Inteligencia Artificial

Informe Final: Problema OSAP

Diego Esteban Rosales León

December 1, 2022

Evaluación

Mejoras 2da Entrega (10%):	_____
Código Fuente (10%):	_____
Representación (15%):	_____
Descripción del algoritmo (20%):	_____
Experimentos (10%):	_____
Resultados (10%):	_____
Conclusiones (20%):	_____
Bibliografía (5%):	_____
Nota Final (100):	_____

Abstract

En las grandes organizaciones existe el desafío respecto a la distribución del espacio físico de las oficinas, cubículos, salas, entre otros, de la manera más eficiente posible. Este reto es llamado *Office Space Allocation Problem* (con sus siglas, OSAP), que consiste en asignar entidades (máquinas, personas, roles, etc.) a un grupo de habitaciones disponibles, rigiendóse por diferentes tipos de restricciones para así optimizar el uso del espacio. Este documento definirá el problema, a través de la revisión de diferentes modelos existentes que se usan para resolver este tipo de obstáculos, para finalmente formular un algoritmo de backjumping, basado en una lista de conflictos (CBJ) que permita analizar cómo se desenvuelven las variables del sistema, con el objetivo de minimizar todo espacio que esté mal distribuido, es decir, el espacio que no se esté ocupando y el que se esté sobre utilizando.

1 Introducción

En este artículo se abordará el OSAP, entendido como el problema más común y corriente dentro de cualquier organización o empresa, respecto a cómo distribuir eficientemente el espacio físico de todas sus instalaciones. Su correcta resolución da respuesta a esta optimización del espacio, impactando directamente en la productividad de la organización y en el puesto de trabajo específico de cada trabajador. Por otro lado, la finalidad u objetivo de este documento versa sobre entender detalladamente este problema en todas sus aristas, para buscar diferentes vías que puedan solucionarlo. Para lo anterior, se definirá este problema en específico, se hará una revisión exhaustiva del estado del arte hasta la fecha, se implementará un algoritmo refinado de

backjumping conocido como Conflict-Based Backjumping que no responderá a la problemática pese a sus diferentes intentos, para luego profundizar en las limitantes que llevaron a este resultado, finalizando con las conclusiones obtenidas de este análisis.

2 Definición del Problema

OSAP se define como un problema de distribución para ciertos espacios finitos en base a diferentes tipos de parámetros (recursos) [1], con el objetivo principal de minimizar el espacio mal utilizado (no utilizado y sobre utilizado) y, a su vez cumplir con restricciones o límites designados. A saber, existen dos tipos de restricciones a considerar: duras o fuertes, que son limitantes que siempre se deben satisfacer y, las blandas, que son condiciones que pueden llegar a ser un óbstaculo inamovible si no se evaden, debido a que están directamente relacionadas con la calidad del problema, es decir, entre más soluciones blandas sean infringidas, peor será la solución encontrada. La solución más factible y óptima será la que cumpla con todas las restricciones duras y eluda cabalmente las restricciones blandas, ya que así se asegurará el funcionamiento y calidad del algoritmo, sin ningún tipo de penalización en la función objetivo. Para lograr lo anterior, se utilizarán diferentes variables, tales como (1) total de entidades (personas, máquinas, etc.), (2) cantidad de pisos y habitaciones y (3) las restricciones anteriormente mencionadas. Además, es posible observar diferentes dificultades, como la obligación de que toda entidad sea ubicada en una habitación, hayan distancias específicas entre cada una o que se tengan que agrupar bajo ciertos criterios, entre otras. Un modelo clásico para resolver el OSAP, está relacionado al método de la búsqueda local [6], al que en versiones posteriores se le fueron introduciendo otras variables para mejorar o modificar su eficacia, como la mezcla de diferentes algoritmos o el uso Tabu Search [4]. Por otro lado, es necesario mencionar que surgieron opciones menos convencionales como la utilización de herramientas harmónicas para la búsqueda de soluciones [1] o adaptaciones del metódo Hill Climbing (ejemplo: Late Acceptance Hill Climbing) [3].

3 Estado del Arte

La búsqueda de la distribución eficaz del espacio y los objetos en un lugar determinado, no tiene fecha de origen exacta, pero es posible observarla de manera global cuando las personas compran muebles, adornan su casa, designan habitaciones a diferentes integrantes de la familia, etc., puesto que cada centímetro disponible llega a ser decisivo. Esto se puede extrapolar a las empresas, organizaciones e instituciones existentes, por lo que surge de manera natural la necesidad de una herramienta tecnológica que resuelva este tema. Y por ende, se denominó este problema como OSAP en pos de aunar los esfuerzos académicos contra un enemigo común.
De lo anterior surgen diferentes métodos de búsqueda que analizan algoritmos y metodologías para buscar una solución, tales como:

1. **Late Acceptance Hill Climbing (LAHC):** Tiene un enfoque similar a Hill Climbing (HC) o a Simulated Annealing (SA). El primero obtiene una solución y la compara con otras alternativas hasta encontrar la más adecuada; el segundo utiliza el mismo proceso pero evita quedarse estático en un mínimo local. Ahora bien, LAHC se basa en lo anterior pero agrega más criterios de aceptación para la resolución del problema, lo que lo hace más completo. Sin embargo, se debe considerar que a mayor cantidad de criterios, mayor tiempo de búsqueda y comparación entre los mismos [3].

2. **Tabu Search:** Esta opción se enfoca en guardar los movimientos de búsqueda que ya fueron realizados, debido a que cualquier movimiento nuevo debe ignorar lo previamente calculado, es decir, se buscarán nuevas soluciones sin visitar las ya existentes. Si bien suena ventajoso, su mayor dificultad consiste en el ahorro de tiempo, debido a que puede lograr tener una cantidad de iteraciones.

3. **Hybrid Meta Heuristics (HMH):** Si bien la metaheurística es un método heurístico para resolver un problema, la HMH le suma la optimización del tiempo, permitiendo mejoras generales a gran escala. Su mayor óbstaculo se relaciona con lo dificultoso que es integrar ambas metodologías de manera óptima [2].

4. **Harmonic Search (HS):** Es una metaheurística de optimización que fue creada para observar el nivel de improvisación que tienen los músicos. Consiste en que hay una memoria harmónica y que guarda la solución creada, la misma esta basada en la consideración de la memoria, la improvisacion sobre los instrumentos y el tono producido [5]. Según Awadallah, Tajudín y Azmi [1], el uso de HS en OSAP, puede ser el equivalente a considerar los espacios físicos como una memoria, elegir al azar las habitaciones como improvisación y seleccionar la habitación con menos penalización de las restricciones como el tono. Si bien es una solución innovadora, su dificultad radica en la incertidumbre de su realización.

Por otro lado, dentro del estado del arte, existe un estudio, de Lopes y Girimonte [6], que hace una comparación entre estos cuatro tipos de metaheurísticas ya mencionadas: Hill Climbing (HC), Simulated Annealing (SA), Tabu Search (TS), Hybrid Meta heuristic (HMH), con el objetivo de resolver un OSAP basado en "Investigación Espacial Europea" (Europe Space Agency), donde evaluaron cada algoritmo obteniendo la siguiente tabla comparativa:

Largo	10000 iteraciones			
Algoritmo	HC	SA	TS	HMH
Espacio mal usado (f_1)	449.37	440.26	430.08	416.81
Restricciones blandas (f_2)	203.57	201.87	206.22	209.52
Total (f)	652.93	642.13	636.30	**626.33**
Largo	5000 iteraciones			
Algoritmo	HC	SA	TS	HMH
Espacio mal usado (f_1)	436.08	440.11	450.11	424.42
Restricciones blandas (f_2)	209.42	213.17	209.82	207.88
Total (f)	645.50	653.28	659.93	**632.30**
Largo	1050 iteraciones (10*n)			
Algoritmo	HC	SA	TS	HMH
Espacio mal usado (f_1)	442.36	451.93	438.87	432.68
Restricciones blandas (f_2)	242.51	231.40	219.54	244.19
Total (f)	684.87	682.32	**658.40**	676.87

Tabla 1: Resultados obtenidos de los cuatro métodos mencionados anteriormente para [6]

En los dos primeros casos de la Tabla 1 (5000 y 10000 iteraciones) se puede apreciar que HMH tiene un rendimiento superior al resto de las metaheurísticas, debido al tamaño de las iteraciones (entre más iteraciones mejor), pero, cuando se tiene una menor cantidad de iteraciones, como en el tercer caso (1050 iteraciones), HMH tiene un menor rendimiento que TS, debido a que este último converge a una mayor rapidez por sobre cualquier otro método. Mientras que HC y SA, en cualquiera de los tres casos, siempre tienen valores similares. Sin embargo, es necesario mencionar que las diferencias de desempeño no son abruptamente lejanas, por lo que para grandes escenarios HMH sobresale como mejor opción y para pequeños escenarios predomina TS.

Finalmente, no queda claro el futuro de los estudios e investigaciones respecto a una metodología de búsqueda que resuelva el OSAP, debido a la variedad de opciones y la cantidad de restricciones y factores a contemplar, por lo que sigue en vigencia el desarrollo de una herramienta de solución única.

4 Modelo Matemático

A continuación, se presentará un modelo matemático basado en la explicación de Francisco Castillo, María Cristina Riff y Elizabeth Montero [4], siendo necesario considerar que tiene un enfoque similar al de Ülker y Landa Silva [7]. Además existen alternativas de solución con métodos diferentes [1], [3] y [6] que no serán considerados por su falta de atingencia a las etapas posteriores de este paper.

4.1 Parámetros

Primero, se definen todas las variables respectivas a utilizar:

$R \rightarrow$ Conjunto de Habitaciones del edificio.
$E \rightarrow$ Conjunto de Entidades.
$J \rightarrow$ Conjunto de todas las restricciones, $J = \{as, asp, mh, dh, hnc, ady, crc, ljn, cap\}$.
$S_e \rightarrow$ Capacidad requerida de la entidad $e, \forall e \in E$.
$S_r \rightarrow$ Capacidad de la habitación $r, \forall r \in R$.
$A_{dr} \rightarrow$ Lista de las habitaciones adyacentes a $r, \forall r \in R$.
$C_{cr} \rightarrow$ Lista de las habitaciones cercanas a $r, \forall r \in R$.

Luego, se definen los parámetros para las restricciones duras y blandas:

$HC^j \rightarrow$ Conjunto de las restricciones duras $j, \forall j \in J$.
$SC^j \rightarrow$ Conjunto de las restricciones blandas $j, \forall j \in J$.

4.2 Variables

Las variables a utilizar son de dos tipos: una para la asignación de variables y la otra para medir si las restricciones blandas fueron infringidas o no.

$$x_{er} = \begin{cases} 1 & \text{si la entidad } e \text{ es asignada a la habitación } r,\ \forall e \in E, r \in R \\ 0 & \text{caso contrario.} \end{cases}$$

$$y_i^j = \begin{cases} 1 & \text{si la restricción } i \text{ del tipo } j \text{ fue infringida},\ \forall i \in |SC^j|, j \in J \\ 0 & \text{caso contrario.} \end{cases}$$

4.3 Restricciones

En [4] se realizaron generalizaciones para nueve tipos de restricciones, las que pueden ser consideradas duras o blandas, (deben ser satisfechas y es deseable que sean satisfechas, respectivamente), siendo importante minimizar esta última que es un objetivo del problema, debido a que mejorará la solución a encontrar.

4.3.1 Restricciones Duras

1. Esta restricción se tiene que cumplir y no puede ser considerada como una restricción blanda. Toda entidad e tiene que tener asignada una habitacion r:

$$\sum_{r \in R} x_{er} = 1, \forall e \in E \tag{1}$$

2. Asignación: La entidad e tiene que estar asignada a una habitación r

$$x_{er} = 1 \tag{2}$$

3. Asignación Prohibida: La entidad e no tiene que ser asignada a una habitación r.

$$x_{er} = 0 \tag{3}$$

4. Misma Habitación: Dos entidades e_1 y e_2 tienen que estar en la misma habitación.

$$x_{e_1r} = 1, x_{e_2r} = 1, \ \forall r \in R$$

$$x_{e_1r} - x_{e_2r} = 0, \ \forall r \in R \tag{4}$$

5. Distinta Habitación: Dos entidades e_1 y e_2 no tienen que estar en la misma habitación.

$$x_{e_1r} = 1 \leftarrow x_{e_2r} = 0, \ \forall r \in R$$

$$x_{e_1r} + x_{e_2r} \leq 1, \ \forall r \in R \tag{5}$$

6. Habitación no compartida: La entidad e no puede compartir habitación con ninguna otra entidad

$$\sum_{f \in \ E-e} x_{fr} \leq (|E| - 1) - (|E| - 1)x_{er}, \ \forall r \in R \tag{6}$$

7. Adyacencia: Las entidades e_1 y e_2 deben asignarse en habitaciones adyacentes.

$$x_{e_1r} \leq \sum_{s \in A_{dr}} x_{e_2s} \leq 1, \ \forall r \in R \tag{7}$$

8. Cercanía: Las entidades e_1 y e_2 deben asignarse en habitaciones cercanas.

$$x_{er} \leq \sum_{s \in C_{cr}} x_{fs} \leq 1, \ \forall r \in R \tag{8}$$

9. Lejanía: Las entidades e_1 y e_2 deben asignarse lejos la una de la otra.

$$0 \leq \sum_{s \in C_{cr}} x_{e_2s} \leq 1 - x_{e_1r}, \ \forall r \in R \tag{9}$$

10. Capacidad: La habitación r no debe superar cierta capacidad respecto a las entidades asignadas en ella. (Sobre uso)

$$\sum_{e \in E} S_e x_{er} \leq S_r \tag{10}$$

4.3.2 Restricciones Blandas

1. Asignación:

$$y_i^{as} = 1 - x_{er} \tag{11}$$

2. Asignación Prohibida:

$$y_i^{asp} = x_{er} \tag{12}$$

3. Misma Habitación:

$$y_{ir}^{mh} - 1 \leq x_{e_1r} - x_{e_2r} \leq 1 - \epsilon + \epsilon y_{ir}^{mh}, \ \forall r \in R. \tag{13}$$

$$y_i^{mh} = \sum_{r \in R} y_{ir}^{mh} \tag{14}$$

4. Distinta Habitación:

$$(1+\epsilon)-(1+\epsilon)y_{ir}^{dh} \leq x_{e_1r} - x_{e_2r} \leq 2 - y_{ir}^{dh},\ \forall r \in R \tag{15}$$

$$y_i^{dh} = \sum_{r \in R}(1 - y_{ir}^{dh}) \tag{16}$$

5. Habitación no compartida:

$$(|E|-1)(2 - x_{er} - y_{ir}^{hnc}) \geq \sum_{f \in\ E-e} x_{fr},\ \forall r \in R \tag{17}$$

$$\sum_{f \in\ E-e} x_{fr} \geq (|E|-1)(1-x_{er}) + \epsilon - (|E|-1+\epsilon)y_{ir}^{hnc},\ \forall r \in R \tag{18}$$

$$y_i^{hnc} = \sum_{r \in R}(1 - y_{ir}^{hnc}) \tag{19}$$

6. Adyacencia:

$$y_{ir}^{ady} + x_{e_1r} - 1 \leq \sum_{s \in A_{dr}} x_{e_2s} \leq x_{e_1r} - \epsilon + (1+\epsilon)y_{ir}^{ady},\ \forall r \in R \tag{20}$$

$$y_i^{ady} = \sum_{r \in R}(1 - y_{ir}^{ady}) \tag{21}$$

7. Cercanía:

$$y_{ir}^{crc} + x_{er} - 1 \leq \sum_{s \in C_{cr}} x_{fs} \leq x_{er} - \epsilon + (1+\epsilon)y_{ir}^{crc},\ \forall r \in R \tag{22}$$

$$y_i^{ljn} = \sum_{r \in R}(1 - y_{ir}^{ljn}) \tag{23}$$

8. Lejanía:

$$1 - x_{e_1r} + \epsilon - (1+\epsilon)y_{ir}^{ljn} \leq \sum_{s \in C_{cr}} x_{e_2s} \leq 2 - x_{e_1r} - y_{ir}^{ljn},\ \forall r \in R \tag{24}$$

$$y_i^{ljn} = \sum_{r \in R}(1 - y_{ir}^{ljnx|}) \tag{25}$$

9. Capacidad:

$$\sum_{e \in E} S_e x_{er} + (S_r + \epsilon)(1 - y_i^{cap}) \geq S_r + \epsilon \tag{26}$$

$$\sum_{e \in E} S_e x_{er} + \sum_{e \in E}(S_e - S_r)(1 - y_i^{cap}) \leq \sum_{e \in E} S_e \tag{27}$$

4.4 Función Objetivo

La función objetivo tiene por finalidad minimizar el mal uso del espacio (desuso y sobre uso) y la cantidad de veces que se infringen las restricciones blandas. Se debe notar que el sobre uso del espacio físico se penaliza como el doble del espacio en desuso. Para simplificar visualmente la función objetivo [4], se considerará w^j como la penalización de cada restricción blanda j, con $j \in J$. Finalmente queda:

$$Min\ z = \sum_{r \in R} max(S_r - \sum_{e \in E} x_{er}S_e,\ 2\sum_{e \in E} x_{er}S_e - S_r) + \sum_{j \in J} w^j \sum_{i=1}^{|SC^j|} y_i^j \tag{28}$$

5 Representación

Para efectos de este documento, la representación de la solución elegida viene dada por una matriz:

nRB	rb1	rb2	rb3	...	rbn		
EMU	NU	SU					
0	NU0	SU0	nE0	E01	E02	...	E0e
1	NU1	SU1	nE1	E11	E12	...	E1e
2	NU2	SU2	nE2	E21	E22	...	E2e
...	...	...	...	...	...	...	...
r	NUr	SUr	nEr	Er1	Er2	...	Ere

Tabla 2: Representación de la solución

Donde de la Tabla 2:

- nRB → Número de restricciones blandas no cumplidas
- rbn → ID restricción blanda n no cumplida
- EMU → Espacio total mal utilizado
- NU → Espacio total no usado
- SU → Espacio total sobre utilizado
- r → ID de habitación r
- NUr → Espacio no utilizado de la habitación r
- SUr → Espacio sobre utilizado de la habitación r
- nEr → Número de entidades asignadas a la habitación r
- Ere → ID de Entidad e que fue asignada a la habitación r

En la Tabla 2 se muestra una solución ficticia y provisional al problema planteado, que representa los valores posibles que se podrían considerar para dar solución al OSAP. Cabe mencionar que esta representación fue elegida debido a que es fácil de usar en la observación de cada espacio físico utilizado, no utilizado y sobre utilizado, para cada una de las habitaciones del edificio.
La técnica elegida para este caso es CBJ, una técnica de tipo completa, donde se le asigna un ID a cada una de las habitaciones para que esta información se vaya almacenando y se tenga una noción de los conflictos de espacio que ocurran entre las mismas. Es importante destacar que los resultados que se obtengan serán parciales dependiendo de la cantidad de iteraciones que se hagan hasta considerar la solución como completa.
La primera variable del modelo matemático es $x_{er} : Dom = \{0, 1\}$, cuyo objetivo es definir si una e es asignada a una habitación r, intercalando su valor entre 0 y 1, hasta revisarlos todos aunque no sean incorporados en la representación de la solución. A su vez, ocurre para la segunda variable, $y_i^j : Dom = \{0, 1\}$, donde dependiendo de si se infringió o no una restricción J, intercalando su valor entre 0 y 1, conlleva o no a una penalización. Por otro lado, es importante considerar que las dos variables anteriores son necesarias para el cumplimiento de las restricciones (1-27).

Variables de ingreso		
Entidad	**Restricciones**	**Habitaciones**
int ent_id: id de la entidad	int r_id: id de la restricción	int h_id: id de las habitaciones
int gid: id del grupo de la entidad	int r_tipo: tipo de restricción	int p_id: id del piso en el que se encuentra la habitación
float espacio_ent: espacio de la entidad	bool rdorb: 0 restricción blanda 1 restricción dura	float h_espac; espacio disponible para la habitación
	int r_1: parámetro de la restricción	vector<int>h_ady; habitaciones adyacentes a la habitación actual
	int r_2: parámetro de la restricción	

Tabla 3: Variables utilizadas para los valores del archivo de ingreso

Cabe destacar que todos los valores de la Tabla 3, son necesarios para poder resolver todas las restricciones planteadas (4-10 y 15-27). Por último, en cuánto al espacio de búsqueda asociado a OSAP, el mismo consiste en encontrar todas y cada una de las soluciones posibles que resuelvan el problema. Por el contrario, al utilizar CBJ, el espacio de búsqueda se reduce significativamente, aunque sigue siendo una gran cantidad de casos a evaluar.

6 Descripción del algoritmo

Para la resolución de OSAP, se estudiará la implementación de una técnica de búsqueda completa llamada *Conflict-Based Backjumping (CBJ)*, que cada vez que encuentra un error, hace un salto hacia atrás y vuelve a intentarlos con otras variables. Backjumping por si solo es un algoritmo que busca un valor posible dentro del dominio de las variables que no genere la creación de conflictos con los valores de las variables que ya están instanciadas. Por otro lado, CBJ es un algoritmo más refinado, ya que utiliza el algoritmo anterior y le agrega una lista de conflictos $Conf(x_i)$, donde para cada valor erróneo (inconsistente) se debe registrar en la lista la variable más temprana instanciada y en conflicto con el intento actual de instanciación. Cuando no quedan más valores por revisar de una de las instancias intentadas, existe un **fallo**, lo que conlleva a que la lista entregue las causas del problema y el punto de regreso (backjumping), que sería la variable más recientemente instanciada en $Conf(x_i)$, para que desde la misma se vuelva a hacer otra iteración.
En espeífico para este OSAP, se elige una habitación cualquiera donde se recorrerá su dominio en búsqueda de valores que sean factibles (que cumplan las restricciones), si se encuentra una asignación, se agrega y se continúa con la búsqueda hasta que no queden más instanciaciones. Ahora bien, si se acaban las iteraciones antes que las instanciaciones, pueden existir resultados parciales.

Algorithm 1 Conflict Directed Backjumping (CBJ)

1: $i \leftarrow 1$, $D_i^{'} \leftarrow D_i$, $J_i \leftarrow \emptyset$ ▷ Iniciar contador, copiar dominio, iniciar lista conflictos
2: **while** $1 \leq i \leq n$ **do**
3: $x_i \leftarrow Select\_value\_CBJ$
4: **if** $x_i = null$ **then** ▷ Ningún valor ha sido retornado
5: $i_{prev} \leftarrow i$
6: $i \leftarrow$ último ancestro en J_i ▷ Backjumping
7: $J_i \leftarrow J_{iprev} - \{x_i\}$ ▷ Unir listas de conflicto
8: **else**
9: $i \leftarrow i + 1$ ▷ Dar un paso
10: $D_i^{'} \leftarrow D_i$ ▷ Restablecer el dominio mutable
11: $J_i \leftarrow \emptyset$ ▷ Restablecer la lista de conflictos
12: **end if**
13: **end while**

Algorithm 2 Select_value_CBJ

1: **while** $D_i^{'}$ not empty **do**
2: Seleccionar un elemento (arbitrario) $\alpha \in D_i^{'}$ y removerlo de $D_i^{'}$
3: consistente $\leftarrow$ TRUE
4: k $\leftarrow 1$
5: **while** $k < i$ and consistente **do**
6: **if** CONSISTENTE $(\overrightarrow{a}_k, x_i = \alpha)$ **then**
7: $k \leftarrow k + 1$
8: **else**
9: Dejar que R_S sea la primera restricción que causa conflicto
10: Agregar las variables en el alcance S de R_S, sin contar x_i, en J_i
11: consistente $\leftarrow$ FALSE
12: **end if**
13: **end while**
14: **if** consistente **then return** α
15: **end if**
16: **end while**
17: **return** null

El algoritmo 1 es la base necesaria para trabajar con técnicas de backjumping, pero específicamente en este caso se trabaja con ancestros (habitaciones ya instanciadas). Si no se cuenta una de las soluciones o si la misma lleva a un conflicto, se agrega a la lista y luego se prosigue de manera normal. Cuando se encuentra una solución, se resta de la lista de conflictos, el dominio y se sigue avanzando en la búsqueda de más soluciones.
El Algoritmo 2 es invocado por el Algoritmo 1 y se encarga de resolver lo que es el algoritmo de CBJ. Para ello, se elige un valor α que pertenece al dominio y se elimina del mismo, para luego revisar si de un subconjunto arbitrario $(\overrightarrow{a}_k)$ existe un valor alfa en el dominio de x_i que sea consistente, de no encontrarse llamamos $\overrightarrow{a}_k$ como una lista de conflictos. En el caso, de que si sea consistente, se continua buscando más soluciones.
Estos dos algoritmos complementan la formulación de CBJ.

7 Experimentos

Se pusieron en práctica los algoritmos propuestos, pero lamentablemente los resultados fueron infructuosos y poco confiables, impactando directamente en su validez para definirlos como una solución factible. Pese lo anterior, hipotéticamente lo que se buscaba implementar consistía en ejecutar CBJ resolviendo las instancias provistas, para luego revisar las restricciones y guardar cada resultado en el formato de la Tabla 2, aún así, se debe mencionar que se está utilizando una técnica completa, por lo que el programa puede ejecutarse sin tener fin alguno, pero para poder controlar y comparar las soluciones obtenidas, se iba a utilizar diferentes cantidades de iteraciones para cada instancia como se muestra en la siguiente tabla:

Instancia	**Número de iteraciones**
1	**10**
2	**20**
3	**30**

Tabla 4: Tabla comparativa y de término del programa

Cabe decir que la experimentación hipotética, iba a ser utilizada sobre un computador con un procesador con 4 núcleos y 12GB de RAM.

8 Resultados

Al momento de hacer las pruebas experimentales y obtener resultados, los mismos no cumplieron su propósito debido a diferentes factores. A saber, el desconocimiento e inexperiencia en el uso del lenguaje de programación utilizado (C++), el problema de trabajar en ello en contra del tiempo y la dificultad inherente que conlleva la implementación de un algoritmo teórico a uno práctico, pues hasta que no se intenta por ensayo y error, no se visualizan las dificultades y necesidades de adaptación del mismo. Sin embargo, cabe mencionar que de manera teórica, los resultados esperados que se debiesen haber logrado, eran que al aumentar la cantidad de iteraciones de término a un número mayor (200 o 500 iteraciones), el resultado obtenido seria mejor, entregando una solución más robusta para el problema de OSAP.

9 Conclusiones

A modo de cierre, se concluye que OSAP es un problema que surge desde la necesidad humana de sacar el máximo provecho al espacio físico que habita, tomando en cuenta diferentes factores. En este sentido, la existencia del OSAP llevó a la creación y construcción de diferentes tipos de métodos para encontrar una solución a este problema, coincidiendo primordialmente en su objetivo de utilizar eficientemente los espacios (disminuyendo su mal uso), tomando en cuenta restricciones basales (duras y blandas) y utilizando los métodos de búsqueda como principal herramienta. A su vez, sus diferencias radican en la forma y el enfoque que utilizan para resolver el problema, por lo que, si bien resuelven diferentes aristas del mismo, aún no logran que sea a nivel único y global.

A la luz de lo anterior, se vuelve importante destacar tanto los métodos de Tabu Search como los de Meta heurística híbrida, ya que ambos se acercan a una adecuada solución dependiendo de si se trata de un bajo nivel o un alto nivel de iteraciones, respectivamente. Por lo que, gracias a sus resultados positivos, es esencial poder considerarlos como base para futuros procedimientos e investigaciones.

Por otro lado, es importante tener en cuenta que en la mayoría de los estudios revisados, se

utilizaron computadores con 4 GB de RAM o con procesadores que no pertenecen a la última generación, lo que puede significar una clara limitante para cada método desarrollado, considerando cómo afecta la velocidad y eficiencia del hardware en los mismos.
Es innegable que el OSAP es un problema interesante de resolver por el gran impacto que su resolución tendría tanto a nivel mundial (industrias y negocios) como a nivel individual (en la vida de las personas). Y aquí es relevante considerar algoritmos no revisados, aplicaciones computacionales a la vanguardia, equipos físicos con mayor capacidad o una mezcla de metodologías diferentes, que abran puertas y posibilidades a obtener una respuesta definitiva para el OSAP o que sienten bases y orientaciones diferentes para futuros investigadores.
A la luz de lo anterior, se intentó desarrollar un acercamiento al backjumping refinado CBJ, explicando sus dos algoritmos en el documento a modo de propuesta de solución. Lamentablemente, si bien en la teoría o el papel daba la impresión de que iba a funcionar, debido a diferentes limitantes (mencionadas más arriba) fue imposible ponerla en práctica y confirmar su aplicación, quedando para futuros análisis como insumo o base perfectible si se corrigen los alcances encontrados.

10 Bibliografía

References

[1] Mohammed A Awadallah, Ahamad Tajudin Khader, Mohammed Azmi Al-Betar, and Phuah Chea Woon. Office-space-allocation problem using harmony search algorithm. In *International Conference on Neural Information Processing*, pages 365–374. Springer, 2012.

[2] Christian Blum and Andrea Roli. Hybrid metaheuristics: an introduction. In *Hybrid metaheuristics*, pages 1–30. Springer, 2008.

[3] Asaju La'aro Bolaji, Ikechi Michael, and Peter Bamidele Shola. Adaptation of late acceptance hill climbing algorithm for optimizing the office-space allocation problem. In *International Workshop on Hybrid Metaheuristics*, pages 180–190. Springer, 2019.

[4] Elizabeth Montero Francisco Castillo, María-Cristina Riff. New bounds for office space allocation using tabu search. *GECCO '16: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 869–876, 2016.

[5] X. Z. Gao, V. Govindasamy, H. Xu, X. Wang, and K. Zenger. Harmony search method: Theory and applications. *Computational Intelligence and Neuroscience*, 2015:258491, Apr 2015.

[6] Rui Lopes and Daniela Girimonte. The office-space-allocation problem in strongly hierarchized organizations. In *European Conference on Evolutionary Computation in Combinatorial Optimization*, pages 143–153. Springer, 2010.

[7] Özgür Ülker and Dario Landa-Silva. A 0/1 integer programming model for the office space allocation problem. *Electronic Notes in Discrete Mathematics*, 36:575–582, 2010.