**Pseudocode for a Branch and Bound DFS algorithm for the Heaviest Connected k\_subgraph problem**

HCkS v1.0, 13/05/2020

|  |
| --- |
| **Algorithm 1: Heaviest Connected k\_Subgraph (HCkS)** |
| Input:   * a weighted graph **G**=(**V**, **E**, ω()), where V is a set of vertices, E⊂V×V is a set of edges, and ω:**E**→**R** is the edge weight function*.* (Note: investigate/consider using R+) * k∈**I***, the* maximum number of vertices in the subgraph   Output***:***   * **bestSolutions**, the set of the heaviest connected k-subgraphs  1. Let **L**v be the list of vertices sorted in non-increasing order based on their weighted degree in the graph **G**. (Note: investigate using only the top edges for the weighted degree) 2. Let**L**Ebe the list of edges sorted in non-increasing order based on their weights in the graph **G.** 3. Initialise bestCost= 0 and **bestSolutions** = Ø 4. Initialise the set of visited root vertices **S**r = Ø 5. **For** each **v** ∈ **L**v 6. Initialise a BnB search tree with root node **n**=Node(**v**) and initialise **n** with:  * **VS** ={v}, // vertices of the subgraph (current solution) * **last = v**, **//** last added vertex in the subgraph * **ES** = Ø, // edges of the subgraph (current solution) * **Vc** = Ø,  **//** candidate vertices for extending the solution * **Vx =**  Ø,  **//** excluded vertices as further candidates * w = 0, // subgraph weight: sum of edge weights in **ES** * ub =, where li∈**L**E and li∉**E**S//upper bound  1. n.**Vx =**  n.**Vx** ∪ **Sr** 2. **Sr** = **Sr** ∪ {**v**} 3. **bestSolutions** =  **bestSolutions** ∪ **BnB\_DFS**(**n**) 4. **End For** 5. return **bestSolutions** |
|  |

|  |
| --- |
| **Algorithm 2: BnB\_DFS** |
| Input: **n**, the BnB node to be visited  Output: **bestSolutions**, the heaviest subgraphs (one or more with the same bestCost)   1. Generate a set of candidate vertices **C =** n.**V**c∪Neighbours(**n.lastV**) **\ n.V**S **\ n.V**x 2. **n**.**V**c = C 3. Let **L**c be the list of candidate vertices sorted by the weighted degree: **L**c = sort(**C**) 4. Initialise the set of visited candidate vertices **S**c = Ø 5. **For** each candidate vertex **v**c ∈ **L**c //generate chid nodes from candidates 6. Create a BnB child node **n**c = node(**n**, **C**) and initialise**n**c with these values:  * **VS** = **n**.**VS**, * **ES = n.ES**, * **VC = C**, // known candidates that have an edge to a vertex in **VS** * **V**x= **n.V**x, * w= **n**.w // the weigth of the parent subgraph  1. **n**c**.V**x= **n**c**.V**x ∪ **S**c 2. **nc.**expandSolution(**vc**) // add **vc** to **VS** and add its relevant edges to **E**s 3. **nc.**compute\_UB() 4. **S**c= **S**c∪ {**v**c} // exclude **v**c from future expansions via edges in siblings 5. **If** (***n*c.**w >= bestCost **) Then** // test for best solution 6. **If *(n*c.**w >bestCost**) Then //** new bestCost 7. bestCost = ***n***c.w 8. **bestSolutions** = {<***n*c.VS, *n*c.ES**>} 9. **Else** 10. **bestSolutions** =  **bestSolutions** ∪ {<***n*c.VS, *n*c.ES**>} 11. **End If** 12. **End if** 13. **If** (|***n*c.VS** | < k) and (***n*c.**UB >=bestCost **) Then /**/ expanding vs pruning test 14. **BnB\_DFS**(***n*c**) 15. **End if** 16. **End For** 17. **return bestSolutions** |
|  |

|  |
| --- |
| **Algorithm 3: ExpandSolution(v)** |
| **Input**: **v**, a vertex to be added to the subgraph   1. **For** each **u** ∈ **VS** 2. **If** (**v** adjacent to **u**) **Then** 3. **ES** = **ES** ∪{(**u, v**)} 4. w = w **+**  ω**(u, v)** 5. **End if** 6. **End for** 7. **VS** = **VS** ∪ {**v}** 8. **last = v** |
|  |

|  |
| --- |
| **Algorithm 4: Compute\_UB()** |
| 1. m ***=*** // m is the maximum number of edges that can be still added |
|  |