Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI-5651 - Diseño de Algoritmos I
Trimestre Enero-Marzo 2017

Proyecto II: Resolviendo el PRPP con Branch and Bound (versión 1.3)

El objetivo de este proyecto es el resolver de manera exacta el PRPP por medio de un algoritmo que usa la técnica de Ramificación y Acotamiento. El enunciado del proyecto está organizado como sigue: En la primera sección se describe un algoritmo basado en Ramificación y Acotamiento para el PRPP, en segunda sección se explican las actividades que debe llevar a cabo y en la tercera sección se dan las condiciones de entrega.

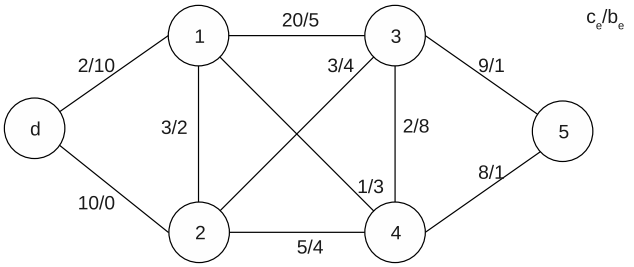
1. Un algoritmo exacto para el PRPP

En esta sección se presenta un algoritmo exacto para resolver el PRPP, que usa un esquema de ramificación y acotamiento. Resolver de manera óptima, problemas de optimización combinatoria que son *NP-hard* puede requerir de un gran esfuerzo computacional. Los algoritmos de ramificación y acotamiento son una de las herramientas más exitosas para el diseño de algoritmos exactos para resolver problemas *NP-hard*. Un algoritmo de ramificación y acotamiento recorre todo el espacio de soluciones del problema en busca de la solución óptima. Pero la enumeración explícita de todas las soluciones factibles, es muchas veces imposible debido al número exponencial de potenciales soluciones. Como lo indican Papadimitriou y Steiglitz [3] el método de ramificación y acotamiento es basado en la idea de hacer una *enumeración inteligente* de todos los posibles puntos de un problema de optimización combinatoria. El uso de cotas para la función a optimizar combinado con el valor de la mejor solución actual, permite al algoritmo buscar en partes del espacio de soluciones sólo de manera implícita.

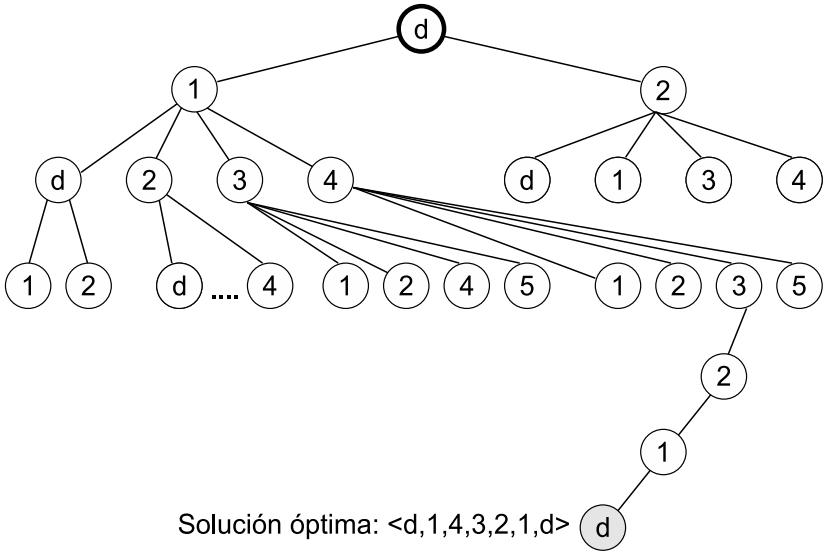
Una solución de un problema con un algoritmo de ramificación y acotamiento puede ser descrita como una búsqueda en un árbol de búsquedas, donde sus nodos representan soluciones parciales o totales del problema. En el caso del PRPP la raíz del árbol corresponde a una solución inicial factible, la cual puede ser una que sólo contenga al depósito. Dado cualquier nodo del árbol, los hijos de ese nodo son los vértices alcanzables desde ese nodo. A medida en que se desciende en el árbol y se visitan nodos, se van formando los caminos. En la Figura 1b se muestra el árbol de búsqueda con que se modela el PRPP. Este árbol corresponde al grafo de la Figura 1a. Se puede observar que el número de hojas que se obtiene en el árbol es exponencial. El árbol de búsqueda es generado dinámicamente durante la búsqueda e inicialmente solo contiene el nodo raíz. La descripción del algoritmo de ramificación y acotamiento para resolver el PRPP se muestra en el Algoritmo 1. Las funciones que componen el Algoritmo 1 se presentan en el Algoritmo 2. En las siguientes secciones se explican los principales componentes del algoritmo exacto.

1.1. Condiciones para la ramificación

Además de la condición de acotamiento, hay otras restricciones adicionales que se toman en cuenta para la expansión de un nodo en el árbol de búsqueda. Dado un nodo en el



(a) Ejemplo de una instancia del PRPP.



(b) Árbol de búsqueda.

Figura 1: Instancia del PRPP y el árbol de búsqueda del algoritmo de ramificación y acotamiento, de la Figura 1a. El nodo gris d es una hoja que representa la solución óptima del PRPP

árbol, que representa un camino C de d a un vértice v, los nodos sucesores de ese nodo corresponderán a caminos a expandir desde C, es decir, de la forma $C||<v,e,w>$. Para expandir un lado e debe cumplirse en primer lugar que ese lado no se encuentre más de dos veces en la solución parcial, esto es por Dominancia 1. En segundo término al agregar el lado a la solución parcial no debe formarse un ciclo de beneficio negativo. Esto se debe a que una solución óptima no posee ciclos de beneficio negativo. En la Figura 2 tenemos un ejemplo de una solución parcial a la cual se le agrega un lado, con el cual se forma una solución que posee un ciclo negativo, en este caso el ciclo es $< 3-8-3 >$. El algoritmo evita agregar el lado $(3,d)$. De manera formal.

Proposición 1. *Un ciclo $\mathscr{C}^*$ que sea solución óptima del PRPP no posee sub ciclos negativos*

Demostración. Por contradicción. Suponemos una solución óptima $\mathscr{C}$ cualquiera que tenga un sub ciclo con beneficio negativo. Al eliminar ese sub ciclo de $\mathscr{C}$ tenemos un nuevo ciclo que es solución del PRPP y tiene mayor beneficio que $\mathscr{C}$. Esto es una contradicción, por lo tanto $\mathscr{C}$ no es solución óptima del PRPP. □

Por último se evita expandir nodos que formen un ciclo que ya haya sido formado anteriormente. Es decir, se evita expandir lados que formen ciclos repetidos. Por ejemplo dada la Figura 3 supongamos que el ciclo a se generó durante la búsqueda del algoritmo.

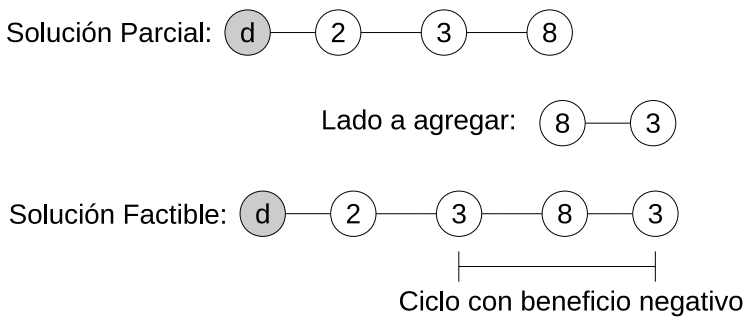


Figura 2: Ejemplo de una solución factible con ciclo negativo

Cuando se va a expandir la solución parcial $< ... - 2 - 3 - 6 - 5 - 4 >$ con el lado $(4, 3)$ para formar el ciclo b, se obtiene un ciclo equivalente al ciclo a, por lo que se evita expandir el lado $(4, 3)$. Esta condición es posible llevarla a cabo, ya que el ciclo a se forma al agregar el lado $(6, 3)$, mientras que el ciclo b al agregar el lado $(4, 3)$, teniéndose que ambos lados tienen el mismo vértice incidente, en este caso el vértice 3. Debido a que los sucesores están ordenados de manera descendente de acuerdo al beneficio de los lados, se evita expandir un lado e que tenga un beneficio menor a un lado e' que se encuentre en la solución parcial.

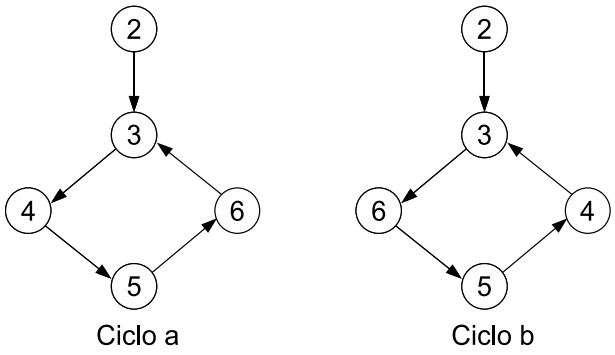


Figura 3: Ciclos equivalentes

1.2. Estrategia para seleccionar el próximo nodo

La estrategia de selección del próximo nodo a expandir tiene como fin encontrar una estrategia que permita explorar el menor número de nodos en el árbol, tal que no se comprometa la capacidad de la memoria del computador. Los lados e que son sucesores a expandir, se almacenan en una lista la cual se ordena de manera descendente según el beneficio que aportan. Si un lado e no se encuentra en la solución parcial su beneficio es el valor $b_e - c_e$, en caso contrario es $-c_e$. Se escoge para expandir el nodo abierto del árbol con mayor profundidad usando la estrategia de búsqueda en profundidad, implementada de manera recursiva. La idea es tratar de explorar siempre el nodo que represente la solución parcial con la más alta cota o beneficio, con el fin de guiar la búsqueda para tratar de encontrar la mejor solución factible lo más rápido posible.

1.3. Función de Acotamiento

La función de acotamiento es uno de los componentes claves del algoritmo de ramificación y acotamiento. Para este algoritmo inicialmente se calcula una solución factible por

medio de una de las heurísticas que se presentan en este trabajo. Esa solución va a corresponder a la mejor solución actual y su valor será el mejor valor conocido, el cual se usará en estrategia de acotamiento. La idea es tener una solución inicial que sea lo más cercana posible al óptimo. Se usa como cota inferior el beneficio de la mejor solución encontrada hasta el momento. Sea una solución parcial representada por el camino hasta un nodo en el árbol de búsqueda. Sea bsp el beneficio de una solución parcial. Sea br el beneficio total de los lados que no están en la solución parcial. El beneficio de la mejor solución encontrada lo denotamos como bms. Si el beneficio de la solución parcial más el beneficio restante de los lados del grafo que no están en la solución parcial es menor que el beneficio de la mejor solución conocida, entonces se descarta esa solución parcial, es decir se poda el subárbol cuya raíz es este nodo. Esto es, si se cumple para un nodo que $bsp + br \leq bms$ entonces se elimina esa rama del árbol de búsqueda.

Algoritmo 1: Algoritmo de Ramificación y Acotamiento

```
   Input: Un grafo G = (V, E) y un ciclo solInicial solución de PRPP
   Output: Un ciclo C, solución óptima de PRPP
 1 begin
 2     solParcial ←< d > // Variable Global
 3     mejorSol ←solInicial   // Variable Global
 4     beneficioDisponible ← obtener-max-beneficio(G)   // Variable Global
 5     busqueda-en-Profundidad()

 6 busqueda-en-Profundidad()
 7 begin
 8     v ← vertice-mas-externo(solParcial)
 9     // Vemos si se encuentra una mejor solución factible
10     if (v = d) then
11         if (beneficio(solParcial) > beneficio(mejorSol)) then
12             mejorSol ← solParcial
13     L(v) ← obtener-lista-de-sucesores(v) // Lista de aristas
14     foreach e ∈ L(v) do
15         if (¬ciclo-negativo(e, solParcial) ∧
           ¬esta-lado-en-sol-parcial(e, solParcial) ∧
           ¬repite-ciclo(L(v), e, solParcial) ∧
           cumple-acotamiento(e, solParcial)) then
16             agregar-lado(e, solParcial)
17             beneficioDisponible = beneficioDisponible − máx(0, b_e − c_e)
18             busqueda-en-Profundidad()
19     e = eliminar-ultimo-lado(solParcial)
20     beneficioDisponible = beneficioDisponible + máx(0, b_e − c_e)
```

2. Actividades a realizar

Debe implementar el algoritmo exacto para el PRPP, basado en la técnica Branch and Bound, que se describió en la Sección 1. Observe que en la línea 3 del Algoritmo 1, se indica que el algoritmo de Branch and Bound hace uso de una solución inicial. La solución inicial

Algoritmo 2: Funciones del Algoritmo de Ramificación y Acotamiento

```
1  func obtener-lista-de-sucesores(v: vértice)
2  begin
3      L(v) ← ∅
4      foreach arista e incidente a v do
5          L(v) ← agregar una arista e_1 con beneficio b_e y costo c_e
6          L(v) ← agregar una arista e_2 con beneficio 0 y costo c_e
7          Ordenar de manera descendente a L(v) según su beneficio
8      return L(v)

9  func ciclo-negativo(e: arista, solParcial: Secuencia de aristas)
10 begin
11     if al agregar la arista e a la solución parcial solParcial se forma un ciclo con
       beneficio negativo then
12         return Verdadero
13     return Falso

14 func esta-lado-en-sol-parcial(e: arista, solParcial: Secuencia de aristas)
15 begin
16     if la arista e no se encuentra en solParcial then
17         return Falso
18     else if la arista e se encuentra una vez en solParcial then
19         if b_e = 0 then // La arista se recorre por segunda vez
20             return Falso
21         else
22             return Verdadero
23     else
24         return Verdadero

25 func repite-ciclo(L(v): Lista de Aristas, e: arista, solParcial: Secuencia de
   aristas)
26 begin
27     if al agregar e en solParcial se forma un ciclo con una arista e' then
28         if (e' ∈ solParcial ∧ ((b_e − c_e) < (b_e' − c_e'))) then
29             return Falso
30         else
31             return Verdadero
32     return Falso

33 func cumple-acotamiento(e: arista, solParcial: Secuencia de aristas)
34 begin
35     beneficioE ← b_e − c_e
36     beneficioSolParcial ← beneficio(solParcial) + beneficioE
37     maxBeneficio ← beneficioDisponible − máx(0, beneficioE) + beneficioSolParcial
38     if (maxBeneficio ≤ beneficio(mejorSol)) then return Falso
39     return Verdadero
```

con que debe comenzar su programa, debe ser la solución dada por el algoritmo ávido para el PRPP realizado para el Proyecto 1. En caso que su algoritmo ávido no resultados que le parezcan satisfactorios, puede hacer uso del algoritmo greedy dado en la sección de apéndices, en lugar del algoritmo que implementó en el proyecto 1. En caso de usar el algoritmo del apéndice, debe indicarlo en el informe.

Las instancias que van a servir para evaluar su programa son las siguientes: desde la instancia P01 hasta la instancia P20, desde la instancia D0 hasta la D14, desde la instancia G0 hasta la instancia G15 y desde la instancia R0 y R10. Este tipo de programas exactos pueden tardar mucho tiempo en obtener la solución a una instancia del PRPP. Debido a esto se va a establecer un *tiempo mínimo de corte* para el cómputo de cada instancia, en caso de que su programa no se encuentre la solución antes de ese tiempo. Si usted desea puede dejar ejecutando su programa un tiempo mayor al tiempo de corte. Se establece como tiempo mínimo de corte para cada instancia, 2 horas de ejecución en un computador con un CPU Intel i7 980X. La idea es que dependiendo del poder de cómputo del CPU de su computador, usted va tener un tiempo de corte mayor o menor. Para poder determinar cual es el poder de cómputo de cada CPU debe hacer uso de los valores indicados en [4]. Se puede observar que en [4], que el procesador Intel i7 980X tiene una puntuación de 1452 en el procesamiento con un solo hilo.

Se quiere que los resultados de su programa se presenten en una tabla, con las que se comparan cada una de las instancias, y que que debe tener las siguientes columnas:

1. Nombre de la instancia
2. Valor óptimo
3. Valor de la heurística
4. % desviación de la heurística
5. Tiempo del algoritmo Branch and Bound

Si su programa alcanza el tiempo de corte sin obtener la solución óptima de la instancia, en la celda correspondiente al *Tiempo del algoritmo Branch and Bound*, debe colocar el símbolo -T-.

Como actividad opcional, puede hacer un segundo programa que contenga mejoras al algoritmo exacto de la Sección 1. En caso de implementar un segundo programa, el mismo debe hacer uso de la misma solución inicial que el algoritmo requerido. Los resultados del nuevo algoritmo propuesto se deben reportar en una columna adicional en la tabla de resultados, la cual debe llevar como nombre *Tiempo del algoritmo propuesto*. Se darán puntos extras a los que implementen la actividad adicional.

Debe hacer un breve informe con dos secciones:

1. **Descripción de la solución propuesta:** esta sección solo se hace en caso de haber hecho una modificación al algoirmto dado. Si ese es el caso, entonces haga una descripción y justificación de la solución propuesta y el algoritmo diseñado.
2. **Detalles de la implementación:** Detalles de la implementación de las estructuras y algoritmos de su programa.
3. **Resultados experimentales y análisis:** los datos del computador donde se realizaron los experimentos, las tablas con los resultados experimentales, y un análisis de los mismos.

3. Condiciones de entrega

La entrega de este proyecto la debe realizar por email antes del miércoles de la semana 12 hasta las 11:59 pm.

Guillermo Palma / gvpalma@usb.ve / Marzo de 2017

Referencias

[1] Baldoquín, M., Rodriguez, S.and Hibbard, T., and Castellini. Un enfoque híbrido basado en metaheurísticas para la solución del problema del cartero rural. In *XI CLAIO* (Concepción de Chile, Chile, 2002).

[2] Feo, T., and Resende, M. Greedy randomized adaptive search procedures. *Journal of Global Optimization 6*, 2 (1995), 109–133.

[3] Papadimitriou, C., and Steiglitz, K. *Combinatorial optimization: algorithms and complexity*. Dover Pubns, 1998.

[4] PassMark. Cpu benchmarks, single thread performance. `http://cpubenchmark.net/singleThread.html`, 2017.

A. Heurística constructiva ávida para el PRPP

Esta heurística esta inspirada en la algoritmo GRASP [2] presentado por Baldoquín et al. [1] para el RPP. El algoritmo agrupa los lados que dan beneficio, del tipo $R \cup Q$, en un conjunto que llamamos T. Se parte de un lado que posee el depósito y luego iterativamente se escoge probabilísticamente un lado del conjunto T que tenga como incidente el vértice que es adyacente al depósito y de esta manera se obtiene el inicio de la solución. En caso de no haber un vértice incidente en el conjunto T se calculan los caminos de costo mínimo desde los vértices que forman parte del conjunto T hasta el vértice que forma parte de la solución. Se escoge probabilísticamente uno de esos caminos y se concatena con el vértice actual de la solución. Estos pasos se repiten hasta que no queden más lados con beneficios en el conjunto T, lo que indica que ya son parte de la solución. La escogencia de los lados y caminos a formar parte de la solución se hace probabilísticamente, siendo la probabilidad de ser seleccionado proporcional a su beneficio. Para el cálculo de los caminos de costo mínimo se utiliza una función de costo que se muestra en la ecuación (1). La misma trata de favorecer la inclusión de lados que posean beneficios cuando estos no forman parte de la solución, por lo que no han sido recorridos.

$$c(e) = \begin{cases} c_e, & \text{si } e \in \mathscr{C} \\ -\varphi_e, & \text{si } e \in P \\ 0, & \text{de lo contrario.} \end{cases} \tag{1}$$

Por el carácter probabilístico del algoritmo ávido, con él se pueden obtener soluciones diferentes, que contienen todos los lados con beneficio. La función principal de la *Heurística constructiva ávida* es servir como generador de soluciones factibles.

Algoritmo 3: Heurística constructiva ávida (*greedy*)

Input: Un grafo $G = (V, E, c)$
Output: Un ciclo $\mathscr{C}$, solución factible de PRPP

```
1  begin
2      C ⟵ ∅ // Contiene la solución del algoritmo
3      T ⟵ R ∪ Q
4      if Si el depósito d no se encuentra en ningún lado de T then
5          Encontrar el lado e_d = (d, u) ∈ E con mayor valor φ_{e_d}
6      b ⟵ d
7      while T ≠ ∅ do
8          if ∃(e ∈ T ∧ u ∈ V) | e = e_{b,u} = (b, u) then
9              e_bu ⟵ obtenerLado(T, b)
10             T ⟵ T\{e_bu}
11             C ⟵ agregarLado(C, e_bu)
12             b ⟵ u
13         else
14             CCM ⟵ ∅ // Conj. de los caminos de costo mínimo
15             foreach e ∈ T do
16                 foreach i ∈ V del lado e do
17                     CM_ib ⟵ caminoCostoMínimo(i, b)
18                     CCM ⟵ CCM ∪ {CM_ib}
19             CM_ib ⟵ obtenerCamino(CCM) // se guarda el i ∈ V
20             C ⟵ unirCaminoAlCiclo(C, CM_ib)
21             Eliminar los lados e ∈ CM_ib del conjunto T
22             b ⟵ i
23     if el último i ∈ V de C no es d then
24         CM_id ⟵ caminoCostoMínimo(i, d)
25         C ⟵ unirCaminoAlCiclo(C, CM_id)
26
27 func obtenerLado (T: Conj. de Lados, b: Vértice) ⟶ Lado
28 begin
29     Se escoge probabilísticamente un lado e ∈ T ∧ e = (b, u) siendo la probabilidad
       de ser seleccionado proporcional al beneficio φ_e del lado e
30     return e
31 func obtenerCamino (CCM: Conj. de caminos) ⟶ Camino
32 begin
33     Se escoge probabilísticamente un camino C ∈ CCM siendo la probabilidad de
       ser seleccionado proporcional al beneficio del camino (∑_{e∈C} φ_e)
34     return C
```