Solución de una versión del problema de la mochila tridimensional con elección múltiple para la aplicación Servicios ETECSA.

Héctor Adrián Castellanos C411, Christian Rodríguez C411.

h.castellanos@estudiantes.matcom.uh.cu, c.rodriguez@estudiantes.matcom.uh.cu

Resumen:

En este informe daremos solución al problema de dados un conjunto de planes que oferta la compañía telefónica ETECSA, donde cada plan nos proporciona minutos de voz, una cantidad mensajes de texto y otra cantidad de megabits de datos, proporcionar al usuario la información de cuales planes y que cantidad de cada uno debería comprar para satisfacer sus necesidades de consumo. Dicho problema lo modelaremos como uno de optimización lineal en enteros y le daremos la solución óptima usando programación lineal. También mostraremos que el problema más específicamente es una variante del problema de la mochila y propondremos una solución con programación dinámica que nos da igualmente el óptimo pero que no es extensible debido a su complejidad computacional. Por ultimo expondremos un algoritmo genético que desarrollamos apoyándonos en resultados previos de otros autores y compararemos los resultados de esta solución con la obtenida con programación lineal.

Abstract:

In this report we will solve the problem of given a set of plans offered by the telephone company ETECSA, where each plan provides voice minutes, a number of text messages and a number of megabytes of data, providing the user whit the information of which plans and what amount of each one should by to meet their consumption needs. We will model this problem as one of linear optimization in integers and we will give you the optimal solution using linear programing. We will also show that the problem more specifically is a variant of the knapsack problem and we will propose a solution with dynamic programming that also gives us the optimum but that is not extensible due to its computational complexity. Finally we expose a genetic algorithm that we develop based on previous results of other authors and we will compare the results of this solutions with the one obtained with linear programing.

Palabras claves: mochila multidimensional, algoritmo genético, optimización combinatoria.

Introducción:

El problema que queremos resolver es el siguiente: dados unos consumos **V** de voz, **S** de sms y **D** de datos de un usuario y el listado de los planes que oferta ETECSA, decir para cada plan cuantos de ese tipo debería comprar el usuario, de forma que se satisfagan los consumos del usuario y que se minimice su precio total. Más formalmente:

$$\min \sum_{j=1}^{n} p_j * x_j \quad [1]$$

$$\text{s.a} \sum_{j=1}^{n} r_{ij} * x_j \geq b_i \quad [2] \qquad i = 1, \dots, m$$

$$x_j \in Z, j = 1, \dots, n$$

Donde m es la cantidad de restricciones en nuestro caso m = 3 (restricciones para voz, sms y datos respectivamente); n son la cantidad de tipos de planes que oferta Etecsa, $p_j$es el precio del plan j, r_{ij} es lo que aporta el plan j del recurso i (voz, sms, datos).

Ya modelado el problema como uno de optimización lineal en enteros, usando la librería Pulp de python, con la siguiente función:

```python
def get_solution_lo(plan_list, limit_voice, limit_sms, limit_data):
    n = len(plan_list)
    opt_model = pulp.LpProblem('Model', pulp.LpMinimize)

    x = pulp.LpVariable.dicts('X', [i for i in range(n)], lowBound = 0, upBound = None,
                              cat = pulp.LpInteger)

    opt_model += sum([ plan_list[i].price * x[i] for i in range(n)])
    opt_model += sum([plan_list[i].voice * x[i] for i in range(n)]) >= limit_voice
    opt_model += sum([plan_list[i].sms * x[i] for i in range(n)]) >= limit_sms
    opt_model += sum([plan_list[i].data * x[i] for i in range(n)]) >= limit_data

    opt_model.solve()

    result = []

    for i in range(n):
        result.append(int(x[i].value()))
    return result
```

Queda resuelto nuestro problema, y es verdad que es una excelente solución pero ocurren dos cosas. Primero que si aumentamos la cantidad de variables (tipos de planes) considerablemente, al ser esta una solución que en realidad lo que hace es ramificación y poda, puede tornarse muy lenta. Segundo que queremos explotar más este problema desde otras perspectivas, en particular tratar este problema como una variante del problema de la mochila tridimensional con elección múltiple, por lo que de aquí en adelante el informe trataremos esa temática.

Del problema específico que nos ocupa no encontramos prácticamente nada en la literatura pues el problema estándar de la mochila consiste en maximizar una ganancia sin exceder la capacidad de la mochila y cada elemento podemos elegir cogerlo o no ;y en nuestro caso la capacidad de la mochila tiene tres dimensiones (voz, sms, datos) y lo que queremos hacer es irnos por encima de la capacidad de la mochila tanto en voz, como en sms y datos, minimizando el costo ,y cada elemento podemos cogerlo cualquier cantidad de veces. Más adelante mostremos como nosotros transformamos el problema de elegir cualquier cantidad

de veces un objeto (plan) al problema de elegir o no un objeto. También veremos cómo transformar el problema de minimizar pasándonos a el de maximizar sin pasarnos. Luego de estas transformaciones tenemos el problema de la mochila multidimensional 0-1 del que si hay literatura.

Los algoritmos exactos para la solución de dicho problema son mayormente de programación lineal en enteros, por lo general usando el algoritmo de ramificación y poda y el algoritmo de Balas. Mejoras y adaptaciones a estos para este problema en específico Shih(1979), Gavish y Pirkul(1985), Cramas y Mazzola(1994), dan la solución exacta pero con elevada complejidad computacional.

Los algoritmos heurísticos con los que se han intentado aproximar la solución de este problema están basados generalmente en búsqueda tabú y algoritmos genéticos, estos últimos con muy buenos resultados y es precisamente en el trabajo de Chu y Beaseley (1998) uno de los más relevantes. Más recientemente Balev(2008) propuso una heurística que usa programación dinámica de forma adecuada para obtener una solución factible por medio de sucesivas mejoras de la solución de redondeo de programación lineal, obteniendo muy buenos resultados respecto a la literatura actual. Respecto al tamaño de los problemas resueltos, Vasquez y Vimont (2005) obtuvieron las soluciones con mejor calidad respecto a los puntos de referencia de la literatura pero con tiempos computacionales considerablemente altos. Chu y Beaseley(1998) que ya los habíamos mencionado antes, proporcionaron la cobertura más completa de un algoritmo genético para la solución de este problema en cuya propuesta hacen uso por primera vez con éxito de una función de restricción o reparación para mantener siempre en el espacio de búsqueda soluciones factibles en lugar de usar por ejemplo funciones de penalidad. Hembecker (2007) usa optimización por enjambres de partículas para resolver este problema.

Desarrollo.

Primera aproximación a la solución del problema con programación dinámica.

El problema estándar de la mochila consiste en, dada un mochila con capacidad W y un conjunto de objetos que tienen un peso wi y un precio pi escoger un subconjunto de estos de forma tal que la suma de los wi de los objetos escogidos no sobrepase W y que la suma de los pi sea la máxima posible, siempre cumpliendo la primera condición. Vamos ahora a transformar nuestro problema a este problema en la medida de lo posible con el objetivo de simplificarlo y de llevarlo a un estado más fácil de trabajar y del que hay más bibliografía y resultados.

Primera transformación de nuestro problema, de mochila con múltiple elección a mochila 0-1:

En nuestro problema cada objeto puede ser seleccionado cualquier cantidad de veces. Lo que haremos es que para todo objeto (plan) j agregaremos un conjunto de objetos Sj de modo que para toda elección de X objetos del tipo j exista un subconjunto de Sj que represente esa elección donde cada objeto de Sj puede ser cogido o no ,y para nuestro problema X es finito. Ilustremos esto.

Por ejemplo sea el plan P tal que P.voz = 40, P.sms = 30, P.datos = 100 digamos que no tiene sentido en nuestro problema tomar un mismo plan más de 200 veces o sea X $\leq$ 200 luego en nuestro nuevo problema nosotros agregaremos el conjunto de planes Sp donde:

$$S_p = \{p | p.voz = P.voz * 2^i, p.sms = P.sms * 2^i, p.data = P.data * 2^i\} \nabla i | 2^i \leq maxX$$

$$, i \geq 0$$

O sea:

p0.voz = 40, p0.sms = 30, p0.datos = 100.

p1.voz = 80, p1.sms = 60, p1.datos = 200.

p2.voz = 160, p2.sms = 120, p2.datos = 400.

Y así sucesivamente, por ejemplo coger el plan P cinco veces es equivalente a seleccionar el plan p0 y el plan p2, se cumple la equivalencia debido a que todo número puede ser expresado en potencias de 2 distintas.

La segunda transformación, de maximizar sin pasarnos a minimizar pasándonos:

Primeramente sabemos que la dinámica estándar de la mochila nos calcula en dp[i] el máximo beneficio con exactamente capacidad i por lo que si pudiéramos hacer esta misma dinámica hacia el infinito lo que minimizando nuestra respuesta del mínimo costo y pasándonos ,sería el min(dp[i]) para todo i mayor o igual que la capacidad de la mochila pero en realidad no tenemos que calcular la dp en el infinito sino solo hasta W + w*, donde W es la capacidad de la mochila y w* es el peso del objeto que más pesa. Esta filosofía es fácilmente extensible para más dimensiones. En nuestro problema la capacidad de la mochila W es en realidad la cantidad de minutos voz que queremos obtener de los planes y esto sería en una dimensión si además tenemos en cuenta la cantidad de megabits en datos y de mensajes entonces tendríamos una mochila en tres dimensiones.

Dicha solución con programación dinámica es exacta pero su complejidad es O(n*V*S*D) donde n es la cantidad de planes V, S y D la voz, los sms y los datos que queremos satisfacer. Por lo que es exponencial, en función de V, S y D. Las transformaciones mencionadas nos van a ser útil luego.

Algoritmo genético.

De ahora en adelante no vamos a perseguir la solución óptima del problema pues sabemos que el problema de la mochila es NP-completo y como nuestro problema se reduce a este, también lo es. Por lo que nos centraremos en obtener un resultado que aproxime lo mejor posible al óptimo con un algoritmo genético, pues estos han demostrado tener gran efectividad en cuanto a problemas de optimización combinatoria se refieren y más particularmente a problemas de la mochila.

Los algoritmos genéticos fueron desarrollados por John Holland (1975) y sus colegas de la Universidad de Michigan en la década de 1970 como una técnica de búsqueda estocástica basada en el mecanismo de selección natural y recombinación.

El término estocástico se refiere a los procesos, algoritmos o modelos en los que existe una secuencia cambiante de eventos a medida que transcurre el tiempo, son incididos por el azar y el no determinismo.

En la resolución de problemas de optimización los algoritmos genéticos imitan el proceso de evolución de una población de individuos (soluciones) en la naturaleza, a cada individuo de la población también se le llama cromosoma y se define una función que determina que tan bueno es un cromosoma *fitness function* .En cada iteración del algoritmo se seleccionan padres (conjunto de cromosomas) a partir de los cuales se generan hijos (nuevos integrantes de la población) mediante operaciones llamadas entrecruzamiento y mutación similar a como ocurre en la naturaleza. Luego se obtiene una población evolucionada en la que pudiéramos descartar los individuos que se van quedando atrás respecto a la *fitness function.* En resumen los pasos básicos de un algoritmo genético son:

Generar una población inicial.

Evaluar la *fitness function* en cada individuo de la población.

Repetir:

> Seleccionar padres.
>
> Reproducir hijos mediante mutaciones y entrecruzamiento.
>
> Evaluar la *fitness function* en los hijos.
>
> Reemplazar algunos o todos los elementos de la población.

Hasta que se encuentre una solución satisfactoria.

Como podemos ver los algoritmos genéticos tienen una estructura bastante bien definida, y varían de uno a otro en el hecho de como representamos las soluciones, como elegimos la población inicial, como definimos el entrecruzamiento y las mutaciones, como medimos que tan bueno es un cromosoma, como seleccionamos los cromosomas a partir de los cuales generaremos descendencia y que tan grande puede ser la población o durante cuánto tiempo la haremos evolucionar, entre otros aspectos. Inmediatamente explicamos cómo se manifiestan cada uno de estos aspectos en nuestro algoritmo.

Representación de las soluciones.

Las soluciones (cromosomas de la población) las representamos como un array binario S de tamaño n donde n es la cantidad de objetos y S[i] = {0, 1}, $1 \leq i \leq n$, si S[i] es 1 significa que en la solución S el objeto i es cogido 0 en otro caso.

Función de aptitud (*fitness function*).

F(S) = $\sum_{j=1}^{n} p_j * S[j]$

Mientras menor sea F(S) mejor es la solución S.

Selección de los padres.

Para la selección de los padres lo que hicimos fue tomar aleatoriamente 2 pares de individuos de la población y en cada par nos quedamos con los que menor evaluación en *fitness function* presenten (pues se quiere minimizar el precio total), esta forma de selección se conoce como torneo binario, binary tournament en inglés, notar que nos quedamos con solo 2 padres para la reproducción.

Entrecruzamiento y mutación.

El entrecruzamiento que utilizamos es el entrecruzamiento uniforme que dados los padres S1 y S2 y el hijo que estos generan S3 el gen i de S3 tiene igual probabilidad de ser tanto de S1 como de S2 o sea S3[i] = S1[i] con probabilidad 0.5 y S3[i] = S1[i] con probabilidad 0.5.

Para mutar un cromosoma S elegimos al azar un gen i de este y lo cambiamos, más formalmente S[i] = 1 – S[i].

Con gen nos referimos al valor que hay en una posición del cromosoma.

Función de reparación.

En lugar de utilizar una función de penalidad lo que hacemos es siempre mantener soluciones factibles en la población (recordemos que una función es factible en nuestro problema si se cumple [2]), como logramos eso, pues sea S una solución cualquiera (factible o no) lo que hacemos es recorrer todos los objetos en algún orden (en nuestro caso utilizamos un recorrido aleatorio porque nos dio mejores resultados) y si hay alguna restricción que no se cumple añadimos este objeto a S (se supone que cogiendo a lo sumo todos los objetos logramos la factibilidad), luego recorremos los objetos nuevamente pero esta vez en orden inverso y para cada objeto si este se puede quitar de la solución y esta continua siendo factible entonces lo eliminamos de S, de esta forma minimizamos el precio total y mantenemos la factibilidad.

Las funciones de penalidad se utilizan para penalizar soluciones no factibles o para guiar la población en alguna dirección.

Población inicial.

La población inicial la obtenemos de manera aleatoria con todos los individuos reparados.

El tamaño de la población es de cincuenta individuos.

Otros aspectos.

Hacemos diez mil iteraciones del algoritmo. En cada iteración seleccionamos dos padres con el método descrito anteriormente, los entrecruzamos y el hijo que obtenemos lo mutamos. Luego de cada ciclo evolutivo actualizamos el óptimo global y descartamos el individuo menos apto.

Antes de llegar al algoritmo genético descrito probamos varias ideas, como usar función de penalidad, aplicar también entrecruzamiento aleatorio, crear una población inicial basándonos en el algoritmo de programación dinámica propuesto, cambios en los tamaños de la población y numero de iteraciones del algoritmo, entre otros aspectos.

Resultados.

Los resultados obtenidos fueron muy favorables, muchas veces coincidiendo con el óptimo y otras con el error relativo bastante cercano a cero. Logramos un programa capaz de resolver el problema planteado y extensible a cualquier cantidad o tipos de planes, que va de la mano de una aplicación para móvil para que el usuario tome la mejor opción.

Ahora mostramos algunos resultados obtenidos con los planes actuales de ETECSA y luego con los planes de ETECSA más algunos planes propuestos por nosotros y algunos consumos hipotéticos. A la izquierda el resultado usando Pulp y a la derecha el proporcionado por el algoritmo genético.

Planes de ETECSA.

PRECIO, VOZ, SMS, DATOS.

Plan0: 7.0 $, 0 min, 0 sms, 600 mb.

Plan1: 10.0 $, 0 min, 0 sms, 1024 mb.

Plan2: 20.0 $, 0 min, 0 sms, 2560 mb.

Plan3: 30.0 $, 0 min, 0 sms, 4096 mb.

Plan4: 1.5 $, 5 min, 0 sms, 0 mb.

Plan5: 2.9 $, 10 min, 0 sms, 0 mb.

Plan6: 4.2 $, 15 min, 0 sms, 0 mb.

Plan7: 6.5 $, 25 min, 0 sms, 0 mb.

Plan8: 10.0 $, 40 min, 0 sms, 0 mb.

Plan9: 0.7 $, 0 min, 10 sms, 0 mb.

Plan10: 1.3 $, 0 min, 20 sms, 0 mb.

Plan11: 2.1 $, 0 min, 35 sms, 0 mb.

Plan 12: 2.5 $, 0 min, 45 sms, 0 mb.

Los sets se darán con el siguiente formato(consumo de VOZ, consumo de SMS, consumo de DATOS)	***fitness function* evaluada en la solución con programación lineal usando Pulp(optimo).**	***fitness function* evaluada en la solución encontrada por el algoritmo genético.**	**Error relativo de nuestra solución.**
(1560, 2120, 4500)	545.2	545.2	0.000000
(100, 100, 2000)	51.4	51.5	0.001945
(500, 500, 5000)	193.9	194.2	0.001547
(50, 60, 600)	23.7	23.7	0.000000
(200, 400, 8000)	132.5	132.5	0.000000
(135, 20, 750)	45.5	45.5	0.000000
(380, 195, 3750)	137.0	137.1	0.000729
(731, 20, 550)	192.5	192.5	0.000000

Planes de ETECSA más planes hipotéticos creados por nosotros.

Plan13: 6.0 $, 20 min, 30 sms, 100 mb.

Plan14: 10.0 $, 30 min, 0 sms, 150 mb.

Plan15: 3.0 $, 0 min, 10 sms, 100 mb.

Plan16: 8.0 $, 15 min, 0 sms, 800 mb.

Plan17: 20 $, 30 min, 40 sms, 3000 mb.

Plan18: 2.9 $, 50 min, 80 sms, 0 mb.

Los sets se darán con el siguiente formato(consumo de VOZ, consumo de SMS, consumo de DATOS)	***fitness function* evaluada en la solución con programación lineal usando Pulp(optimo), costo.**	***fitness function* evaluada en la solución encontrada por el algoritmo genético, costo.**	**Error relativo de nuestra solución.**
(1560, 2120, 4500)	448.9	454.9	0.013366
(100, 100, 2000)	39.7	39.7	0.000000
(500, 500, 5000)	156.5	158.0	0.009584
(50, 60, 600)	20.0	20.0	0.000000
(200, 400, 8000)	97.9	97.9	0.000000
(135, 20, 750)	39.3	39.3	0.000000
(380, 195, 3750)	115.4	116.0	0.005199
(731, 20, 550)	189.3	189.3	0.000000

Experimentos.

Los resultados de los experimentos los obtuvimos en un laptop Sony VAIO core i3 de segunda generación con 8 gb de RAM. Usamos los planes actuales de ETECSA y consumos hipotéticos, también hicimos los experimentos con planes creados que combinaran voz, texto y datos.

Conclusiones.

En general obtuvimos buenos resultados que se acercan mucho a los dados por Pulp. Nos quedan como trabajos futuros mejora la elección de la población inicial, incluir un índice para la mutación (mutar menos al inicio y más al final) y quizás en lugar de hacer la implementación en python hacerla en C por ejemplo para disminuir el tiempo de ejecución y poder aumentar el tamaño de la población y el número de iteraciones del algoritmo genético, para mejorar los resultados.

Referencias.

An Improved Genetic Algorithm for Knapsack Problems, Taskiran, Gamze Kilincli

A Genetic Algorithm for the Multidimensional Knapsack Problem, P.C. CHU AND J.E. BEASLEY