0.1 Resolución

Para la resolución se cuenta inicialmente con una cantidad de vértices n, aristas m, particiones k, y los pesos entre los pares de vértices. Para determinar la k-partición de peso mínimo se empezara utilizando un solo conjunto, *conjunto_1*. Se realizaran todas las combinaciones que existen para esa cantidad de conjuntos, la cual es solamente una (todos los vértices juntos). Y nos guardaremos el peso de esta combinación en una variable, *suma_solucion*. Si esta suma es cero, dado que los pesos son mayores o iguales a cero, entonces es la mínima que se puede obtener. Esta combinación es una solución óptima.
Sino creamos un nuevo conjunto y procedemos a buscar si existe una combinación, con esta cantidad, cuyo peso sea menor a *suma_solucion*. Si existe, se actualiza esta variable y esta combinación reemplazara a la que se tenia como posible solución. En caso de no existir, se agrega un nuevo conjunto y se proceden a formar nuevas combinaciones. Esto se realizara hasta que exista alguna combinación donde la suma de todos los conjuntos sea cero. O cuando se terminaron de realizar las combinaciones para k conjuntos. Ya que entonces, habremos visto todas las combinaciones posibles para el máximo de conjuntos que disponemos. Obteniendo una de las posibles soluciones óptimas.
Para lograr las combinaciones con mas de un conjunto. Se ubicará al vértice 1 en el último conjunto que haya. Luego, se procede a ubicar al siguiente vértice en el primero, si es que no exceda a *suma_solucion*. Una vez ubicado continuamos con el siguiente vértice, siguiendo la misma lógica. Si no es posible ubicar a alguno en un conjunto, se prueba metiéndolo en el siguiente. En el caso de que un vértice t, 1 < t <= n, no pueda ser ubicado en ninguno de los conjuntos disponibles hasta el momento. Significa que la combinación que se tiene con los vértices 1 a t-1 no es útil para avanzar. Por lo que se procede a remover a t-1 al próximo conjunto posible (desde la posición que ocupa t-1 actualmente). Si se lo ubicó, procedemos a tratar de ubicar nuevamente a t (empezando desde el primer conjunto) y sino, volvemos a retroceder. Pudiendo caer en dos casos:
Se ubicaron a los n vértices, entonces encontré una combinación mejor que la que tenia representada en *solucion*, nos guardamos este nuevo peso y la combinación. Removemos el ultimo vértice para úbicarlo en el siguiente conjunto posible. Para asi continuar realizando nuevas combinaciones.
Caso contrario, retrocedí hasta llegar al vértice 1. Esto significa que termine de observar todas las combinaciones posibles para los conjuntos disponibles actualmente. Por lo que voy a agregar un nuevo conjunto, ubicar al vértice 1 en este ultimo, y proceder con las combinaciones.

procedure UBICAR_VERTICE
    conjunto_1 ← *agregar todos los vértices*
    suma_solucion ← *sumar el peso del conjunto_1*
    solucion ← *conjunto_1*
    Nuevo_conjunto ← *crear un nuevo conjunto*
    k_particion ← *agregar el Nuevo_conjunto*
    vertice_actual ← 1
    for i= 1,...,*total de particiones* **do**
        if (*suma_solucion* == 0) **then**
            Return solucion
        end if
        Nuevo_conjunto ← *crear un nuevo conjunto y agregar el vertice_actual*
        k_particion ← *agregar el Nuevo_conjunto*
        ubicar_siguientes_vertices $(....i, vertice\_actual_{++}, solucion, k\_particion....)$
        Sacar vertice_actual del Nuevo_conjunto
    end for
    Return solucion
end procedure

```
procedure UBICAR_SIGUIENTES_VERTICES(.., conjuntos_disponibles, vertice_actual, solucion, k_particion..)
    for i = 1...total_conjuntos do
        if (suma_solucion == 0) then
            return solucion
        end if
        if (agregar vertice_actual al conjunto_i no excede suma_solucion) then
            agrego el vertice_actual al conjunto_i
            if (agregue el último vértice) then
                suma_solucion ← suma de los pesos de cada conjunto en k_particion
                solucion ← k_particion
            else
                ubicar_siguientes_vertices (....conjuntos_disponibles, vertice_actual++, solucion, k_particion....)
            end if
            sacar vertice_actual del conjunto_i
        end if
    end for
end procedure
```

0.2 Resolución agregando poda:

En la resolución anterior se comienza con el valor de *suma_solucion* igual al peso obtenido de ubicar a todos los vértices en un mismo conjunto. Ya que esta es la cota máxima y luego se realizan distintas combinaciones para ir optimizándola. La poda se encargara de comenzar con otro valor. Lo que haremos es obtener el peso que resulta de empezar distribuyendo al vértice número 1 en el primer conjunto, al segundo en el siguiente y así hasta que lleguemos al conjunto k o ya no nos queden nodos. Si se llego al k y aún quedan vértices, volvemos al primer conjunto. Hasta que distribuyamos a todos. A diferencia de la resolución anterior para este caso es necesario que el vector que representa a la combinación tenga tamao k en vez de uno. De esta forma, en el mejor de los casos si la cantidad de vértices es menor que k (cantidad total de conjuntos) obtendremos una de las combinaciones óptimas (peso total igual a cero). Y si no, al no tener a todos los vértices juntos estaremos eliminando adyacencias y ubicando a posibles vértices adyacentes en distintos conjuntos. Por lo que *suma_solucion* tiene un valor menor que el planteado en la anterior resolución. Y evitamos realizar las combinaciones que antes hacíamos para llegar al mismo. Luego se continua a partir de la linea 8 de $UBICAR\_VERTICE$.

0.3 Complejidad

k_particion es un conjunto de conjuntos de vértices (en él se van realizando las combinaciones). El mismo se representa con un vector de vectores de enteros. Pero, para un rápido acceso a la suma total de los pesos de los conjuntos. Se tendrá una tupla, la primer componente es la suma y la segunda el vector. Para calcular la complejidad total vamos a analizar por partes nuestro algoritmo.
A) Complejidad de la función *ubicar_vertice*:
1) Calcular el valor inicial de *suma_solucion* (que va a ser el peso de tener a todas los vértices en un solo conjunto). Es decir, sumar m aristas. O(m)
2) Ubicar a todos los vértices en un único posición conjunto que va a ser *solucion*. Se realizan n *push_back* en *solucion*[0]. O(n).
3) Se crea un primer vector *conjunto* y se lo agrega atrás en la segunda componente de *k_particion* O(1).
4) Se proceden a realizar las combinaciones, para esto se ejecuta un for que va desde 1 hasta k. En él, se van a realizar todas las combinaciones para los i-esimos camiones, 1 <= i <= k. Y se van a ir agregando los nuevos conjuntos. En cada iteración se crea uno, se agrega al vértice 1 en el mismo y se ubica al conjunto atrás, en el vector de *k_particion*, costo total O(1)(por iteración). Luego, se procede a llamar a la función *ubicar_siguientes_vertices* (mas adelante se analiza la complejidad de esta función). Y por último se retira al vértice, pop_back.

B) Complejidad de *ubicar_siguientes_vertices*:
Se trata de una función recursiva, primero vamos a comenzar por analizar que se hace en cada llamada y luego cuantas se hacen en total.

En cada llamada se trata ubicar a un vértice, entre los conjuntos disponibles actualmente. Para esto se realiza un for que va desde 1 hasta el valor de *conjuntos_disponibles*.
1)En cada iteración, al comenzar se evalúa si el valor de *suma_solucion* es cero, O(1). Si no, se procede a verificar, y de ser posible, a agregar el *vertice_actual* al i-esimo conjunto, 1 <= i <= *conjuntos_disponibles*. Para esto, se le suma a la primer componente de *k_particion* el peso que se adiciona el agregar el *vertice_actual* a dicho conjunto. Todos los vértices van a estar distribuidos entre los actuales *conjuntos_disponibles*. Si estoy tratando de ubicar al vértice t, 1 < t <= n, entonces voy a tener t-1 vértices distribuidos. Pero por cada iteración solo realizo tantas sumas como vértices haya en el i-esimo conjunto. Es decir que al realizar las *conjuntos_disponibles* iteraciones estoy realizando la suma de aristas para t-1 vértices. En el caso de que estos t-1 vértices formen un subgrafo completo tengo (t-1)*(t-2)/2 aristas, que fueron sumadas.
2) Si no se pudo agregar el vértice, se procede a realizar una nueva iteración. En caso contrario agrego al *vertice_actual* atrás, en el conjunto correspondiente, push_back. Si agregue el vértice n, el último, se actualiza el valor de *solucion_suma* a la de la primer componente de *k_particion*, O(1). Pero, además se copia el vector de la segunda componente a *solucion*. Este vector consta de tantas posiciones como *conjuntos_disponibles* haya hasta entonces. Con n vértices distribuidos en total. Costo de la copia O(n).
En caso de no serlo se realiza una nueva llamada recursiva de la función pero ahora para ubicar al $vertice\_actual_{++}$.
Finalizada la llamada se retira al *vertice_actual* del i-esimo conjunto, pop_back, y se realiza otra iteración.
Como se detalló a lo sumo en cada llamada los costos son O((t-1)*(t-2)/2 + n), como t esta entre 1 <= t $<= n$ acotamos por n, obteniendo $\mathrm{O}(n^2 + n) = \mathrm{O}(n^2)$.
3) Por último calculamos cuantas veces realizamos esta tarea. Cada llamada recursiva va a ubicar a un único vértice. Cada vez que se agrega un nuevo conjunto. Se evalúa ubicar nuevamente a los n vértices. Y se vuelven a ejecutar los for's de las funciones recursivas pero, ahora hasta una iteración más que la llamada anterior. Como se comienza ubicando al primer vértice en el último conjunto creado, y al siguiente desde el primero de los disponibles. Cada vez que un vértice es ubicado en un nuevo conjunto posibilita a lo sumo k conjuntos para el siguiente. Pero lo mismo ocurre con el siguiente a este, así hasta tener los n vértices. En conclusión tenemos las siguientes combinaciones:

$$\sum_{V_n=1}^{k} \sum_{V_{n-1}=1}^{k} \ldots\ldots \sum_{V_1=1}^{k} 1 = k^n$$

Donde V_s, 1<= s <= n, representa al vértice número s.
Y cada una asociada a los costos mencionados en los ítem 1-2 de la sección B.
El costo total, de ambas secciones seria: $\mathrm{O}(m)$ + $\mathrm{O}(n^2)$*$\mathrm{O}(k^n)$ = $\mathrm{O}(m)$ + $\mathrm{O}(n^2 * k^n)$
acotando m por $n * (n-1)/2$, que es el número máximo de aristas que puede haber para n vértices, tenemos: $\mathrm{O}(n^2)$ + $\mathrm{O}(n^2 * k^n)$ = $\mathrm{O}(n^2 * k^n)$

0.4 Complejidad con la poda:

Como se explicó en la parte de *resolucion con poda*, con la misma se cambia la forma de obtener el valor inicial de *peso_solucion* y a la k-partición que la respresenta. Mientras que el resto del algoritmo se mantiene, por lo que el análisis anterior solo cambia para los items A.1 y A.2. Los cuales van a ser reemplazados por la complejidad de la función que representa el funcionamiento de la poda. La misma va a distribuir a todos los nodos O(n) en un vector de tamaño k, O(k), e ir calculando el valor de *peso_solucion* O(m) (a lo sumo es la suma de todas las aristas) . En total O(m) + O(n) + O(k). Mientras que los siguientes pasos siguen conservando la misma complejidad. Por lo que ahora la complejidad total sería $\mathrm{O}(n^2 * k^n)$ + O(k)