Algorithm Engineering – Exercise 1

Team 5: Jonas Köppeler, Julia Maria Lydia Henkel, Zia Badar

1. Implemented Features

We have used the same method explained in the lecture of converting a graph into cluster graph. Complexity of our program is $3^k * n^3$, k is the minimum cost to convert graph to cluster graph, n is the no of nodes in graph, 3^k are the search states possible, n^3 is the time for finding a p3.

Psuedo code of our program is:

Algorithm 1 Cluster graph

```
function SOLVE
    while BRANCH(k) != CLUSTER_GRAPH do
        k ← k+1
    end while
end function
//——————————
function BRANCH(k)
    if k < 0 then return NONE
    end if

    u, v, w ← GET_P3

    if BRANCH_EDGE(u, v, k) == CLUSTER_GRAPH then return CLUSTER_GRAPH
    end if
    if BRANCH_EDGE(v, w, k) == CLUSTER_GRAPH then return CLUSTER_GRAPH
    end if
    if BRANCH_EDGE(u, w, k) == CLUSTER_GRAPH then return CLUSTER_GRAPH
    end if

    return NONE
end function
```

2. Data Structures

Graph is implemented as a $n \times n$ adjancency matrix of size , positive weights represents connection, negative weights represents no connection, DO_NOT_DELETE preprocessor directive is replaced with INT32_MAX and represents weights of edge that has been added and should not be removed to avoid cycles in search space, similarly

Algorithm 2 Cluster graph(cont.)

```
function BRANCH_EDGE(u, v, k)
    if WEIGHT(u, v) == ALREADY_MODIFIED then return NONE
    end if

    weight ← WEIGHT(u, v)

    if WEIGHT(u, v) > 0 then
        DELETE_EDGE(u, v)
    end if
    if WEIGHT(u, v) < 0 then
        ADD_EDGE(u, v)
    end if

    if BRANCH(k-abs(weight)) == CLUSTER_GRAPH then return CLUSTER_GRAPH
    end if

    WEIGHT(u, v) ← weight                ▷ backtracking

    return NONE
end function
```

DO_NOT_ADD is replaced with INT32_MIN and represents weight of edge that has been removed and should not be added.

3. Highlights

- Solving a p3 can lead to other p3's getting solved/new p3's getting generated or both.
- Order of solving p3's does not matter if one p3 cannot be solved by exploring all of its three possibilities then solving other p3's will not lead to the unsolved p3 to get solved, therefore we only need to try to solve one of the p3's at a certain search state and decide based on it if solution exists at that state with given budget(k).
- adding and deleting edges is done in O(1) time.
- search for p3's is done in O(n^3)

4. Experiments

Time dependence on the value of n(vertices) and k(optimal cost) is actually not seen from the data acquired from test except for real world data which shows time dependence a little bit. Our algorithm time complexity is O($3^k n^3$)

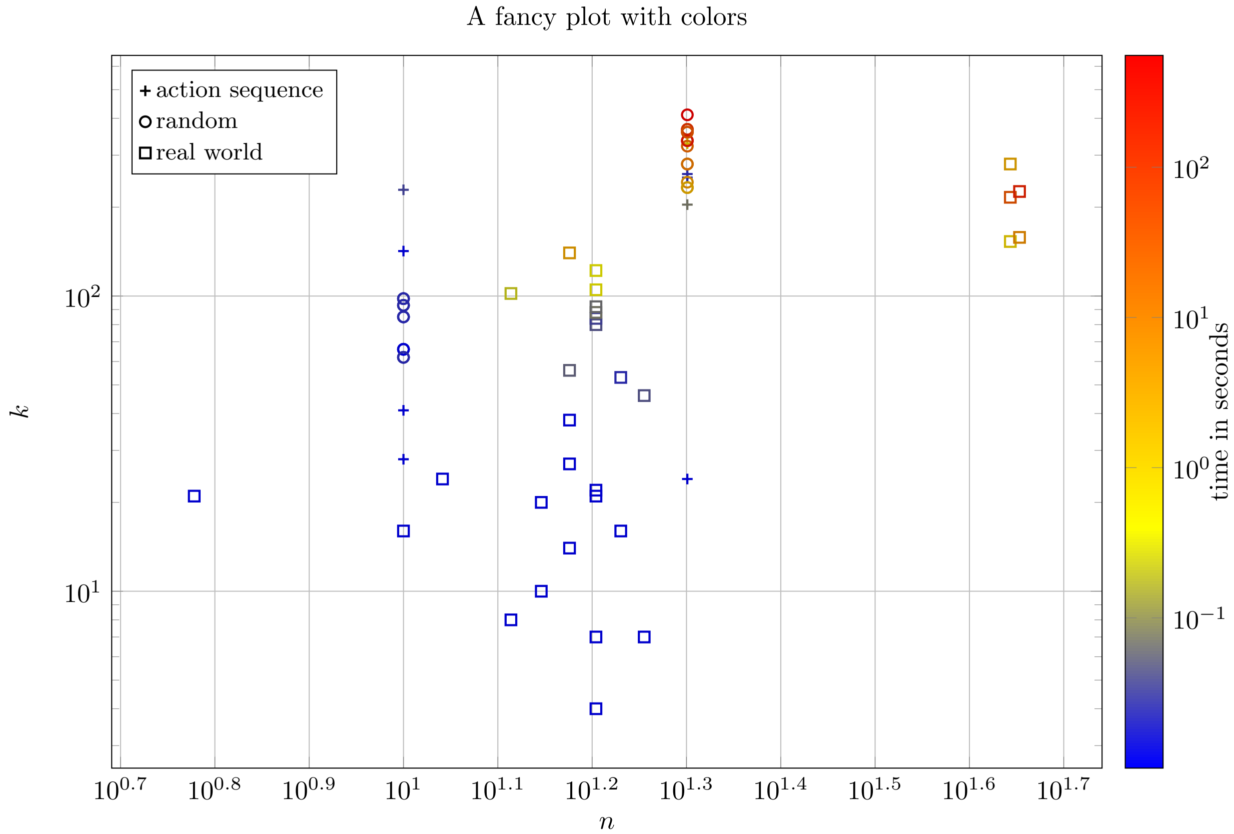

Figure 1: **a**lgorithm ran on different datasets, plus symbol representing action sequence data set, circle symbol representing random data set and square symbol representing real world data set.