swaps que afectan a las posiciones conflictivas, hacer el swap que lleve a una configuración C' con $f(C')$ mínimos entre todas las configuraciones consideradas (rompiendo empates lexicográficamente). En adición, se asocia una lista tabú con cada semana que guarda todos los pares de jugadores que fueron intercambiados en las ultimas 10 iteraciones en esa semana. Swaps que afectan a un par de jugadores en la lista tabú de la semana correspondiente son considerados solamente si resultan en una mejora sobre la mejor solución encontrada hasta el momento. Si no hubo una mejora en las últimas 4 iteraciones, se hacen 2 swaps aleatorios. Se determinaron todos estos parámetros en pruebas sobre todas las instancias que se definieron en el artículo, y de todas esas, la configuración dada funcionaba mejor cuando se promedió sobre los tiempo de ejecución de todas las instancias probadas. En la figura 2 se pueden ver los resultados obtenidos.

Table 3 Running times for selected instances using the parameters described in Sect. 7 ($\gamma = 0$)

Instance	Time	Instance	Time
5−3−7	0.2 s	8−8−5	37 s
8−3−10	0.4 s	9−3−11	0.2 s
8−4−9	1.2 s	9−4−9	5.6 s
8−4−10	11 min	10−3−13	7.8 s

Figura 2: Tabla con los resultados obtenidos usando búsqueda local

La heurística usada es aleatoria y generalizada, y mostró que mejora resultados obtenidos mediante búsqueda local. Además de ser una de las más simples, es una de las más competitivas con otras meta heurísticas y técnicas basadas en restricciones en otras instancias.

En [12] se utiliza un *greedy randomized adaptive search procedure* (GRASP) para resolver SGP. Dentro de una estructura básica de un GRASP para un problema de minimización, se adhiere al esquema con lo siguiente: La condición de termino es el descubrimiento de un horario sin conflictos (o un time out, cualquiera que ocurra primero); basándose en el mismo concepto de grado de libertad mencionado anteriormente ([10]), se generan las configuraciones iniciales con la misma heurística greedy aleatoria; Se usa la misma componente de búsqueda local que en el caso anterior, es decir, búsqueda tabú. Hay que notar que este estudio fue publicado antes que [10], pero las especificaciones entre ambos son las mismas y cambian solamente en la parte de los resultados obtenidos. En este caso, la ejecución de una única instancia procede de la siguiente manera: La heurística greedy es usada para generar cinco condiciones iniciales variando γ, incluyendo 0, 0.1, 0.2 y dos valores sacados en momentos aleatorios entre 0.3 y 1. Luego, la componente de búsqueda local se ejecuta con cada una de las configuraciones obtenidas por a lo más un minuto. Si no se encuentra una solución, se reinicia la búsqueda con la configuración

inicial que tenía el mínimo número de conflictos mientras era ejecutado, y luego es ejecutado hasta su finalización. Los resultados se muestran en la figura 3.

Table 1 Running times for selected instances ($\gamma = 0$)

instance	time	instance	time
5−3−7	0.2s	8−8−5	37s
8−3−10	0.4s	9−3−11	0.2s
8−4−9	1.2s	9−4−9	5.6s

Figura 3: Rendimiento de GRASP para instancias seleccionadas

Comparando con la figura 2 los resultados obtenidos son bastante similares.

En [13] se propone un modelo basado en programación entera y se rompen las simetrías durante la búsqueda para resolver el problema de Kirkman. Según este paper, las simetrías de SGP se pueden enumerar de la siguiente manera: 1) los jugadores pueden ser intercambiados dentro de los grupos; 2) los grupos pueden ser intercambiados dentro de las semanas; 3) Las semanas pueden ser ordenadas de manera arbitraria; 4) los jugadores pueden ser renombrados en $(gp)!$ permutaciones. Las restricciones que rompen simetrías fallan en remover estáticamente todas las simetrías entre los jugadores en SGP. En [13] se especializa una técnica llamada *Symmetry Breaking via Dominance Detection* (SBDD), la cual en su forma base consta en detectar puntos de elección simétricos dentro del procedimiento de búsqueda. Cada vez que el algoritmo de búsqueda genera un nuevo punto de decisión, se chequea si es equivalente a o dominado por un nodo que fue expandido antes. En ese caso, el punto de la decisión actual puede ser podado; caso contrario, se procede de forma normal [14]. Este algoritmo requiere calcular una coincidencia en un grafo bipartito, para lo cuál el mejor algoritmo conocido corre en $\mathcal{O}(n^{\frac{5}{2}})$ [15]. Sin embargo es significativamente más fácil chequear que existe una función que mapea una solución vieja (una hoja del árbol de búsqueda) a una nueva. En [14] se remarca que las posibles simetrías pueden ser enumeradas buscando un match desde la primera semana de la primera solución (o solución parcial) hacia cualquier semana de la segunda solución. La idea clave para realizar el chequeo de la simetría de forma eficiente es hacerlo de manera floja: en lugar de elegir un matching completo y chequear las otras semanas después, vale la pena chequearlas mientras que el matching es construido. El chequeo para mapear pares puede ser rendido con un costo bajo con una fase de cálculos previos: 1) Derivar el conjunto de pares de cada semana i de la primera solución P, ordenado de acuerdo al mayor elemento del par, denotado $C_i(P)$. Este cálculo puede llevarse a cabo solamente una vez que la solución es encontrada y guardada; 2) Construir una tabla de los números de la semana indexados por pares de la segunda solución P'. Un procedimiento eficiente de chequeos de simetría aplicado en las hojas permite calcular todas las soluciones únicas pero no mejora la búsqueda en sí: ningún subárbol es eliminado. Sin embargo, usando isomorfismos de grafos, se podría usar chequeos de simetría para podar subárboles grandes. En la figura 4 se ilustra esta idea. Sea P una solución inicial (el árbol se explora con depth first search, de izquierda a derecha) y P' una segunda solución demostrada ser isomorfa a P ($P' = \gamma(P)$ con γ una biyección entre los vértices de P' a P). Se denota $n = P \setminus P'$ como el nodo más profundo en el árbol de búsqueda que es común a los caminos desde la raíz a P y P', y a s (respectivamente s') su sucesor inmediato (se asume que el árbol de búsqueda es binario) que lleva a P. Se puede demostrar que el nodo s' es la imagen de s mediante el isomorfismo γ. En este caso, lo que falte por explorar del subárbol de s' es la imagen de un subárbol ya explorado que empieza por s. Entonces puede ser podado porque solamente llevaría a soluciones que son imágenes por γ de soluciones que ya se encontraron. A este idea se le conoce como poda profunda.

En [13] se propone una integración de poda profunda donde no solo se calculan los isomorfismos de las soluciones sino que también en estados del árbol de búsqueda que pueden ser

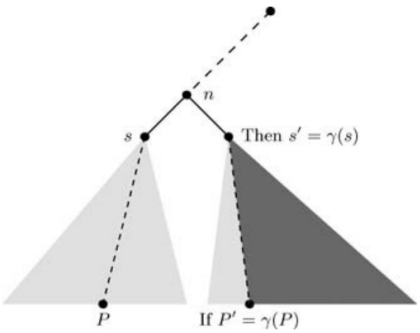


Figura 4: Árbol de búsqueda mencionado; El subárbol más oscuro puede ser podado

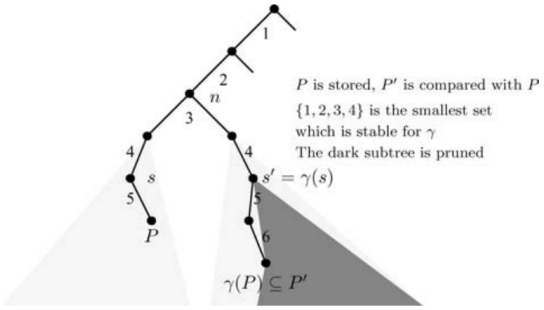


Figura 5: Poda profunda con SBDD

descritos con los dominios actuales de las variables. A esta variación se le denomina SBDD+. El mismo algoritmo para chequear simetrías puede ser modificado fácilmente para ser aplicado a situaciones incompletas, intentando mapear la primera semana de la solución inicial solo en la primera semana de la segunda. La fase de chequeo se mantiene igual, una simetría se encuentra si todos los pares en P tienen una imagen en P'. El chequeo de la dominancia sigue siendo costoso y no debe ser realizado tan seguido. La frecuencia de chequeo debe estar relacionada a la estructura del problema. Una buena premisa para SGP es guardar nodos en cada profundidad del árbol de búsqueda, chequear dominancia para nodos solamente contra nodos de profundidad más pequeña y chequear dominancia solo para nodos en profundidades que son múltiplos de p. La profundidad del árbol de búsqueda es el número de golfistas gp; en cada nivel, pueden haber g^w nodos por guardar. El tamaño de todos los nodos que se guardan está delimitado por gpg^w. Este límite es malo debido a las numerosas simetrías que son removidas por las restricciones. Para el problema de Kirkman solo se guardan 15 nodos al mismo tiempo. La figura 5 ilustra el método en cuestión.

En la figura 6 se puede ver el número de puntos de decisión creados, la cantidad de *backtracks*, el total de soluciones encontradas, el número de chequeos de dominancia y el tiempo de CPU para diferentes combinaciones de métodos presentados para calcular las 7 soluciones del problema de Kirkman (que se hayan encontrado 11 soluciones significa que se detecto el isomorfismo solamente en la hoja del árbol de búsqueda). Los métodos +(4) y +(5) se refieren a la adición de restricciones redundantes al modelo con el objetivo de reducir la cantidad de backtacks.

En una investigación más reciente ([16]), se formula un modelo parcialmente booleano (PB) de SGP para encontrar una configuración para un torneo de Mahjong. El uso de un modelo PB sobre la formulación SAT es debido a que el modelo SAT de este problema (que veremos en la sección de modelos matemáticos) para una cantidad grande de jugadores es muy lenta para

Table 1. Computing the 7 solutions of Kirkman's problem

	Choice points	Fails	Solutions	Dom. checks	CPU(s)
Leaves	20 062 206	19 491 448	20 640		5 925
McKay	1 845 543	1 803 492	934		484
SBDD	107 567	104 134	11	5 373	24
SBDD+	29 954	28 777	11	456	7.8
+(4)	18 705	16 370	11	456	9.4
+(5)	18 470	16 169	11	443	36

Figura 6: Tabla comparativa entre los métodos usados

generar las restricciones. Se asume que, mientras que un PB pueda no encontrar una solución óptima global, si dejó de encontrar mejoras, al menos habrá encontrado una solución óptima local. Además se asegura que a la fecha del paper, el método automatizado desarrollado en [10] es el mejor, aunque el método SAT/PB es competitivo. Una de las restricciones de este caso particular es el balance del viento (durante cada ronda a cada jugador se le asigna un punto cardinal distinto, conocido como viento): Con el fin de compartir una penalidad potencial de no tener un segundo turno siendo dealer a través de todos los jugadores, cada jugador debe ser alocado a cada posición inicial del viento aproximadamente el mismo número de veces. Para calendarios de torneos que corresponden a instancias duras de SGP, se puede importar una solución de la instancia y simplemente sintonizar el balance del viento. Una vez definido un enncoding para restricciones con los vientos (en la sección de modelos matemáticos se entrara en más detalle), se usa para generar calendarios para el torneo European Riichi Mahjong Championship (ERMC) del año 2016 y varios torneos más pequeños en el Reino Unido. La instancia de SGP que describe el torneo ERMC es la $32-4-10$ y se usa el encoding desarrollado, encendiendo (o habilitando) las restricciones de manera incremental para obtener los mejores resultados. Forzar a que ocurra la socialización (cada jugador debe jugar contra todos los demás) de por si solo tomó 18s. Se habilito el movimiento de las mesas y el balance del viento para darle a cada jugador 2 turnos en cada asiento más 1 turno como Este o Sur y 1 turno como Oeste o Norte, lo que tomo 2m con 10s en resolver. Manteniendo una restricción dura y agregando otra blanda, con un tiempo límite de 1 hora, se generó un calendario que no respeta 122 de $\binom{128}{2} = 8128$ restricciones blandas. En general la instancia ocupó 1.3M de variables y 3.9M de restricciones y en la figura 8 se muestran puntos de referencia para instancias similares. A modo de comparación, resolver la instancia con búsqueda local balanceando los vientos usando la formulación propuesta produjo una solución en menos de 1 s. Para los torneos más pequeños, el grafo del torneo era más pequeño por lo que no se forzó con restricciones. Torneos de 1 día solían tener 5 sesiones y tenían de 24 a 52 jugadores ($6-4-5$ a $13-4-5$). Torneos de 2 días solían tener 8 sesiones y de 32 a 68 jugadores ($8-4-8$ a $17-4-8$). Se comparo la formulación monolítica con estos intervalos, con un tiempo máximo de 10 minutos para el solver y haciendo que el balance del viento sea una restricción suave. Para instancias en el intervalo de 1 día se tomo menos de 0.5s para resolver restricciones para un calendario con máxima socialización, movimiento de mesas y balance del viento, a excepción de $6-4-5$, la cuál tomó 3.6s. Para intervalos de 2 días (ver figura 7), la instancia $8-4-8$ es significativa ya que es la formulación original de SGP y queda fuera del alcance para la formulación SAT y PB, por lo que se importó una solución para balancear. Para $9-4-8$ y $10-4-8$, *clasp* resolvió las restricciones solo con el balance del viento deshabilitado. Para el resto, *clasp* encontró soluciones sin respetar de 2 a 21 restricciones con el viento. En todos los casos se pudo sintonizar perfectamente el balance del viento, satisfaciendo todas las restricciones, usualmente en menos de 2s.

Table 2. Benchmarks applying monolithic encoding to 2-day tournaments (g-4-8).

Groups	8	9	10	11	12	13	14	15	16	17
Variables	14k	18k	22k	27k	32k	37k	44k	50k	57k	64k
Constraints	43k	60k	81k	105k	134k	168k	207k	252k	304k	361k
Socialisation only time (s)	–	53	0.52	0.46	0.56	0.77	0.94	1.2	1.5	1.7
Constraints violated after 10 m	–	–	–	2	3	3	3	13	11	21
Total wind constraints	–	–	–	176	192	208	224	240	256	272
Wind balance tuning time (s)	0.045	1.2	0.094	0.15	0.17	0.030	1.5	0.13	15	9.5

Figura 7: Puntos de referencia aplicando el encoding para torneos de 2 días

Table 4. Benchmarks applying monolithic encoding to large tournaments (g-4-10).

Groups		28	29	30	31	32	33	34	35	36
Soc. + wind time (s)		2.7	3.1	4.2	3.3	131	5.3	102	76	241
Soc. + wind + $d = 2$ time (s)		46	46	38	100	847	664	1189	-	-
With $d = 3$ soft:	Variables	0.9M	1.0M	1.1M	1.2M	1.3M	1.4M	1.5M	1.7M	1.8M
	Constraints	2.6M	2.9M	3.2M	3.5M	3.8M	4.2M	4.6M	5.0M	5.5M
Constraints violated after 1h		0	62	61	1	122	196	-	327	-

Figura 8: Instancias similares al torneo ERMC 2016

4. Modelos Matemáticos

Como lo prometido es deuda, empezaremos con los modelos que se consideran más simples y vamos a ir escalando un poco la dificultad a medida que revisemos más modelos. Partimos revisando una formulación SAT de SGP, la cual es una versión mejorada de un modelo previo. En [17], consideramos $x = gp$ golfistas. La formulación original introduce variables G_{ijkl} ($1 \leq i \leq x, 1 \leq j \leq p, 1 \leq k \leq g$ y $1 \leq l \leq w$) que denotan si es que el jugador i juega en la posición j en el grupo k y semana l. Notar que en total tenemos $xpgw$ variables cada una con 2 valores posibles, por lo que la cardinalidad del espacio de búsqueda (hasta ahora) es de 2^{xpgw}. Las restricciones se imponen en forma de clausulas asegurando que:

- Cada jugador juega exactamente una vez a la semana:
  - Cada jugador juega por lo menos y a lo más una vez por semana
- Cada grupo consiste de exactamente p jugadores
- Dos jugadores no juegan en el mismo grupo más de una vez

A modo de nota, las clausulas establecen condiciones que deben cumplirse dentro de un conjunto de proposiciones (las restricciones de SGP, en este caso). Especifican cómo ciertas combinaciones de variables deben ser verdaderas (o falsas) para satisfacer estas proposiciones. Las siguientes clausulas se aseguran de que cada jugador juega al menos una vez a la semana:

$$\bigwedge_{i=1}^{x} \bigwedge_{l=1}^{w} \bigvee_{j=1}^{p} \bigvee_{k=1}^{g} G_{ijkl} \tag{1}$$

Para asegurarnos que cada jugador juega a lo más una vez a la semana, primero nos debemos asegurar que cada jugador juegue a lo más una vez por grupo en cada semana:

$$\bigwedge_{i=1}^{x}\bigwedge_{l=1}^{w}\bigwedge_{j=1}^{p}\bigwedge_{k=1}^{g}\bigwedge_{m=j+1}^{p} \neg G_{ijkl} \vee \neg G_{imkl} \tag{2}$$

Un segundo conjunto de clausulas se supone que debe garantizar que ningún jugador juegue en más de un grupo en cualquier semana:

$$\bigwedge_{i=1}^{x}\bigwedge_{l=1}^{w}\bigwedge_{j=1}^{p}\bigwedge_{k=1}^{g}\bigwedge_{m=k+1}^{g}\bigwedge_{n=j+1}^{p} \neg G_{ijkl} \vee \neg G_{inml} \tag{3}$$

Sin embargo, el conjunto de clausulas en 3 es incompleto; en la versión modificada se cambia por:

$$\bigwedge_{i=1}^{x}\bigwedge_{l=1}^{w}\bigwedge_{j=1}^{p}\bigwedge_{k=1}^{g}\bigwedge_{m=k+1}^{g}\bigwedge_{n=1}^{p} \neg G_{ijkl} \vee \neg G_{inml} \tag{4}$$

El cambio realizado es que n debe ir de 1 hasta p en vez de $k+1$ a p, esto debido a que un jugador que juega en la semana l en la posición j en el grupo k no debe jugar en cualquier otra posición para futuros grupos de esa semana, no posiciones mayores a j. Tomando la modificación presentada en consideración, el conjunto de clausulas $1 \cup 2 \cup 4$ asegura que cada jugador juega exactamente una vez a la semana. Un conjunto de clausulas similares se introduce para grupos de golfistas:

$$\bigwedge_{l=1}^{w}\bigwedge_{k=1}^{g}\bigwedge_{j=1}^{p}\bigvee_{i=1}^{x} G_{ijkl} \tag{5}$$

$$\bigwedge_{l=1}^{w}\bigwedge_{k=1}^{g}\bigwedge_{j=1}^{p}\bigwedge_{i=1}^{x}\bigwedge_{m=i+1}^{x} \neg G_{ijkl} \vee \neg G_{mjkl} \tag{6}$$

Notar que 6 también es una modificación a las restricciones originales. El conjunto $5 \cup 6$ tiene como intención producir grupos validos, es decir, grupos donde exactamente un jugador está en la posición j para cada $1 \leq j \leq p$.

En el modelo original se introducen variables auxiliares G'_{ikl} ($1 \leq i \leq x$, $1 \leq k \leq g$ y $1 \leq l \leq w$), que denotan si el jugador i juega en el grupo k y semana l. Están relacionadas con las variables G_{ijkl} mediante la equivalencia:

$$G'_{ikl} \leftrightarrow \bigvee_{j=1}^{p} G_{ijkl} \tag{7}$$

Bajo estas nuevas variables, la cardinalidad del espacio de búsqueda ahora es de $2^{xpgw} \cdot 2^{xgw} = 2^{xgw(1+p)}$. En el paper se presentan varias maneras de reducir el tiempo de ejecución, entre ellas:

- Reducir el número de variables.
- Reducir la cantidad de clausulas.
- Imponer restricciones que rompen simetrías.

El modelo original tenía cerca de quince restricciones extra que no eran necesarias y solo complicaban el modelo. Para mejorarlo, se propone una diferente forma de codificar la socialización deseada que se encuentra en el núcleo de SGP. El conjunto de clausulas propuesto pueden ser descritas como:

$$\bigwedge_{l=1}^{w}\bigwedge_{k=1}^{g}\bigwedge_{m=1}^{x}\bigwedge_{n=m+1}^{x}\bigwedge_{k'=1}^{g}\bigwedge_{l'=l+1}(\neg G'_{mkl}\vee\neg G'_{nkl})\vee(\neg G'_{mk'l'}\vee\neg G'_{nk'l'}) \tag{8}$$

Esto significa que si dos jugadores m y n juegan en el mismo grupo k de una semana l, entonces ellos no pueden jugar juntos en ningún grupo de las semanas que siguen. Los siguientes cuadros comparativos muestran el rendimiento del modelo original (figura 9) y el modelo modificado (figura 10) con instancias $5-3-w$ y $8-4-w$.

Table 1 Revised formulation by Gent and Lynce

Instance	#Vs.	#Cl.	Walksat	SATO
5–3–1	930	6105	0.00s	0.01s
5–3–2	1755	12210	0.02s	0.02s
5–3–3	2580	18315	–	–
5–3–4	3405	24420	–	–
5–3–5	4230	30525	–	–
5–3–6	5055	36630	–	–
8–4–1	5744	52928	–	0.06s
8–4–2	10992	105856	–	0.11s
8–4–3	16240	158784	–	–
8–4–4	21488	211712	–	–
8–4–5	26736	264640	–	–
8–4–6	31984	317568	–	–

Figura 9: Rendimiento del modelo SAT original

Table 2 Our formulation

Instance	#Vs.	#Cl.	Walksat	SATO
5–3–1	300	3480	0.00s	0.12s
5–3–2	600	9585	0.00s	0.01s
5–3–3	900	18315	0.00s	0.01s
5–3–4	1200	29670	0.01s	0.03s
5–3–5	1500	43650	2.42s	0.04s
5–3–6	1800	60255	98.94s	–
8–4–1	1280	33088	0.00s	0.03s
8–4–2	2560	97920	0.01s	0.10s
8–4–3	3840	194496	0.05s	1.16s
8–4–4	5120	322816	0.75s	–
8–4–5	6400	482880	0.98s	–
8–4–6	7680	674688	198.92s	–

Figura 10: Rendimiento del modelo SAT modificado

Como se puede observar, la formulación modificada tiene muchas menos variables para cada instancia, aunque para algunos casos la cantidad de restricciones es mayor. La columna Walksat corresponde al tiempo promedio en segundos hasta que se encuentra una solución factible en 10 intentos. La columna SATO muestra la cantidad de segundos hasta que se encuentra una solución cualquiera, promediada en más de 10 ejecuciones para reducir la varianza de los tiempos de ejecución medidos. El símbolo ”significa que no se encontraron soluciones en el tiempo límite. El modelo modificado (no así el original) pudo encontrar soluciones a todas las instancias dadas, lo que hace claro que las modificaciones realizadas pueden resultar en una mejora en rendimiento para resolver SGP.

Un modelo parcialmente booleano (PB) propuesto en [16], donde se afirma que la restricción de socialización es $\mathcal{O}(g^4p^2w^2)$ lo cual para un número grande de jugadores se encontró que únicamente generar las restricciones es lento. Por esto mismo, se optó por una formulación PB, la cual asume que no necesariamente encuentra una solución óptima, si es que dejó de hacer progreso, al menos encontrará una solución localmente óptima. Recordemos del estado del arte que el propósito de esta investigación era organizar un torneo de Mahjong, donde hay que considerar otro tipo de generalizaciones (en este caso la orientación del viento, turnos en la mesa, rotaciones de dealers, entre otros). Se presenta una codificación monolítica parcialmente booleana, donde se considera que $n = gp$ como el número de jugadores. Las restricciones usan las siguientes variables binarias, donde $h, i, j \in [1, n]$ con $j > i$, $h \neq i$ y $h \neq j$, $k \in [1, g]$, $l \in [1, w]$ y $s \in [1, p]$:

- P_{ikl}: 1 si i juega en el grupo k en la sesión l, 0 e.o.c
- S_{ikls}: 1 si i juega en el grupo k en la sesión l en la posición de asiento s, 0 e.o.c
- M_{ijl}: 1 si i y j se encuentran en la sesión l, 0 e.o.c
- C_{ij}: 1 si i y j compiten entre ellos en alguna sesión, 0 e.o.c
- D_{ijh}: 1 si ambos i y j compiten (indirectamente) contra h, 0 e.o.c

Luego, para la cardinalidad del espacio de búsqueda se tiene $2^{ngw} \cdot 2^{ngwp} \cdot 2^{\frac{n(n-1)w}{2}} \cdot 2^{\frac{n(n-1)}{2}} \cdot 2^{\frac{n(n-1)(n-2)}{2}} = 2^{ngw(1+p)} \cdot 2^{\frac{n(n-1)}{2}(w+n-1)}$. Cada potencia de 2 de la primera parte corresponde al espacio de búsqueda de cada variable según su orden de aparición en el listado anterior. Para P_{ikl} se tendrán ngw variables, mientras que para S_{ikls} serían $ngwp$. Para el resto hay que usar combinatorias, en M_{ijl} sabemos que $j > i$ (aprovechando la simetría esto ayuda a reducir el espacio de búsqueda), por ende tenemos $\binom{n}{2} = \frac{n!}{(n-2)!2!} = \frac{n(n-1)(n-2)!}{(n-2)!2} = \frac{n(n-1)}{2}$ y agregándole la sesión l tenemos un total de $\frac{n(n-1)w}{2}$ variables. Para C_{ij} seguimos el mismo procedimiento anterior (menos las sesiones) y llegaremos a un total de $\frac{n(n-1)}{2}$ variables. En D_{ijh} podemos aprovechar el mismo resultado, como $h \neq i$ y $h \neq j$ se tienen $n-2$ posibilidades, por lo que hay $\frac{n(n-1)(n-2)}{2}$ variables.

Para las direcciones del viento se codifica Este como 1, Sur como 2, etc. Las restricciones planteadas en el paper son las siguientes:

Cada jugador juega en exactamente un grupo en cada sesión:

$$\sum_i P_{ikl} = p \quad \forall k, l \tag{9}$$

Opcionalmente, para romper simetrías, ordenar a los jugadores secuencialmente en la primera sesión:

$$P_{i\lceil i/g \rceil 1} = 1 \quad \forall i \tag{10}$$

Si i y j juegan en el mismo grupo en la misma sesión, entonces deben encontrarse en esa sesión:

$$-P_{ikl} - P_{jkl} + M_{ijl} \geq -1 \quad \forall i, j, k, l \tag{11}$$

Además deben encontrarse a lo más una vez en todas las sesiones:

$$\sum_l -M_{ijl} \geq -1 \quad \forall i, j \tag{12}$$

i y j compitieron solo si jugaron en el mismo grupo en cualquier sesión:

$$-C_{ij} + \sum_{k,l} P_{ikl} P_{jkl} \geq 0 \quad \forall i, j \tag{13}$$

i y j compitieron indirectamente vía h solo si ambos compitieron con h:

$$C_{\text{mín}(i,h)\,\text{máx}(i,h)} + C_{\text{mín}(j,h)\,\text{máx}(j,h)} - 2D_{ijh} \geq 0 \quad \forall i, j, h \tag{14}$$

i y j deben competir directamente, o competir indirectamente d veces (d configurable):

$$d \cdot C_{ij} + \sum_{h} D_{ijh} \geq d \quad \forall i, j \tag{15}$$

Cada jugador debe jugar en cada grupo a lo más una vez en todas las sesiones:

$$\sum_{l} -P_{ikl} \geq -1 \quad \forall i, k \tag{16}$$

Si i se sienta en una posición en un grupo, debe jugar en ese grupo:

$$-S_{ikls} + P_{ikl} \geq 0 \quad \forall i, k, l, s \tag{17}$$

Si i juega en un grupo, debe sentarse en alguna de sus posiciones:

$$-P_{ikl} + \sum_{s} S_{ikls} \geq 0 \quad \forall i, k, l \tag{18}$$

Exactamente un jugador debe sentarse en cada puesto:

$$\sum_{i} S_{ikls} = 1 \quad \forall k, l, s \tag{19}$$

Cada jugador debe jugar en cada posición (aproximadamente) el mismo número de veces:

$$\begin{aligned} \sum_{k,l} S_{ikls} &\geq \lfloor w/p \rfloor \quad \forall i, s \\ \sum_{k,l} -S_{ikls} &\geq -\lceil w/p \rceil \quad \forall i, s \end{aligned} \tag{20}$$

Las restricciones 9 a 13 provienen de otro paper mencionado, donde el objetivo es maximizar la socialización entre jugadores ([18]), mientras que el resto son originales de la investigación de [16]. Tambien se advierte que, en la práctica, no siempre va a ser posible satisfacer todas las restricciones simultaneamente, ya sea porque no existe solución, o porque el solver no puede encontrar uno. En estos casos, las restricciones 12, 15, 16 o 20 se pueden volver blandas.

Para codificar una restricción para el balance del viento se usan variables W_{ils} que son 1 solamente si el jugador i se encuentra en la posición s en la sesion l. Esto agranda la cardinalidad del espacio de búsqueda en 2^{nwp} veces. Las restricciones son como sigue:
Cada jugador debe tener un asiento:

$$\sum_{s} W_{ils} = 1 \quad \forall i, l \tag{21}$$

Exactamente un jugador en un grupo puede tomar cada asiento:

$$\sum_{\{i\,|P_{ikl}\}} W_{ils} = 1 \quad \forall k, l, s \tag{22}$$

Cada jugador debe jugar en cada posición (aproximadamente) la misma cantidad de veces:

$$\begin{aligned} \sum_{l} W_{ils} &\geq \lfloor w/p \rfloor \quad \forall i, s \\ \sum_{l} -W_{ils} &\geq -\lceil w/p \rceil \quad \forall i, s \end{aligned} \tag{23}$$

Los resultados de este modelo se presentaron al final de la sección anterior.

Generalmente el problema siempre viene con una instancia en que la cantidad de semanas en las que dura el torneo son fijas. Aunque hay otros casos en donde se quiere saber hasta cuándo se podría extender el juego tenemos que tener claro que tipo de restricciones entran en juego. Por ejemplo en la instancia original del problema $(8-4-w)$ [10] menciona que la cantidad de semanas no puede ser más de 10 ya que un jugador se tendría que emparejar con 33 jugadores para hacer esto posible, lo cual sabemos que no lo es pues la instancia solo tiene 32 jugadores.

5. Representación

En este caso particular es conveniente usar una matríz tridimensional que guarde la organización de los grupos a través de las semanas, algo similar a lo siguiente:

	Week 1				Week 2				Week 3				Week 4				Week 5			
Group 1	0	1	2	3	0	4	8	12	0	5	10	15	0	6	11	13	0	7	9	14
Group 2	4	5	6	7	1	5	9	13	1	4	11	14	1	7	10	12	1	6	8	15
Group 3	8	9	10	11	2	6	10	14	2	7	8	13	2	4	9	15	2	5	11	12
Group 4	12	13	14	15	3	7	11	15	3	6	9	12	3	5	8	14	3	4	10	13
Group 5	16	17	18	19	16	20	24	28	16	21	26	31	16	22	27	29	16	23	25	30
Group 6	20	21	22	23	17	21	25	29	17	20	27	30	17	23	26	28	17	22	24	31
Group 7	24	25	26	27	18	22	26	30	18	23	24	29	18	20	25	31	18	21	27	28
Group 8	28	29	30	31	19	23	27	31	19	22	25	28	19	21	24	30	19	20	26	29

Figura 11: Representación de una configuración, sacada de [10]

Dentro de nuestro código, esto se verá como sale en la figura 12.

Se eligió esta representación ya que era lo más fácil para tratar con colisiones de jugadores (como repeticiones de grupos, jugadores que se repiten dentro de la misma semana, entre otros). Cabe recalcar que también tenemos un listado que enumera los jugadores desde el 0 hasta el $g \cdot p - 1$ que nos será de útilidad para los algoritmos implementados.

6. Descripción del algoritmo

6.1. Construcción

Usamos una combinación de una heurística greedy basada en el concepto de grado de libertad (visto anteriormente en [10]) con otras 2 propuestas que también dan buenos resultados. Se

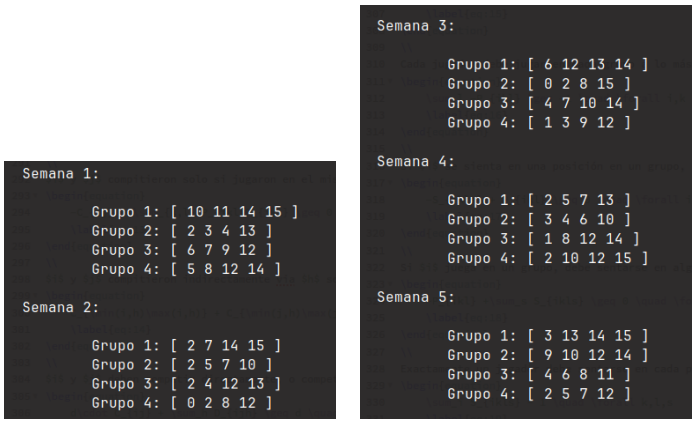
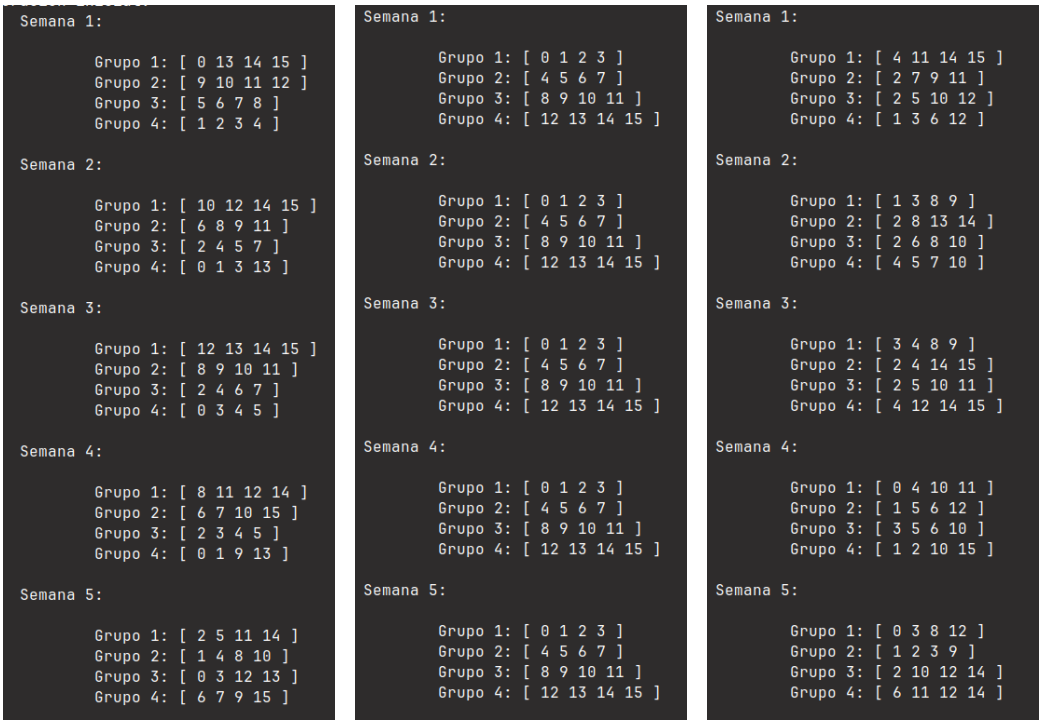


Figura 12: Representación de una configuración en el código propuesto. La instancia mostrada es la 4-4-5.

proponen 3 algoritmos greedy para hacer la construcción de una solución inicial; en el primero, iteramos sobre las semanas y grupos dentro de cada semana, para cada grupo queremos insertar el par de jugadores que tengan el mayor grado de libertad posible. Una vez seleccionados estos jugadores se les agrega a una lista de pares restringidos, la cuál nos sirve para no generar la misma tupla de jugadores en semanas distintas, además de ser agregados cada uno por independiente a una lista de jugadores restringidos para los grupos de esa semana en particular, obligando a que un jugador juegue únicamente una vez a la semana. A este algoritmo se le agrega una componente de alietoriedad con una probabilidad $p_{insertion}$; su modo de uso es que elegimos si insertamos en el grupo actual el segundo jugador del par de jugadores elegidos en base a la máxima libertad con probabilidad $p_{insertion}$. Con probabilidad $1-p_{insertion}$ elegimos un jugador al azar del listado de jugadores disponibles. Si la cantidad de jugadores por grupo es impar, se sigue la misma estrategia hasta llegar a la última casilla donde se vuelve a aplicar la probabilidad para elegir el último jugador del grupo. En la figura 13a podemos apreciar una configuración inicial construida con esta heurística. Los otros dos algoritmos son más directos: Para el greedy aleatorio simplemente sacamos una muestra aleatoria de tamaño p del arreglo de los jugadores de la instancia en específico; aquí la idea es que se generen las mayores colisiones posibles para que la búsqueda local pueda encontrar buenas soluciones. El otro algoritmo propuesto es un greedy a fuerza bruta; iteramos en cada una de las semanas y en los grupos de la semana en específico. A cada grupo le vamos asignando los jugadores enumerados de mayor a menor, repitiéndose los mismos grupos durante todas las semanas. La gracia de este último es que aunque venga con colisiones entre grupos de distintas semanas, cada jugador juega una sola vez y le da un buen punto de partida a la búsqueda local (como veremos más adelante, se obtienen buenos resultados para este). Las figuras 13b y 13c muestran configuraciones iniciales obtenidas con fuerza bruta y aleatoriamente, respectivamente.

6.2. Búsqueda local

Para búsqueda local implementamos un algoritmo Hill Climbing tanto con mejor mejora como con alguna mejora. La implementación de cada uno es bastante simple en lo que es el algoritmo en sí: Para mejor mejora, generamos el vecindario de la solución actual (en la siguiente sección entraremos en más detalles sobre cómo se construye el vecindario) y después evaluamos cada uno de los vecinos. Si la mejor evaluación encontrada es mejor o igual a la que ya teníamos la nueva solución será la que tuvo mejor evaluación entre todos los vecinos. El proceso se repite hasta que alcancemos el límite de iteraciones. Para alguna mejora la idea es básicamente la



(a) Configuración obtenida usando grado de libertad.

(b) Configuración obtenida usando fuerza bruta.

(c) Configuración obtenida usando greedy aleatorio.

Figura 13: Distintas configuraciones iniciales para la instancia 4-4-5.

misma pero en vez de evaluar todo el vecindario solo llego hasta aquel vecino que tiene mejor evaluación que la solución actual. La función de evaluación usada es la siguiente:

$$\#\text{ Colisiones dentro de un único grupo} \cdot g \cdot p + \#\text{ Colisiones de grupos repetidos en semanas distintas} \cdot g + \#\text{ Colisiones de jugadores dentro de la semana}$$

El primer tipo de colisión se da cuando un jugador es asignado más de una vez dentro del mismo grupo (físicamente imposible, por lo que la penalización es más dura); el segundo tipo de colisión hace referencia a aquellos grupos de jugadores que se repiten en semanas distintas (muy útil para guiar la búsqueda cuando la solución inicial se generó aleatoriamente). El tercer y último tipo se explica por sí mísmo: hace referencia a la restricción de que un jugador juega a lo más una vez por semana, si este aparece más de una vez dentro de esta se deben ir sumando la cantidad de veces que el jugador aparece en la misma semana.

Vale la pena mencionar que el algoritmo también tiene soporte para realizar restarts; cuando se hace un restart se utiliza el mismo algoritmo constructor para generar una nueva solución. Cabe recalcar que hacer uso de restart en fuerza bruta no tiene mucho sentido ya que esta no tiene una componente aleatoria que pueda hacer que generemos distintas soluciones iniciales. En la siguiente sección discutiremos de qué nos sirve hacer restart aunque se haya escogido la heurística por fuerza bruta.

6.3. Generación de vecindario

Para generar el vecindario de la búsqueda local se tomo en consideración un enfoque más estocástico. Se introduce otro parámetro que nos permite elegir la probabilidad de hacer un swap entre jugadores distintos **en la misma semana** con una probabilidad p_{swap}. Con probabilidad

$1 - p_{swap}$ hacemos un swap entre jugadores distintos de semanas distintas, lo que nos permite una gama más amplia de vecinos, aunque al ser un proceso basado en probabilidades es más difícil controlar la calidad de las soluciones en las que nos estámos moviendo. Esta es también la razón por la cuál hayan casos en los que conviene hacer restarts aunque la técnica constructora sea a fuerza bruta: igualmente podemos obtener vecinos distintos.

7. Experimentos

Los experimentos fueron realizados en un notebook Acer Aspire 5 el cuál tiene un procesador Intel Core i3-10110U de 4.1 GHz y una memoria RAM de 11.775 MiB, el sistema operativo es Ubuntu 22.04 LTS. Se trató de encontrar la mejor configuración de parámetros para cada una de las instancias con un script de python que usa múltiples threads para ejecutar (de fondo) el programa con distintas combinaciones de parámetros para una instancia dada. Se determino un promedio de 20 parámetros generados aleatoriamente por el script mediante prueba y error y limitaciones de la máquina (ver instrucciones de ejecución en README). Esto se hizo de esta forma para poder aprovechar al máximo la capacidad computacional disponible. Al final de la ejecución de una instancia en específico se da a conocer el porcentaje de mejora de la solución después de la búsqueda local, por lo que con este script se busca maximizar este porcentaje al 100 %.

8. Resultados

En la siguiente tabla podemos ver, por cada instancia disponible, el mayor porcentaje de mejora alcanzado en conjunto a sus respectivos parámetros; el porcentaje de mejora se calcula en base a $(eval_i - eval_{HC})/eval_i \cdot 100$, es decir, que tanto disminuyó la evaluación de la configuración después de aplicar Hill Climbing, el valor ideal es de un 100 % ya que se espera que no hayan colisiones en la configuración final. La cantidad de restarts e iteraciones necesarias para hacer un restart se dejaron en 0 ya que usualmente no ayudan mucho a mejorar la solución (en muchos casos la empeoran).

Instancia	# de iteraciones	Modo de HC	Modo Greedy	p_{swap}	$p_{insertion}$	Mejora (%)
$4-2-7$	305	Alguna Mejora	Bruto	0.1552	0.2275	88.5417
$4-4-5$	468	Alguna Mejora	Bruto	0.3	0.8	100
$5-2-9$	136	Alguna Mejora	Bruto	0.2908	0.6710	91
$5-3-7$	486	Alguna Mejora	Bruto	0.3	0.8	100
$6-3-8$	480	Mejor Mejora	Bruto	0.3	0.8	100
$7-3-10$	343	Mejor Mejora	Bruto	0.3	0.8	100
$7-4-9$	219	Alguna Mejora	Bruto	0.3	0.8	100
$8-4-10$	260	Alguna Mejora	Bruto	0.3	0.8	100

Cuadro 1: Parámetros ideales para cada instancia promediados con 10 ejecuciones.

Como podemos observar todos encontraban mejores soluciones al usar un greedy a fuerza bruta antes que con los otros. En otras repeticiones del mismo experimento también aparecía el greedy con grados de libertad; esta variabilidad entre los greedy más convenientes se debe a la aleatoriedad con la que se generan estos parámetros, por lo que se recomienda jugar con la cantidad de parámetros que se generan (esto obviamente tiene un costo en computación). Para los parámetros que se muestran en la tabla, los tiempos de ejecución promedio se pueden ver en el cuadro 2.

Instancia	Tiempo de ejecución de HC (s)
$4-2-7$	1.792
$4-4-5$	4.036
$5-2-9$	1.677
$5-3-7$	8.072
$6-3-8$	16.134
$7-3-10$	25.948
$7-4-9$	12.004
$8-4-10$	23.528

Cuadro 2: Tiempos de ejecución de HC para cada instancia, promediado en 5 ejecuciones.

9. Conclusiones

Podemos concluir que SGP es un problema que ha sido muy explorado en el campo de la optimización combinatoria no solo para resolver el problema en sí sino que también para instancias o generalizaciones que pueden ser reducibles a SGP y ser resueltas con algunos de los métodos mencionados a lo largo de esta investigación. La técnica que usemos depende netamente de la formulación que se le da al problema, como vimos anteriormente a veces un modelo más complejo puede obtener mejores soluciones bajo el contexto en el que se estudia en comparación con alguna extrapolación a otros casos, donde la misma complejidad del modelo resulta ineficiente y se debe optar por opciones más simples pero con heurísticas más creativas para resolver el problema. El enfoque general que se le da a un problema de este tipo es tratar de aprovechar al máximo sus simetrías para poder hacer un modelo y técnica de resolución que use bien toda la información aportada y mejorar su rendimiento al máximo posible, lo que lo convierte en un estándar para comparar técnicas de resolución. Es claro que un enfoque de búsqueda local con una inicialización greedy es la que da mejores resultados a la hora de comparar con los otros algoritmos explorados, que a pesar de tener un peor rendimiento siguen siendo bastante buenos ya que nos dan otro punto de vista que nos puede ser útil a la hora de querer implentar soluciones innovadoras.

Se propuso una metodología para poder resolver el problema desde un enfoque un tanto más automatizado el cuál es bastante útil para poder encontrar los parámetros ideales para correr nuestro programa. La heurística por fuerza bruta es la que más predomina para poder encontrar soluciones de calidad aunque el factor de la cantidad de iteraciones y el modo en que se realiza Hill Climbing también toman peso en el porcentaje de mejora de una solución. Todos estos resultados son coherentes con respecto a las conclusiones que pudimos obtener en el estado del arte del problema.

Referencias

[1] M. Triska, *Solution Methods for the Social Golfer Problem*. 2008. dirección: https://www.dbai.tuwien.ac.at/staff/musliu/TriskaThesis.pdf.

[2] K. Liu, S. Löffler y P. Hofstedt, "Social Golfer Problem Revisited," en *Agents and Artificial Intelligence*, J. van den Herik, A. P. Rocha y L. Steels, eds., Cham: Springer International Publishing, 2019, págs. 72-99, ISBN: 978-3-030-37494-5.

[3] I. Dotú y P. Van Hentenryck, "Scheduling social golfers locally," en *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: Second International Conference, CPAIOR 2005, Prague, Czech Republic, May 31-June 1, 2005. Proceedings 2*, Springer, 2005, págs. 155-167.

[4] H. F. MacNeish, "Euler Squares," *Annals of Mathematics*, vol. 23, n.º 3, págs. 221-227, 1922, ISSN: 0003486X. dirección: http://www.jstor.org/stable/1967920 (visitado 24-05-2024).

[5] M. Pavone, "On the seven non-isomorphic solutions of the fifteen schoolgirl problem," *Discrete Mathematics*, vol. 346, n.º 6, pág. 113 316, 2023, ISSN: 0012-365X. DOI: https://doi.org/10.1016/j.disc.2023.113316. dirección: https://www.sciencedirect.com/science/article/pii/S0012365X2300002X.

[6] F. Salassaa y F. Della Croceb, "The Social Doubles Tournament Problem," en *Proceedings of MathSport International 2017 Conference*, 2017, pág. 320.

[7] T. Limpanuparb, S. Datta, P. Tawornparcha y K. Chinsukserm, "ACAD-Feedback: Online Framework for Assignment, Collection, Analysis, and Distribution of Self, Peer, Instructor, and Group Feedback," *Journal of Chemical Education*, vol. 98, n.º 9, págs. 3038-3044, 2021. DOI: 10.1021/acs.jchemed.1c00424. eprint: https://doi.org/10.1021/acs.jchemed.1c00424. dirección: https://doi.org/10.1021/acs.jchemed.1c00424.

[8] A. Miller, M. Barr, W. Kavanagh, I. Valkov y H. C. Purchase, "Breakout Group Allocation Schedules and the Social Golfer Problem with Adjacent Group Sizes," *Symmetry*, vol. 13, n.º 1, 2021, ISSN: 2073-8994. DOI: 10.3390/sym13010013. dirección: https://www.mdpi.com/2073-8994/13/1/13.

[9] R. Lambers, L. Rothuizen y F. C. R. Spieksma, "The Traveling Social Golfer Problem: The Case of the Volleyball Nations League," en *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, P. J. Stuckey, ed., Cham: Springer International Publishing, 2021, págs. 149-162, ISBN: 978-3-030-78230-6.

[10] M. Triska y N. Musliu, "An effective greedy heuristic for the social golfer problem," *Annals of Operations Research*, vol. 194, n.º 1, págs. 413-425, 2012.

[11] C. Cotta, I. Dotú, A. J. Fernández y P. Van Hentenryck, "Scheduling Social Golfers with Memetic Evolutionary Programming," en *Hybrid Metaheuristics*, F. Almeida, M. J. Blesa Aguilera, C. Blum et al., eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, págs. 150-161, ISBN: 978-3-540-46385-6.

[12] M. Triska y N. Musliu, "Solving the social golfer problem with a GRASP," en *Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling,(PATAT 2008). Springer, Montréal*, 2008.

[13] N. Barnier y P. Brisset, "Solving Kirkman's schoolgirl problem in a few seconds," *Constraints*, vol. 10, págs. 7-21, 2005.

[14] T. Fahle, S. Schamberger y M. Sellmann, "Symmetry Breaking," en *Principles and Practice of Constraint Programming — CP 2001*, T. Walsh, ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, págs. 93-107, ISBN: 978-3-540-45578-3.

[15] J. E. Hopkroft y R. M. Karp, "An n5/2 algorithm for maximum matching in bipartite graphs," *SIAM J. Comput*, vol. 2, págs. 225-231, 1973.

[16] M. M. Lester, "Scheduling Reach Mahjong Tournaments Using Pseudoboolean Constraints," en *Theory and Applications of Satisfiability Testing – SAT 2021*, C.-M. Li y F. Manyà, eds., Cham: Springer International Publishing, 2021, págs. 349-358, ISBN: 978-3-030-80223-3.

[17] M. Triska y N. Musliu, "An improved SAT formulation for the social golfer problem," *Annals of Operations Research*, vol. 194, n.º 1, págs. 427-438, 2012.

[18] W. Harvey, *CSPLib Problem 010: Social Golfers Problem*, C. Jefferson, I. Miguel, B. Hnich, T. Walsh e I. P. Gent, eds., http://www.csplib.org/Problems/prob010.