PROBLEMA DE LA ASIGNACIÓN GENERALIZADA

Marcelo A. Paciulli*

*Universidad Nacional del Noroeste de la Provincia de Buenos Aires (UNNOBA)
Junin, Bs. As., Argentina
e-mail: marcelopaciulli@nexo.unnoba.edu.ar

1 RESUMEN

El proceso de toma de decisiones respecto a la asignación de recursos con el objetivo de cubrir tareas en un determinado ámbito estriba en un problema complejo, que requiere modelos y métodos sofisticados para su resolución óptima, debido a la restricción cualitativa y cuantitativa de diversos agentes, humanos y/o materiales, entre otros, para producir resultados aceptables en tales procesos.
Nos centramos en resolver el Problema de la Asignación Generalizada o GAP (Generalised Assignment Problem), mediante la modelización matemática y posterior traducción al lenguaje Python, aplicando el método multi-start basado en metaheurística constructiva y de búsqueda local iterada, conocido como GRASP (Greedy Randomized Adaptive Search Procedure, Feo & Resende, 1989).
Si bien se incluyen algunos experimentos con el lenguaje GLPK (GNU Linear Programming Kit, paquete Open Source que engloba la programación lineal y lineal-entera mixta), el trabajo se basa en los resultados obtenidos con GRASP como guía de resolución, debido a su bajo costo computacional, sobre todo cuando los agentes y tareas a cubrir crecen gradualmente. Asimismo, es importante recalcar que existen diversas técnicas que también apuntan al campo de las heurísticas adaptativas y logran interesantes resultados[1]. Algunas de ellas son los algoritmos Tabu Search y Ant Colony Optimization (optimización por colonia de hormigas).
Las pruebas fueron llevadas a cabo mediante un conjunto de datos proporcionados por la colección dedicada OR-LIB[2], logrando encontrar, de este modo, soluciones adecuadas para el problema GAP, haciendo hincapié en un subconjunto especial de estos data sets, centrales en varios artículos de investigación[3].

2 INTRODUCCION

El problema de la asignación generalizada es uno de los tópicos fundamentales de optimización combinatoria. Consiste en encontrar la forma de asignar ciertos recursos disponibles (máquinas o personas) para la realización de determinadas tareas al menor coste.
El modelo se puede aplicar a la asignación de empleados a tareas, de fábricas a productos, postores a contratos, etc. Son casos particulares de los problemas de transporte, y constituyen la clase más sencilla de los problemas lineales, en el cual los trabajadores representan las fuentes y los puestos representan los destinos.
Los orígenes del problema de la asignación tuvieron lugar en tiempos de la revolución industrial, ya que el surgimiento de las máquinas hizo que fuera necesario asignar una tarea a un trabajador. De este acontecimiento, y en forma generalizada, se deduce la posibilidad de que una máquina (o agente, en general) realice más de un trabajo, y cada trabajo debe ser correspondido sólo a una máquina.
El objetivo principal es realizar una óptima asignación de los trabajos a los recursos existentes, intentando minimizar el coste total, o maximizar la productividad (según el criterio elegido); teniendo en cuenta las restricciones de tales recursos.
El Problema de la Asignación Generalizada se considera de complejidad NP-hard (Martello & Toth, 1981; Sahni & Gonzalez, 1976), debido a que, si bien es posible verificar que una solución sea válida o no, en un tiempo polinomial, garantizar que esa solución sea un óptimo no puede hacerse bajo las mismas condiciones.

3 EL PROBLEMA

Se conoce como Problema de la Asignación Generalizada (GAP) (Ross & Soland, 1975) a la situación en la cual es necesario destinar un conjunto $J = \{a_1, a_2, ..., a_n\}$ de actividades a otro conjunto $I = \{r_1, r_2, ..., r_m\}$ de agentes, donde i=1,...,m, y j=1,...,n.
Los parámetros que hacen al modelo matemático correspondiente son los siguientes:
a_{ij}: capacidad del agente i para realizar la tarea j.
b_i: cantidad de recursos disponibles para el agente i.
c_{ij}: coste (a minimizar) –productividad a maximizar, si fuera el caso– del agente i al asignarle un trabajo j.
Para conocer si cierto agente i se destina a una actividad j, se incorporan las variables de decisión x_{ij} (variables binarias que adoptan el valor de 1 siempre que esté asignada, y 0 en caso contrario).

Formalmente, diremos que:

$$\mathrm{x}_{ij} = 1 \Leftrightarrow \sigma(\mathrm{j})=\mathrm{i} \tag{1}$$

Donde $\sigma(j)$ representa la asignación de la actividad j a cierto agente i. Seguidamente, el coste (o en su defecto, ganancia), será en función de las asignaciones (lo que radica en la complejidad e importancia del problema, apuntando así directamente al proceso de toma de decisiones óptimo).

Modelo matemático del GAP

$$\begin{aligned} minimize \quad & \mathrm{cost}(\sigma) = \textstyle\sum_{i\epsilon I} \sum_{j\epsilon J} c_{ij} x_{ij} \\ subject \quad to \quad & \textstyle\sum_{j\epsilon J} a_{ij} x_{ij} \leq b_i, \quad \forall i \epsilon I \\ & \textstyle\sum_{i\epsilon I} x_{ij} = 1, \quad \forall j \epsilon J \\ & \mathrm{x}_{ij} \epsilon \{0,1\}, \quad \forall i \epsilon I \quad and \quad \forall j \epsilon J \end{aligned}$$

La primera restricción significa que el conjunto de tareas realizadas por un agente no excederá el límite de recursos que el mismo tiene disponible. La segunda, indica que las tareas serán realizadas únicamente por un agente.

La flexibilidad del algoritmo GAP lo hace idóneo para el problema de la maximización de beneficios, si se cuenta con una matriz de ganancias que recoja la información resultante de asignar la actividad j al agente i, $profit_{ij}$, modificando al modelo sólo en la expresión:

$$maximize \quad \Gamma(\sigma) = \textstyle\sum_{i\epsilon I} \sum_{j\epsilon J} profit_{ij} x_{ij}$$

4 SOLUCION PROPUESTA

El algoritmo metaheurístico implementado en Python (que combina o integra diferentes estrategias de construcción y posterior búsqueda, para explorar el espacio de soluciones), es GRASP[4], por su

capacidad de construir una buena solución de manera iterativa, agregando elementos que hacen a la calidad de la misma.
Primeramente, se genera una solución factible, un arreglo de filas y columnas binarias, donde cada columna tiene sólo un 1, (el resto son 0). Esto hace a la restricción segunda. Aunque, es importante afirmar que la generación al azar de este arreglo no resultó beneficioso, ya que la primer restricción le da una probabilidad casi nula a la construcción aleatoria de soluciones factibles, de modo que se pensó en construir el arreglo inicial basándonos en ambas restricciones al mismo tiempo, y sobre ello recae una segunda parte aleatorizada.
Sobre lo anterior, se trabajó en métodos que generan parcialmente la matrix (empezando por un arreglo de ceros, recorrido por columnas, secuencial y, mediante un valor aleatorio de fila, asigno un valor 1, con la consecuencia que, si al colocar ese 1 allí, deja de cumplir la condición de capacidad del agente i -esto es, se excedió el límite de capacidad de ese agente-fila, puede que quede vacía tal columna, ya que no se hará la asignación), y de esta manera le damos la posibilidad a otro método de *rellenar* las columnas vacías atendiendo a la primer restricción para, ahora sí, retornar la solución inicial factible.
Haciendo un stop, cabe aclarar que, al contar con un fuerte componente aleatorizado, debemos asegurarnos de generar tal factibilidad, desde ya, y esto puede no ser trivial; aunque el algoritmo ha respondido de manera robusta en las pruebas. Subrayamos también que esta forma de construcción inicial tiene una contribución importante, que le da mayor flexibilidad de resolución al problema, segun explicaremos más adelante.
Lo fundamental en el algoritmo GRASP, que hace a su núcleo de funcionamiento, es la capacidad de encontrar elementos en una solución factible que generen menor costo (o mayor ganancia), al incorporarlos a la misma, sin que se pierda factibilidad. Para ello, se decidió encontrar esos posibles elementos (columnas, en la resolución), que le den un poder de guía mayor al método, y explorar así más eficientemente el espacio de búsqueda.
Tales elementos son generados por *build_seed()*, que tiene la capacidad de intercambiar dos columnas entre sí una determinada cantidad de veces y, además, *desplazar* el 1 en alguna columna aleatoria. Al mismo tiempo, se comprueba la factibilidad, y se guardan los elementos correspondientes, con sus respectivas posiciones de mínimo coste o mayor ganancia. También, es importante decidir si se utilizó la primer herramienta (intercambio de dos columnas entre sí, aleatorias) o la segunda (desplazo del 1, una columna). Aclaramos que *build_seed()* elige en modo random qué herramienta aplicar, lo que puede conllevar mejoras al proceso de búsqueda, como nombramos más arriba.

El procedimiento se muestra en la tabla siguiente

Solución factible, costo f(a) = 450

0	1	0	0
1	0	1	0
0	0	0	1
0	0	0	0
C1	C2	C3	C4

Solución Inicial
a : asignación
f: función costo

Solución factible, costo f(a)=458

0	1	0	0
0	0	1	1
1	0	0	0
0	0	0	0
C4	C2	C3	C1

Luego de aplicar build_seed(), con el método de intercambio de dos columnas entre si, columnas 1 y 4, respectivamente, el costo aumentó, no sirve

Solución factible, costo f(a) = 420

0	1	0	0
1	0	0	0
0	0	1	1
0	0	0	0
C1	C2	C3	C4

build_seed() eligió al azar el método de desplazo de un 1 en una columna aleatoria, la 3 en este caso, y el costo disminuyó. Guardamos el elemento columna C3 nuevo en la posición 3, que es donde minimiza al incorporarlo a la solución inicial

Una vez que contamos con un conjunto de *buenos* elementos, podemos restringir los mejores de ellos en una lista, de acuerdo a un rango de valores que manejaremos con un parámetro, fijo a lo largo de toda la ejecución. El valor de *alfa* exige un costo mínimo (o ganancia máxima) necesario para poder acceder a la lista de tales elementos restringidos (RCL), con lo cual, será de utilidad conocer el valor que este parámetro tomará en cada ejecución.
Este tipo de método de construcción *greedy* se conoce, en la literatura, como una *heurística semi-codiciosa*, descrita por primera vez en Hart y Shogan (1987).

Algoritmo GRASP

```
procedimiento GRASP(Max_Itera,Semilla)
for k = 1, ..., Max_Itera do
    s = Greedy Randomized Construction(Semilla)
    s = búsqueda local (s)
    actualiza solución(s,Mejor Solución)
return Mejor Solución
```

En la fase de construcción se encuentra una solución factible cuya vecindad es investigada hasta llegar a un mínimo local. La mejor solución es guardada.

Procedimiento Greedy Randomized Construction

```
procedimiento Greedy Randomized Construction(Semilla)
s ← 0
Inicializa el conjunto candidato: C ← E
Evalua el costo incremental c(e)∀e ∈ C
while C ≠ 0 do
    c^min ← min{c(e)|e ∈ C}
    c^max ← max{c(e)|e ∈ C}
    RCL ← {e ∈ C|c(e) ≤ c^min + α(c^max − c^min)}
    selecciona un elemente t de RCL aleatoriamente
    s ← s ∪ {t}
    actualiza el conjunto candidato C
    re-evalua los costos incrementales c(e)∀e ∈ C
return s
```

Con $\alpha = 0$ se vuelve un algoritmo completamente greedy, mientras que con $\alpha = 1$ se vuelve una estrategia aleatoria (considerando la minimización, ya que si quisiéramos maximizar beneficios, con cierta modificación, las elecciones se invierten).
Este parámetro sirve para establecer un balance entre costo computacional y calidad de las soluciones.
El elemento que se incorpora en la construcción de la solución, se selecciona aleatoriamente dentro de RCL. Una vez incorporado, se actualiza la lista de candidados y se re-evaluan los costos incrementales.
El algoritmo de búsqueda local puede ser con el mejor candidato (best-improvement) o el primer candidato (first-improvement) que mejore la solución actual.
Algunas extensiones se ocupan principalmente por como controlar los posibles valores de α.

- Dejar un valor fijo.
- Variar de acuerdo a la calidad de las soluciones, por ejemplo, dentro de un rango de valores y favoreciendo con el tiempo aquellos valores que dan mejores resultados.
- Seleccionar aleatoriamente dentro de un cierto rango.
- Seleccionar aleatoriamente dentro de una distribución decreciente y no uniforme.

También se han hecho extensiones para no seleccionar el elemento de RCL aleatoriamente, sino siguiendo ciertas preferencias de acuerdo al lugar que ocupan.

Por otro lado, se han hecho combinaciones o mejoras usando *path relinking*

5 PRUEBAS

La tarea de encontrar óptimos locales que son buena aproximación (no mayores a un 3% de error, a bajo costo computacional) de aquellos valores globales que resuelven cada ejercicio, se ha vuelto eficiente al correr el algoritmo GRASP.
Las tablas siguientes indican los valores óptimos para determinadas instancias, con tiempo de ejecución promedio en la obtención y, luego, la estimación dada por nuestro algoritmo en cada una de ellas.

Table 2. Computational results for 'large-sized' problems

Prob type	Size m	Size n	Best sol'n in each of the 10 trials										Best Overall sol'n	Avg. Best Sol'n time	Avg. Exec. time
A	5	100	o	o	o	o	o	o	o	o	o	o	1698	0.5	435.0
		200	o	o	o	o	o	o	o	o	o	o	3235	0.3	898.4
	10	100	o	o	o	o	o	o	o	o	o	o	1360	0.3	409.4
		200	o	o	o	o	o	o	o	o	o	o	2623	17.0	1213.0
	20	100	o	o	o	o	o	o	o	o	o	o	1158	0.4	737.0
		200	o	o	o	o	o	o	o	o	o	o	2339	43.4	1687.8
B	5	100	1847	b	1847	1849	1846	1848	1852	1859	1860	b	1843	126.9	288.2
		200	3563	3570	3563	3559	3566	3555	3567	b	3565	3566	3553	439.5	790.0
	10	100	b	b	b	1409	b	b	1409	1409	1409	1409	1407	30.1	276.0
		200	2842	2838	2833	2836	2835	2836	2844	2835	b	2837	2831	608.4	1027.4
	20	100	1168	1167	1168	b	1167	1167	b	b	1167	b	1166	191.5	617.3
		200	2341	2341	2344	2341	2341	2341	2343	2341	2341	b	2340	518.5	1323.5
C	5	100	b	1938	1940	1939	1940	1943	1942	b	1942	1937	1931	139.1	302.4
		200	b	3461	3463	3461	3465	3460	3466	3468	3467	3470	3458	531.2	810.1
	10	100	1409	b	b	b	1405	b	1412	1407	1412	b	1403	170.6	394.2
		200	2822	2821	2820	2818	2821	2826	2822	2816	b	2815	2814	628.6	1046.0
	20	100	1247	1254	1255	1249	1247	1247	1251	1249	1251	b	1244	279.9	669.3
		200	2401	b	2406	2410	2402	2409	2404	2412	2409	2408	2397	1095.9	1792.3
D	5	100	6379	6406	6409	6397	6415	b	6388	6391	6406	6384	6373	369.9	530.3
		200	12869	12825	b	12801	12826	12816	12827	12835	12843	12823	12796	1665.9	1942.8
	10	100	6438	6431	6436	6431	6476	6417	6443	6418	b	6407	6379	870.2	1094.7
		200	12603	12641	12638	12654	12605	12633	b	12632	12615	12648	12601	2768.7	3189.6
	20	100	6327	6293	b	6332	6308	6314	6273	6324	6308	6337	6269	1746.1	2126.1
		200	12532	12483	12535	b	12552	12567	12516	12567	12510	12567	12452	4878.4	5565.1

o = optimal solution value.
b = best overall solution value.

A continuación, podemos observar las pruebas realizadas con GLPK. Es evidente que, a medida que la cantidad de agentes y tareas aumentan, los tiempos lo hacen también para este software; si bien puede obtener óptimos globales en ciertos datasets a bajos costos. Los valores para los parámetros están indicados en las instancias.
Por ejemplo, 515-x significa que se utilizaron 5 agentes para cubrir 15 tareas, instancia x

Caso	instancia	tiempo (seg)	nºejec	valor hallado	óptimo	error rel. (%)
gap max 1	515-1	0.1	20	336	336	0
gap max 1	515-3	0.1	20	339	339	0
gap max 1	515-5	0.1	20	326	326	0
gap max 2	520-1	0.2	20	434	434	0
gap max 2	520-3	0.0	20	420	420	0
gap max 2	520-5	0.1	20	428	428	0
gap max 3	525-1	0.3	20	580	580	0
gap max 3	525-3	0.3	20	573	573	0
gap max 3	525-5	0.2	20	564	564	0
gap max 5	824-1	0.5	20	563	563	0
gap max 5	824-3	0.3	20	564	564	0
gap max 5	824-5	2.7	20	559	559	0
gap max 7	840-1	56.7	20	942	942	0
gap max 7	840-3	2.2	20	968	968	0
gap max 7	840-5	7.3	20	951	951	0
gap max 8	848-1	316.5	20	1133	1133	0
gap max 8	848-3	43.5	20	1141	1141	0
gap max 8	848-5	420	20	1122	1127	1
gap max 9	1030-1	4.3	20	709	709	0
gap max 9	1030-3	5.7	20	712	712	0
gap max 9	1030-5	51.5	20	706	706	0
gap max 10	1040-1	3.6	20	958	958	0
gap max 10	1040-3	34	20	960	960	0
gap max 10	1040-5	420	20	944	947	1
gap max 11	1050-1	75.5	20	1139	1139	0
gap max 11	1050-3	0.6	20	1195	1195	0
gap max 11	1050-5	18.5	20	1171	1171	0
gap max 12	1060-1	7.5	20	1451	1451	0
gap max 12	1060-3	19.2	20	1433	1433	0
gap max 12	1060-5	300	20	1446	1446	0

Tabla comparativa con ejecuciones de GRASP

instancia	nº ejec	alfa	tiempo	optimal	value found	error relat (%)
gap7-840-1	20	1	15.6997	942	922	3
gap9-1030-5	20	1	19.2572	706	689	3
gap10-1040-5	20	0.92	55.4369	947	937	3
gap11-1050-5	20	0.9	64.8346	1171	1147	3
gap11-1050-1	20	1	28.962	1139	1113	3
gap12-1060-5	20	1	388.3736	1446	1435	1
gap-A-20200	20	0	401.3421	2339	2353	1

6 CONCLUSIONES

Las tecnicas metaheurísticas basadas en dos fases, construcción y búsqueda local, representan una alternativa eficiente para problemas de decisión bien conocidos por pertenecer a la clase NP-hard, de modo de encontrar, a bajo costo, respuestas factibles y cercanas al óptimo global.
No obstante, es posible realizar importantes mejoras al algoritmo, en cuanto al estancamiento que puede sufrir en un determinado óptimo local, de modo de provocar una perturbación en la mejor solución actual y poder guiar el método de búsqueda por otras regiones (diversificación), para posteriormente efectuar intensificación (explorar a fondo tal región). Nuevamente, Path Relinking puede aportar resultados significativos en estos procesos.

REFERENCES

[1] Lourenco, H.R. *Heurísticas adaptativas para el problema de asignación generalizada (Adaptive heuristics for the generalised assignment problem). In Proceedings of The First Spanish Congress in Evolutive and Bioinspired Algorithms*, Merida, Spain, February 6-7, pp. 267-275. ISBN 84-607-3913-9.

[2] I.H. Osman. *Heuristics for the Generalised Assignment Problem: Simulated Annealing and Tabu Search Approaches*, OR Spektrum, Vol.17, 211-225, (1995).
D. Cattrysse, M. Salomon and L.N. Van Wassenhove. *A set partitioning heuristic for the generalised assignment problem*, European Journal of Operational Research, Vol.72, 167-174, (1994)

[3] P.C. Chu and J.E. Beasley,*A genetic algorithm for the generalised assignment problem*, Computers Operations Research, 24 (1997) 17-23.

M. Yagiura, T. Ibaraki and F. Glover, *A Path Relinking Approach with Ejection Chains for the Generalized Assignment Problem*, European Journal of Operational Research, 169 (2006) 548-569.

[4] T.A. Feo and M.G.C. Resende, *Greedy randomized adaptive search procedures. J. of Global Optimization*, 6: 109-133, 1995.
M. G. C. Resende and C. C. Ribeiro, *Greedy randomized adaptive search procedures.* In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pp. 219249, Kluwer Academic Publishers, (2003).