Glory Uche Alozie, Kerem Akartunal, Ashwin Arulselvan, Eduardo L. Pasiliao Jr

Efficient methods for the distance-based critical node detection problem in complex networks

Mathematical Optimization 2021/2022

Biagio Licari

Abstract

“An important problem in network survivability assessment is the identification of critical nodes. The distance-based critical node detection problem addresses the issues of internal cohesiveness and actual distance connectivity overlooked by the traditional critical node detection problem. In this study, we consider the distance-based critical node detection problem which seeks to optimise some distance-based connectivity metric subject to budgetary constraints on the critical node set.

We exploit the structure of the problem to derive new path-based integer linear programming formulations that are scalable when compared to an existing compact model. We develop an efficient algorithm for the separation problem that is based on breadth first search tree generation.We have applied our models and algorithm to two different classes of the problems determined by the distance based connectivity functions.

Extensive computational experiments on both real-world and randomly generated network instances, show that the proposed approach is computationally more efficient than the existing compact model especially for larger instances where connections between nodes consist of a small number of hops.”

Application of critical node detection problem

Computational Biology

- The critical node detection problem has been applied to study the interaction of proteins in biological networks.
- Biological organisms such as bacteria consist of sets of interconnected proteins whose interactions form protein-protein interaction networks with nodes as proteins and interactions as edges.
- Critical nodes in the context of protein-protein interaction networks are the proteins that are important to the connectivity of the network.
- In Boginski & Commander (2009) the authors employed the cardinality-constrained critical node problem (CC-CNP) variant to identify critical proteins whose removal would destroy the primary interactions leading to a neutralisation of harmful organisms such as bacteria
- The authors proposed that the application of the CNDP in protein-protein interaction networks is potentially useful in drug design.

Contagion Control

- Critical node detection problem finds application is in contagion control.
- Breaking or limiting connections between nodes of the networks is necessary to contain the spread of these undesirable events.
- The classical CNP has been proposed as an efficient strategy for immunisation of populations against diseases
- Since vaccinating an entire population is impracticable, immunising or isolating the identified critical nodes provides a cost-effective way to halt infection propagation.

Problem description and base formulation

DCNDP

Problem definition

- Let $G = (V, E)$ be a simple undirected unweighted graph with a finite set V of nodes and a finite set $E \subseteq VxV$ of edges where $n := \mid V \mid$ and $m := \mid E \mid$
- For any given node i, denote by $\mathcal{N}_{\mathcal{G}}(i)$ the set of all neighbours of i
- Node i is said to be connected to another node j if there is a path between them. A shortest path between i and j is a path containing the least number of edges among all paths connecting i and j.
- The number of edges of a shortest path between i and j in G is known as the hop distance
- The maximum distance between any pair of nodes is known as the diameter of G
- The goal of the distance-based critical node detection problem is to find a subset of nodes $R \subset V$ with $\left|R\right| \leq B$, whose removal minimises some distance-based connectivity measure $f(d)$

DCNDP

Classes of the distance-based critical node detection problem

Class 1. *Minimise the number of node pairs connected by a path of length at most k:*

$$f(d) = \begin{cases} 1, & if\ d \leq k \\ 0, & if\ d > k \end{cases}$$

where k is a given positive integer representing the cut-off hop distance . I will refer to the DCNDP of this class as DCNDP_1

Class 2. *Minimise the Harary index or efficiency of the graph:*

$$f(d) = \begin{cases} d^{-}1, & if\ d \leq L \\ 0, & otherwise \end{cases}$$

This metric is based on the assumption in communication network analysis that communication efficiency between node pairs is inversely proportional to the distance between them. I will refer to the DCNDP of this class as DCNDP_1

DCNDP

Premise

- Associated with any node $i \in V$, define a variable x_i as:

$$x_i = \begin{cases} 1, & if\ node\ i\ is\ deleted \\ 0, & if\ otherwise \end{cases}$$

- Similarly, define nnectivity variables by:

$$y_{i,j} = \begin{cases} 1, & if\ (i,j)\ are\ connected\ by\ a\ path\ \leq l\ in\ G^r \\ 0, & otherwise \end{cases}$$

New integer programming formulations

DCNDP_2

New Integer Programming Formulations

- 3.21 ensure that node pairs (i,j) are L-distance disconnected iff at least one node along all paths of length less or equal to L connecting i and j is deleted.
- Explicitly model the non-redundant constraints $y_{ij} + x_i + xj \geq 1$ for edges and leave the rest to be identified in a separation routine (BFS).
- 3.22 ensure that there is a path of length /, if there is a path of length /–1
- 3.24 represents the budgetary constraint limiting the cardinality of the critical node set
- 3.25-3.26 specify the domains of the decision variables.

$$\text{Minimise} \quad \sum_{i,j \in V: i<j} \left(f(1) y_{ij}^1 + \sum_{\ell=2}^{L} f(\ell) \left(y_{ij}^{\ell} - y_{ij}^{\ell-1} \right) \right) \tag{3.19}$$

$$\text{s.t.} \quad \sum_{r \in V(P)} x_r + y_{ij}^{|P|} \geq 1, \quad \forall P \in \mathcal{P}_L(i,j),\ i,j \in V,\ i<j \tag{3.21}$$

$$y_{ij}^{\ell-1} \leq y_{ij}^{\ell}, \quad \forall (i,j) \in V,\ i<j,\ \ell \in \{2,\ldots,L\} \tag{3.22}$$

$$y_{ij}^{\ell} = y_{ij}^{1}, \quad \forall (i,j) \in E,\ i<j,\ \ell \in \{2,\ldots,L\} \tag{3.23}$$

$$\sum_{i \in V} x_i \leq B \tag{3.24}$$

$$x_i \in \{0,1\}, \quad \forall i \in V \tag{3.25}$$

$$y_{ij}^{\ell} \in \{0,1\}, \quad \forall (i,j) \in V,\ i<j,\ \ell \in \{1,\ldots,L\} \tag{3.26}$$

DCNDP_1

Formulation and relaxation

$$\text{Minimise} \quad \sum_{i,j \in V : i<j} y_{ij} \tag{3.28}$$

$$\text{s.t.} \quad \sum_{r \in V(P)} x_r + y_{ij} \geq 1, \quad \forall P \in \mathcal{P}_k(i,j),\ (i,j) \in V,\ i<j \tag{3.29}$$

$$\sum_{i \in V} x_i \leq B \tag{3.30}$$

$$x_i \in \{0,1\}, \quad \forall i \in V \tag{3.31}$$

$$y_{ij} \in \{0,1\}, \quad \forall (i,j) \in V,\ i<j \tag{3.32}$$

Objective Function 3.28 minimises the number of node pairs connected by a path of hop distance at most k.
Constraints 3.29 - 3.31 similar to MIP formulation

- For DCNDP_1 the objective function does not explicitly make use of the path lengths indexed by / in the *y* variables
- This enables us to eliminate the /- index in the distance connectivity variable $y^l_{i,j}$ used in the path-based model
- computationally more efficient solutions
- New sets of connectivity variables:

$$y_i j = \begin{cases} 1, & if\ (i,j)\ are\ connected\ by\ a\ path\ \leq k\ in\ G^r \\ 0, & otherwise \end{cases}$$

Solution Methods

DCNDP

Solution Methods

- Use of a custom approach to separate violated lazy cuts.
- This approach involves in generating a breadth-first-search tree for every node $v \in V$
- For DCNDP_1, since we are only interested in paths up to a specific length k, we stop the traversal up to that particular depth
- For DCNDP_2, , we continue traversal up to depth L. In the BFS tree, we identify the path from the root node *i* to the nodes in level *l* = 1,...,L and if this path is violated, we add it as a cut
- The generation of these BFS trees is much more efficient
- BFS tree can be generated in $O(|E|)$ as compared to solving shortest path that will take $O(|E| + |V| \log |V|)$

DCNDP

BFS

- The algorithm begins by selecting a root node r from the set of candidate root nodes and constructs a BFS tree up to a specified depth (or tree level) L.
- The algorithm follows a standard BFS algorithm with the only difference being that it keeps track of the depth of each discovered node (lines 6 & 10).
- For each node t that is being visited, the algorithm proceeds in one of three possible ways depending on its tree level l[t] :
  - $l[t] = 1$ implies that node t is a direct neighbour of the root node r that is $(r, t) \in E$.
  - $1 < l[t] \leq L$, implies that $(r, t) \notin E$ but hop distance between root node r and t is less than or equal to L. This is the case of interest. Hence we not only add t to the queue Q and mark it as visited, but we also check if path inequalities(3.21 or 3.29) of the corresponding shortest path P_{rt} are violated (lines 14-17) and add the violated inequality to the queue Q_C.
  - $l[t] > L$, implies that all nodes reachable within hop distance L from the root r have been explored

Algorithm 1: Separation algorithm for the proposed model

1 **Input :** Graph $G = (V, E)$, Incumbent $(\tilde{x}, \tilde{y})$ or LP-relaxation solution $(\bar{x}, \bar{y})$, set of candidate root nodes R_c, and tree depth L

2 **Output:** Queue of violated inequalities Q_c

3 **for** $r \in R_c$ **do**

4 $Q \leftarrow \{r\}$; // initialise queue Q with root node r

5 $T \leftarrow \{r\}$; // mark r as visited

6 $l[r] = 0$;

7 **while** $Q \neq \emptyset$ **do**

8 $s \leftarrow Q.remove$; // retrieve the first element in queue Q

/* Explore node s, where δ_s denotes neighbours of s */

9 **for** $t \in \delta_s \setminus T$ **do**

10 $l[t] = l[s] + 1$;

11 **if** $l[t] = 1$ **then**

12 $Q.add(t)$; // add t to queue Q

13 $T \leftarrow T \cup \{t\}$; // mark t as visited

14 **else if** $l[t] \in [2, L]$ **then**

15 $Q.add(t)$;

16 $T \leftarrow T \cup \{t\}$; // mark t as visited

/* check violation of (3.29) (resp. (3.21) for DCNDP-2b) for the path P_{rt} */

17 **if** $\sum_{i \in V(P_{rt})} \tilde{x}_i + \tilde{y}_{rt} < 1$ *(resp.* $\sum_{i \in V(P_{rt})} \tilde{x}_i + \tilde{y}^{l[t]}_{rt} < 1$*)* **then**
$Q_c.add(\sum_{i \in V(P_{rt})} x_i + y_{rt} \geq 1)$
(resp. $Q_c.add(\sum_{i \in V(P_{rt})} x_i + y^{l[t]}_{rt} \geq 1)$);

end

18 **else**

19 $Q \leftarrow \emptyset$;

20 break;

Callback & BFS implementation

```python
# bfs tree generation to separate integer solution
def first_depth_k_bfs(input_graph, cost, connect, L, cut_limit, model):
    global cut_count
    roots = [n for (n, attr) in cost.items() if attr == 0]
    for root in roots:
        # keep track of all visited nodes and nodes to be checked
        visited, queue = {root}, collections.deque([root])
        # this dict keeps track of levels
        levels = {root: 0}

        # this dict keeps track of predecessors
        predecessor = {root: -1}
        cut_count = 0

        # keep looping until there are no nodes still to be checked
        while queue:
            if cut_count > cut_limit:
                break
            # pop first node from the queue
            vertex = queue.popleft()
            for neighbour in input_graph[vertex]:
                # only consider neighbors with solution=0
                if neighbour not in visited and cost[neighbour] <= 1e-5:
                    new_levels = levels[vertex] + 1
                    if new_levels == 1:  # direct neighbours of root node
                        predecessor[neighbour] = vertex
                        levels[neighbour] = new_levels
                        # mark neighbours of node as visited to avoid revisiting
                        visited.add(neighbour)
                        # add neighbours of node to queue
                        queue.append(neighbour)
                    elif new_levels in range(2, L + 1):
                        predecessor[neighbour] = vertex
                        levels[neighbour] = new_levels
                        # mark neighbours of node as visited to avoid r
                        visited.add(neighbour)
                        # add neighbours of node to queue
                        queue.append(neighbour)
                        i = min([root, neighbour])
                        j = max([root, neighbour])
                        if connect[(i, j, new_levels)] < 1 - 1e-5:
                            lazy_cut_lhs = LinExpr(0)
                            lazy_cut_lhs.add(model._x_delete[root])
                            while neighbour != root:
                                lazy_cut_lhs.add(model._x_delete[neighb
                                nxt = predecessor[neighbour]
                                neighbour = nxt
                            model.cbLazy(lazy_cut_lhs + model._u_connec
                            cut_count += 1

                    else:  # new_levels >k
                        break
    else:
        continue
    break
```

```
if where == GRB.Callback.MIPSOL:  # if integer solution
    # get MIPSOL_OBJBDN = Current best objective bound
    bound_save = model.cbGet(GRB.callback.MIPSOL_OBJBND)
    for j in G.nodes():
        # retrieve values from the new MIP solution
        cost[j] = abs(model.cbGetSolution(model._x_delete[j]))
        for i in G.nodes():  # range(ind,j):
            if i < j:
                for l in range(1, L + 1):
                    connect[(i, j, l)] = abs(model.cbGetSolution(model._u_connect[i, j, l]))

    first_depth_k_bfs(G, cost, connect, L, GRB.INFINITY, model)
```

```python
# if fractional solution
elif where == GRB.Callback.MIPNODE:
    # Optimization status of current MIP node if it is OPTIMAL
    if model.cbGet(GRB.Callback.MIPNODE_STATUS) == GRB.Status.OPTIMAL:
        # get current best objective bound
        current_bound = model.cbGet(GRB.callback.MIPNODE_OBJBND)
        # get current explored node count
        node_count = int(model.cbGet(GRB.callback.MIPNODE_NODCNT))
        if bound_save == current_bound:
            bound_check += 1
            if bound_check >= 5 or node_count > 0:
                bound_check = 0
            else:
                for j in G.nodes():
                    cost[j] = abs(model.cbGetNodeRel(model._x_delete[j]))
                    G.nodes[j]['LPsol'] = cost[j]  # set node attributes to lp solution
                    for i in G.nodes():
                        if i < j:
                            for l in range(1, L + 1):
                                connect[(i, j, l)] = abs(model.cbGetNodeRel(model._u_connect[i, j, l]))
                roots = [n for (n, attr) in cost.items() if attr < 1]
                for rt_node in roots:
                    second_depth_k_bfs(G, cost, connect, rt_node, L, 300, model)
```

```
else:
    bound_save = current_bound
    for j in G.nodes():
        cost[j] = abs(model.cbGetNodeRel(model._x_delete[j]))
        G.nodes[j]['LPsol'] = cost[j]  # set node attributes to lp solution
        for i in G.nodes():
            if i < j:
                for l in range(1, L + 1):
                    connect[(i, j, l)] = abs(model.cbGetNodeRel(model._u_connect[i, j, l]))
    roots = [n for (n, attr) in cost.items() if attr < 1]
    for rt_node in roots:
        second_depth_k_bfs(G, cost, connect, rt_node, L, 300, model)
```

DCNDP

Model implementation examples

$$Min\,.\sum_{i,j\in V:i<j}(\,f(1)y_{i,j}^{1}+\sum_{l=2}^{L}f(l)(y_{i,j}^{l}-y_{i,j}^{l-1}))$$

```
# objective
obj = LinExpr(0)
for j in H.nodes():
    for i in H.nodes():
        if i < j:
            obj.add(f[0] * u_connect[i, j, 1])
            for l in range(1, L):
                obj.add(f[l] * (u_connect[i, j, l + 1] - u_connect[i, j, l]))
```

$$ST\,.\sum_{i\in V}x_i\leq B$$

```
# constraint on number of critical nodes
model.addConstr(sum((x_delete[j]) for j in H.nodes()) <= C)
```

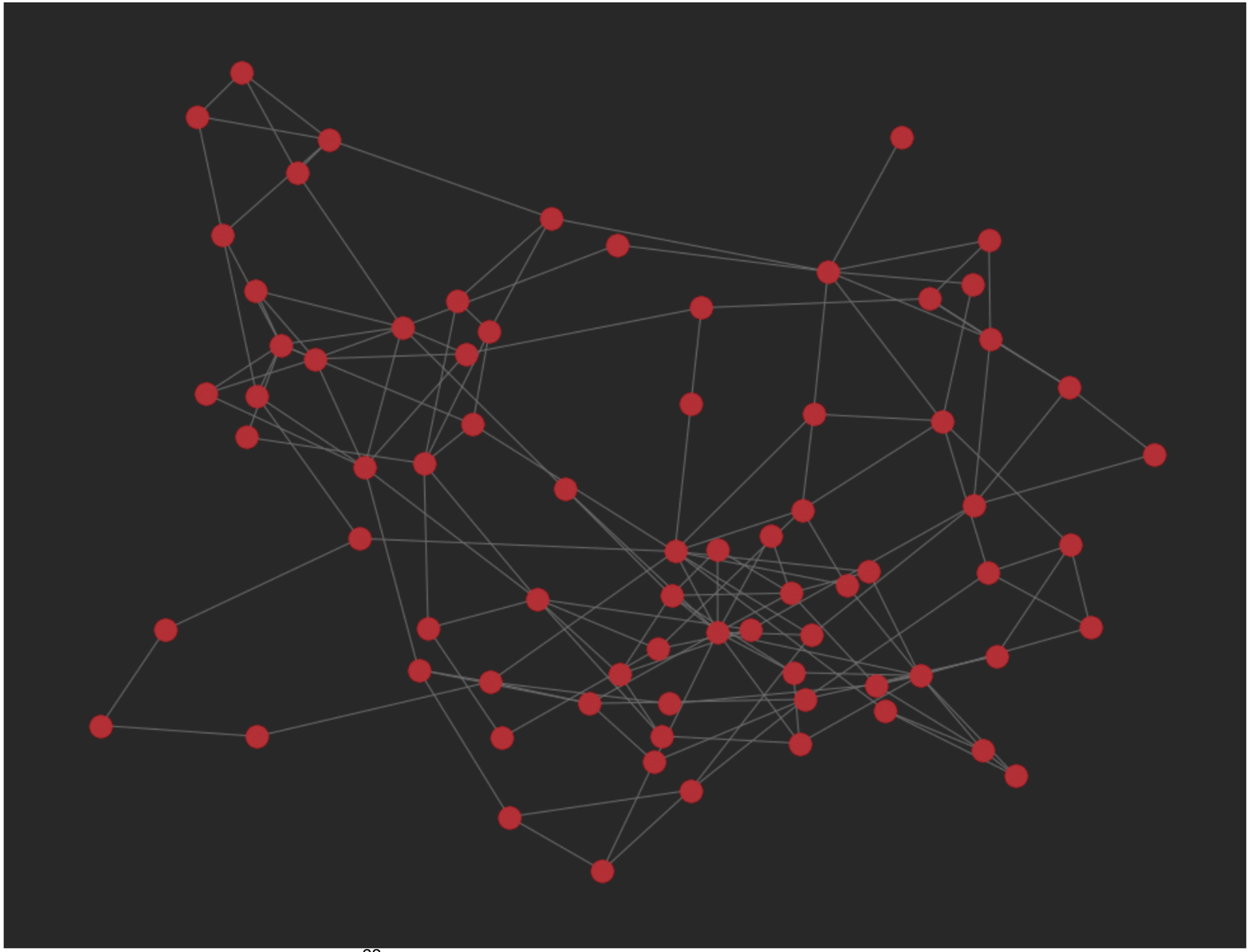
$$y_{ij}^{l}=y_{i,j}^{1},\quad\forall(i,j)\in E,\quad i<j,\quad l\in 2,....,L$$

$$y_{i,j}^{l}+x[i]+x[j]\geq 1,\quad(i,j)\in E,\quad i<j$$

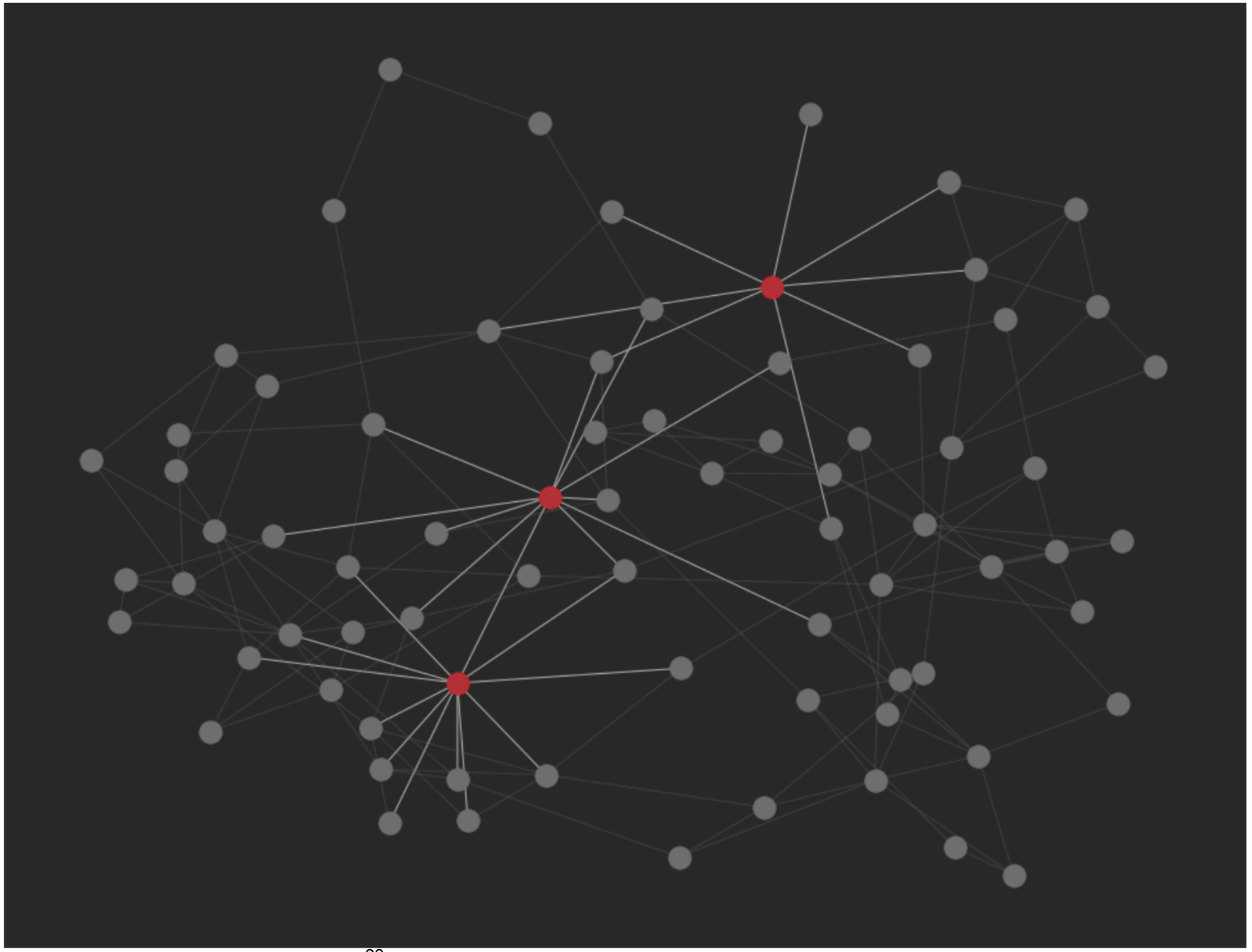
```
# constraints on (i,j) in E
for (i, j) in H.edges():
    if i < j:
        model.addConstr(u_connect[i, j, 1] + x_delete[i] + x_delete[j] >= 1)
        for l in range(2, L + 1):
            model.addConstr(u_connect[i, j, 1] == u_connect[i, j, l])
```

DCNDP

- DCNDP_1 example with B=0.05n
- Graph : Sanjuansur
- First image is the original graph
- Second image is graph without critical nodes after optimization



DCNDP



Computational experiments

Experiments

Hardware & Software

- Computational studies performed on a Macbook Pro with Apple M1, 8GB of Ram and MacOS 12.4
- Models and Algorithms written in Python 3.9, using Gurobi 9.5.2 as optimisation suite.
- NetworkX used for random graph generation and manipulating of the graphs
- 3 class of random graph generators used:
  - Barabasi-Albert
  - Erdos-Renyi
  - Uniform

Experiments

Test Instances

- Test instances comprise both real-world and randomly generated graphs
- Real-world instances are a subset of networks from the Pajek and UCINET datasets

Experiments

Real World instance

Graph	N	M	Diameter	% k-Conn	Note
Hi-Tech	33	91	5	88.3	Friendship network of employees in a hi-tech firm
Karate	34	78	5	85.6	A social network of a karate club at a U.S University(1970)
Mexican	35	115	4	98.0	A network of relations (family, political and business) of political elite in Mexico
Chesapeake	39	170	3	100.0	Chesapeake Bay Mesohaline network
Dolphins	62	159	8	58.5	A social network that represents frequent associations between dolphins
LesMiserable	77	254	5	85.4	Network of co-appearance of characters in the novel Les Miserable
Santafe	118	200	12	32.9	Collaboration network of scientists at the Santa Fe Institute
Sanjuansur	75	155	7	48.7	Social networks of families in a rural area in Costa Rica
Attiro	59	128	8	68.0	-
LindeStrasse	232	303	13	12.1	Network of friendly relationships between characters of the soap opera
SmallWorld	233	994	4	95.2	A citation networ
Netscience	379	914	17	13.3	Co--authorship network of scientists in science
UsaAir97	332	2126	6	84.8	Transportation network of US airlines

Experiments

Synthetic World instance

Graph	N	M	Diameter	%density	% k-Dist Connectivity	%efficiency
BA2	100	200	4	18	99,9	49,6
ER2	200	1004	4	5	97.6	42,8
GNM1	300	1500	4.4	3.3	94,1	39,6
GNM2	300	2000	4	4.5	99,6	43,4

Experiments

Parameters & Settings

- For DCNDP-1, where we minimise the number of node pairs connected by a path of length at most k, the threshold distance is k = 3.
- For DCNDP-2 which minimises the Harary index or efficiency, L is equal to the diameter of the input graph.
- Budget B btw 1% to 10% of graph size for real-world graph
- Budget B btw 5% to 10% of graph size for synthetic-world graph

Experiments

Result legend

- The computational experiments were performed for varying sizes (nodes, edges) and classes of graphs with different edge densities and diameters.
- In each table, along with graph characteristics such as number of nodes, edges and diameter, there is present computational times in seconds and/or percentage gaps for different B
- Columns labelled InitObj represent the initial objective values in the input graph prior to solving the mode:
  - For DCNDP-1, this is the percentage of node pairs connected by paths of length at most k
  - For DCNDP-2, it is the initial communication efficiency of the input graph
- Columns labelled FinObj represent the final objective value (in percentage) at the end of optimisation or the best objective realised
- Columns labelled call_back_time represent the computational time of callback BFS
- Columns labelled run_time represent the total computational time

Experiments

DCNDP_1 on real-world with B = 0.05n

graph	n_nodes	n_edges	diameter	init_obj	final_obj	cost	n_vars_sol	distance-based pairwise connectivity	critical_node	callback_ time (s)	run_time (s)
Hi-Tech	33	91	5	88,30%	75,19%	1	561	397	[2]	0.05	0.12
Karate	34	78	5	85,6%	57,75%	1	595	324	[1]	0.04	0.07
Mexican	35	117	4	98,0%	88,57%	1	630	527	[12]	0.07	0.16
Sawmill	36	62	8	63,0%	34,1%	1	666	215	[12]	0.03	0.07
Chesapeake	39	170	3	100,0%	93,93%	1	780	696	[39]	0,2	0,32
Dolphins	62	159	8	58,5%	43,36%	3	1953	820	[36, 40, 51]	0,48	0,88
Lesmiserable	77	254	5	85,4%	31,78%	3	3003	930	[11, 23, 27]	0,33	0,73
Santafe	118	200	12	32,9%	4,42%	5	7021	305	[1, 2, 12, 6, 24]	0,47	0,69
Sanjuansur	75	155	7	48,7%	28,94%	3	2850	803	[57, 34, 41]	0,13	0,24
Attiro	59	128	8	68,00%	43,42%	2	1770	743	[53, 35]	0.09	0.16
UsaAir97	332	2126	6	84,8%	19,33%	16	55278	10623	[118, 201, 258, 182, 152, 47, 144, 65, 261, 255, 248, 166, 112, 313, 67, 230]	17,29	189,18
SmallWorld	233	994	4	95,2%	17,13%	11	27261	4629	[1, 3, 6, 14, 55, 215, 239, 240, 243, 252, 265]	31,1	69,2

Experiments

DCNDP_1 on real-world with B = 0.1n

graph	n_nodes	n_edges	diameter	init_obj	final_obj	cost	n_vars_sol	distance-based pairwise connectivity	critical_node	callback_time (s)	run_time (s)
Hi-Tech	33	91	5	88,30%	55,49%	3	561	297	[2, 21, 19]	0,19	0,36
Karate	34	78	5	85,6%	26,2%	3	595	147	[1, 33, 34]	0,03	0,05
Mexican	35	117	4	98,0%	60,17%	3	630	358	[18, 10, 12]	0,06	0,16
Sawmill	36	62	8	63,0%	21,43%	3	666	135	[12, 36, 31]	0,02	0,04
Chesapeake	39	170	3	100,0%	69,1%	3	780	512	[39, 36, 38]	0,29	0,55
Dolphins	62	159	8	58,5%	30,83%	6	1953	583	[14, 17, 20, 29, 37, 40]	0,64	1,22
Lesmiserable	77	254	5	85,4%	11,04%	7	3003	323	[11, 23, 25, 26, 27, 48, 55]	0,36	0,57
Santafe	118	200	12	32,9%	1,68%	11	7021	116	[1, 2, 12, 15, 22, 72, 53, 4, 6, 24, 9]	0,3	0,43
Sanjuansur	75	155	7	48,7%	16,47%	7	2850	457	[57, 34, 41, 68, 72, 39, 40]	0,19	0,42
Attiro	59	128	8	68,00%	25,95%	5	1770	444	[53, 47, 51, 35, 42]	0,15	0,35
UsaAir97	332	2126	6	84,8%	5,64%	33	55278	3100	[1, 3, 6, 2, 53, 4, 6, 24 14, 30, 55, 215, 236, 238, 239, 240, 243, 246, 252, 265, 271, 279, 285, 322, 353, 391, 67, 287]	216,76	872,71
SmallWorld	233	994	4	95,2%	6,27%	23	27261	1694	[1, 3, 6, 14, 30, 55, 215, 236, 238, 239, 240, 243, 246, 252, 265, 271, 279, 285, 322, 353, 391, 67, 287]	55,42	170,52

Experiments

DCNDP_2 on real-world with B = 0.05n

graph	n_nodes	n_edges	diameter	init_obj	final_obj	cost	n_vars_sol	distance-based pairwise connectivity	critical_node	callback_time (s)	run_time (s)
Hi-Tech	33	91	5	50,8%	43,69%	1	2673	230,67	[2]	0,1	0,36
Karate	34	78	5	49,20%	33,74%	1	2839	189,27	[53, 35]	0,02	0,11
Mexican	35	117	4	54,9%	49,09%	1	2415	291,92	[12]	0,07	0,27
Sawmill	36	62	8	39,9%	27,46%	1	5076	173	[12]	0,04	0,23
Chesapeake	39	170	3	60,3%	53,71%	1	2262	398	[39]	0,09	0,25
Dolphins	62	159	8	37,9%	29,33%	3	15190	554,83	[17, 33, 51]	1,68	12,46
Lesmiserable	77	254	5	43,2%	18,43%	3	14707	539,53	[11, 23, 27]	0,39	1,97
Santafe	118	200	12	27,09%	2,95%	5	82954	203,82	[1, 2, 12, 6, 24]	0,4	1,24
Sanjuansur	75	155	7	34,2%	25,90%	3	19500	71861	[57, 34, 70]	1,22	8,18
Attiro	59	128	8	38,9%	31,11%	2	13747	532	[53, 35]	0,55	2,29

Experiments

DCNDP_2 on real-world with B = 0.1n

graph	n_nodes	n_edges	diameter	init_obj	final_obj	cost	n_vars_sol	distance-based pairwise connectivity	critical_node	callback_time (s)	run_time (s)
Hi-Tech	33	91	5	50,8%	32,81%	3	2673	173,23	[53, 47, 51, 35, 8]	0,11	0,4
Karate	34	78	5	49,20%	16,69%	3	2839	93,65	[1, 33, 34]	0,04	0,16
Mexican	35	117	4	54,9%	36,59%	3	2415	217,67	[18, 10, 12]	0,06	0,21
Sawmill	36	62	8	39,9%	14,17%	3	5076	89,23	[12, 36, 27]	0,04	0,09
Chesapeake	39	170	3	60,3%	35,87%	3	2262	265,83	[39, 36, 38]	0,09	0,25
Dolphins	62	159	8	37,9%	18,63%	6	15190	352,83	[28, 30, 36, 40, 45, 51]	0,58	1,38
Lesmiserable	77	254	5	43,2%	7,88%	7	14707	230,58	[11, 23, 25, 26, 27, 48, 55]	0,41	1,97
Santafe	118	200	12	27,09%	1,39%	11	82954	95,87	[1, 2, 12, 15, 22, 72, 53, 4, 6, 24, 9]	0,58	1,24
Sanjuansur	75	155	7	34,2%	14,41%	7	19500	399,87	[3, 57, 34, 68, 70, 72, 46]	0,68	7,9
Attiro	59	128	8	38,9%	22,30%	5	13747	381.62	[53, 47, 51, 35, 8]	0,66	7,18

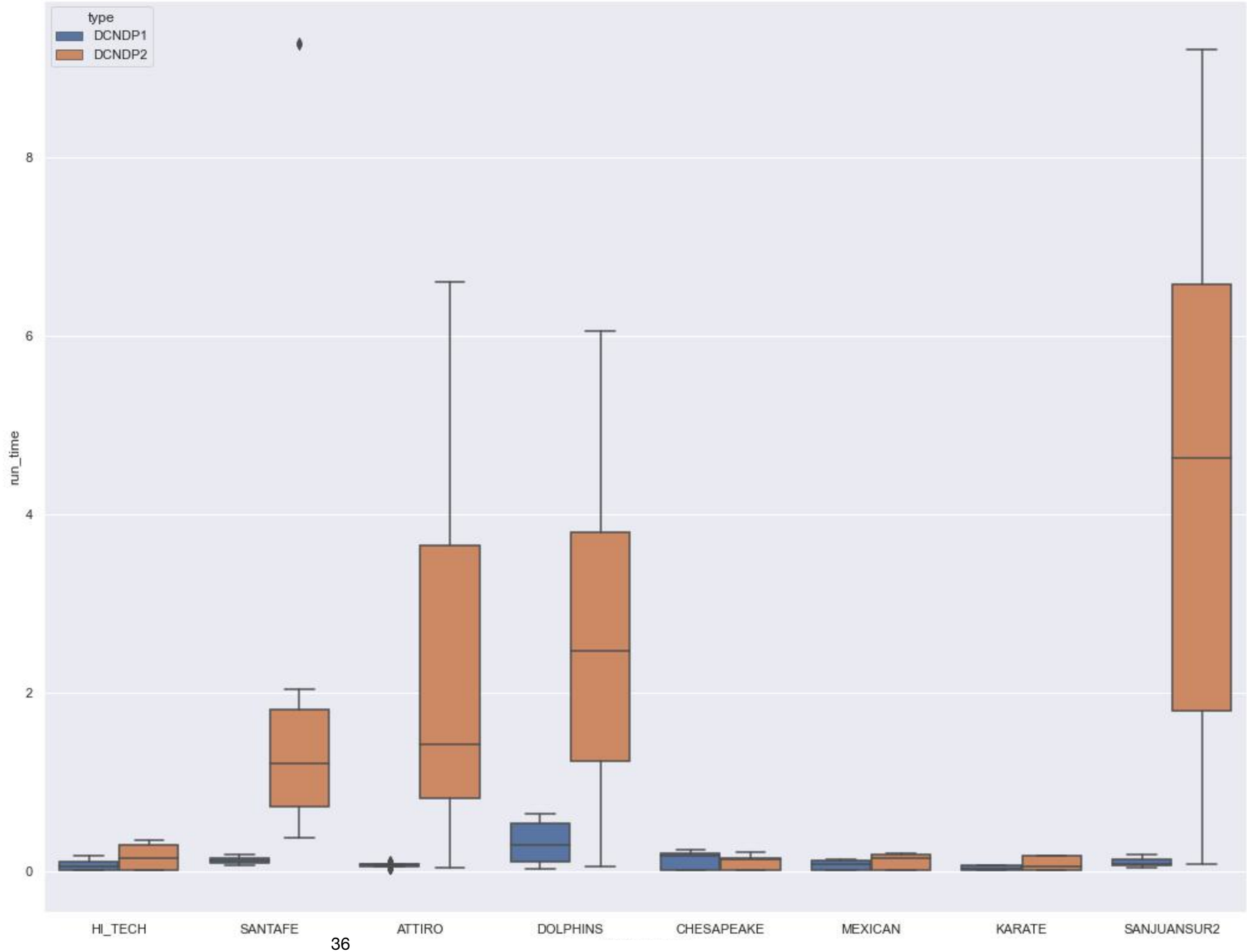
Experiments

DCNDP_2 vs DCNDP_1

				DCNDP_1				DCNDP_2			
graph	n_nodes	n_edges	diameter	init_obj	final_obj	callback_time (s)	run_time (s)	init_obj	final_obj	callback_time (s)	run_time (s)
Mexican_5	35	117	4	98,0%	88,57%	0.07	0.16	54,9%	49,09%	0,07	0,27
Mexican_10	35	117	4	98,0%	60,17%	0,06	0,16	54,9%	36,59%	0,06	0,21
Sawmill_5	36	62	8	63,0%	34,1%	0.03	0.07	39,9%	27,46%	0,04	0,23
Sawmill_10	36	62	8	63,0%	21,43%	0,02	0,04	39,9%	14,17%	0,04	0,09
Chesapeake_5	39	170	3	100,0%	93,93%	0,2	0,32	60,3%	53,71%	0,09	0,25
Chesapeake_10	39	170	3	100,0%	69,1%	0,29	0,55	60,3%	35,87%	0,09	0,25
Dolphins_5	62	159	8	58,5%	43,36%	0,48	0,88	37,9%	29,33%	1,68	12,46
Dolphins_10	62	159	8	58,5%	30,83%	0,64	1,22	37,9%	18,63%	0,58	1,38
Lesmiserable_5	77	254	5	85,4%	31,78%	0,33	0,73	43,2%	18,43%	0,39	1,97
Lesmiserable_10	77	254	5	85,4%	11,04%	0,36	0,57	43,2%	7,88%	0,41	1,97
Santafe_5	118	200	12	32,9%	4,42%	0,47	0,69	27,09%	2,95%	0,4	1,24
Santafe_10	118	200	12	32,9%	1,68%	0,3	0,43	27,09%	1,39%	0,58	1,24
Sanjuansur_5	75	155	7	48,7%	28,94%	0,13	0,24	34,2%	25,90%	1,22	8,18
Sanjuansur_10	75	155	7	48,7%	16,47%	0,19	0,42	34,2%	14,41%	0,68	7,9

Experiments

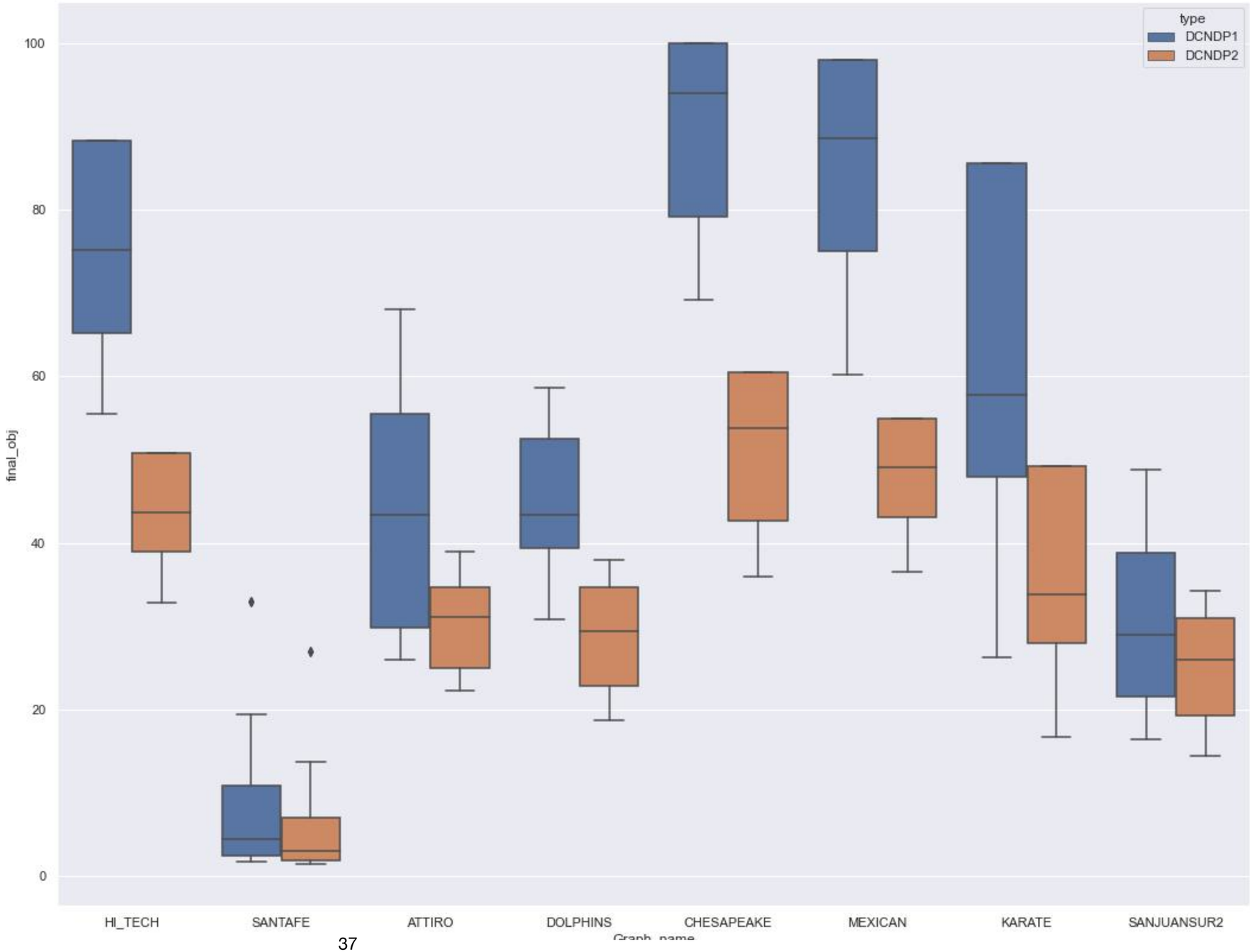
DCNDP_1 vs DCNDP_2 run_time difference on small/ medium real-world graph



Experiments

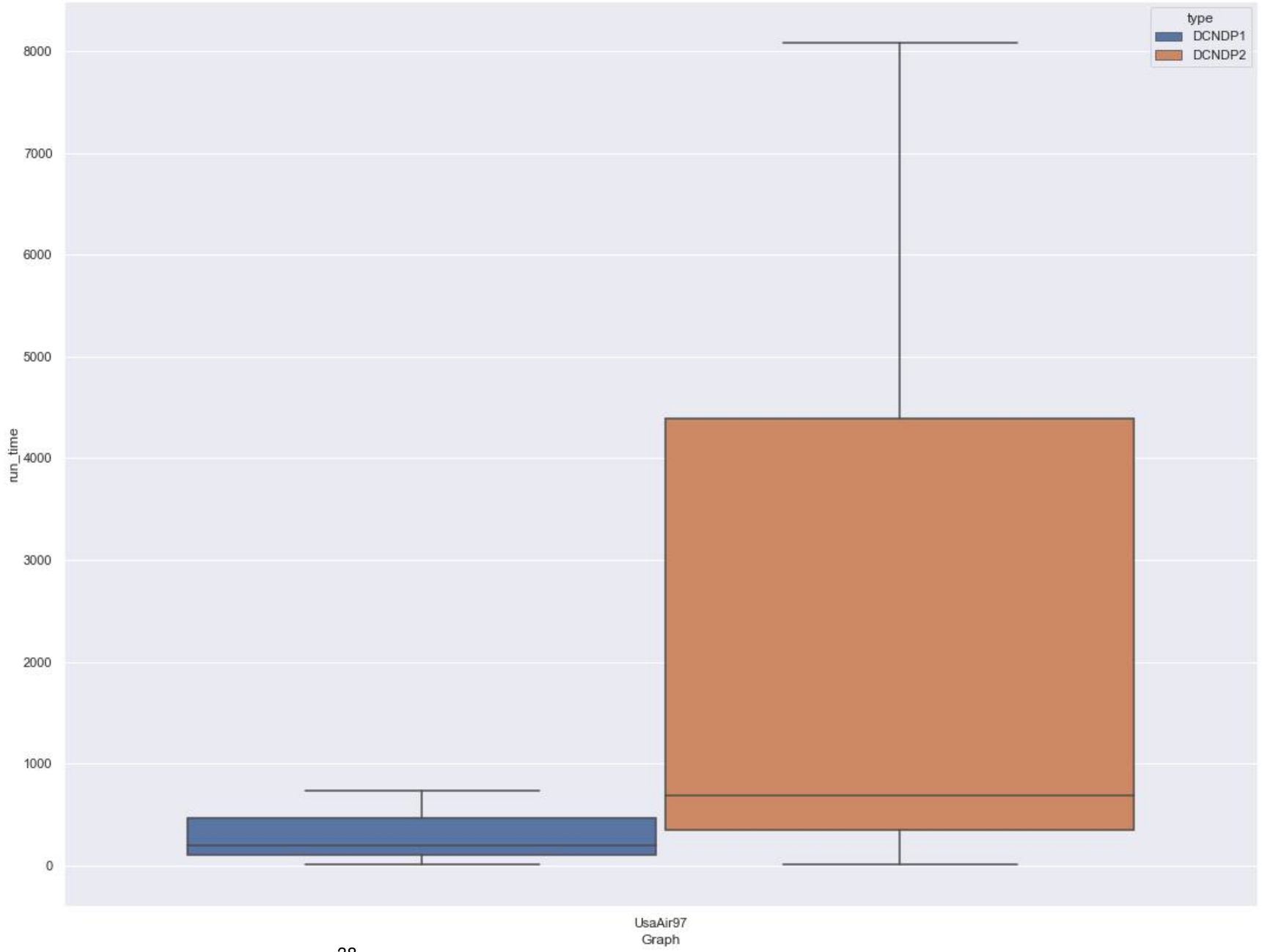
DCNDP_1 vs DCNDP_2

%gap difference



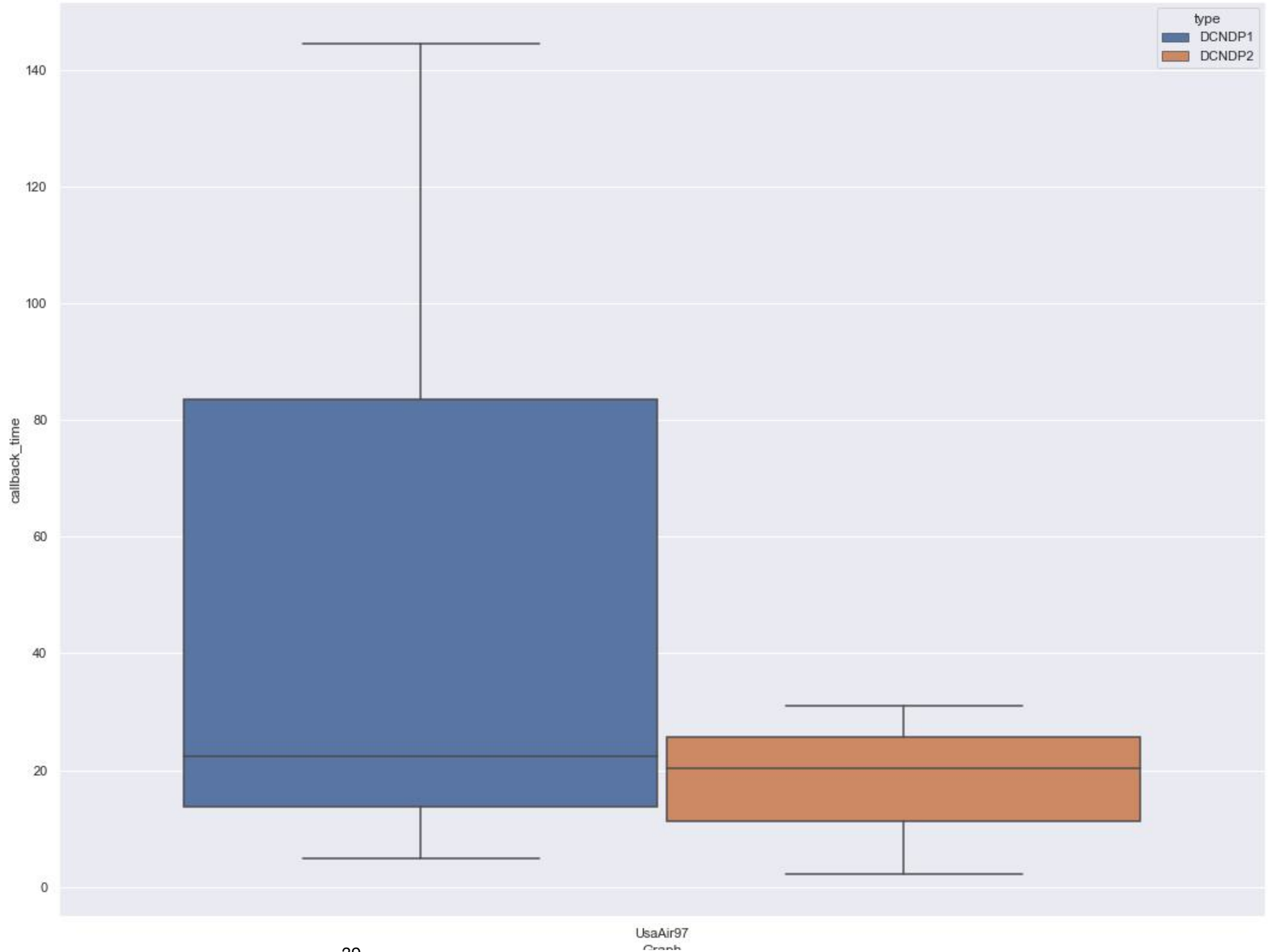
Experiments

Big real world test.
Runtime UsaAir97



Experiments

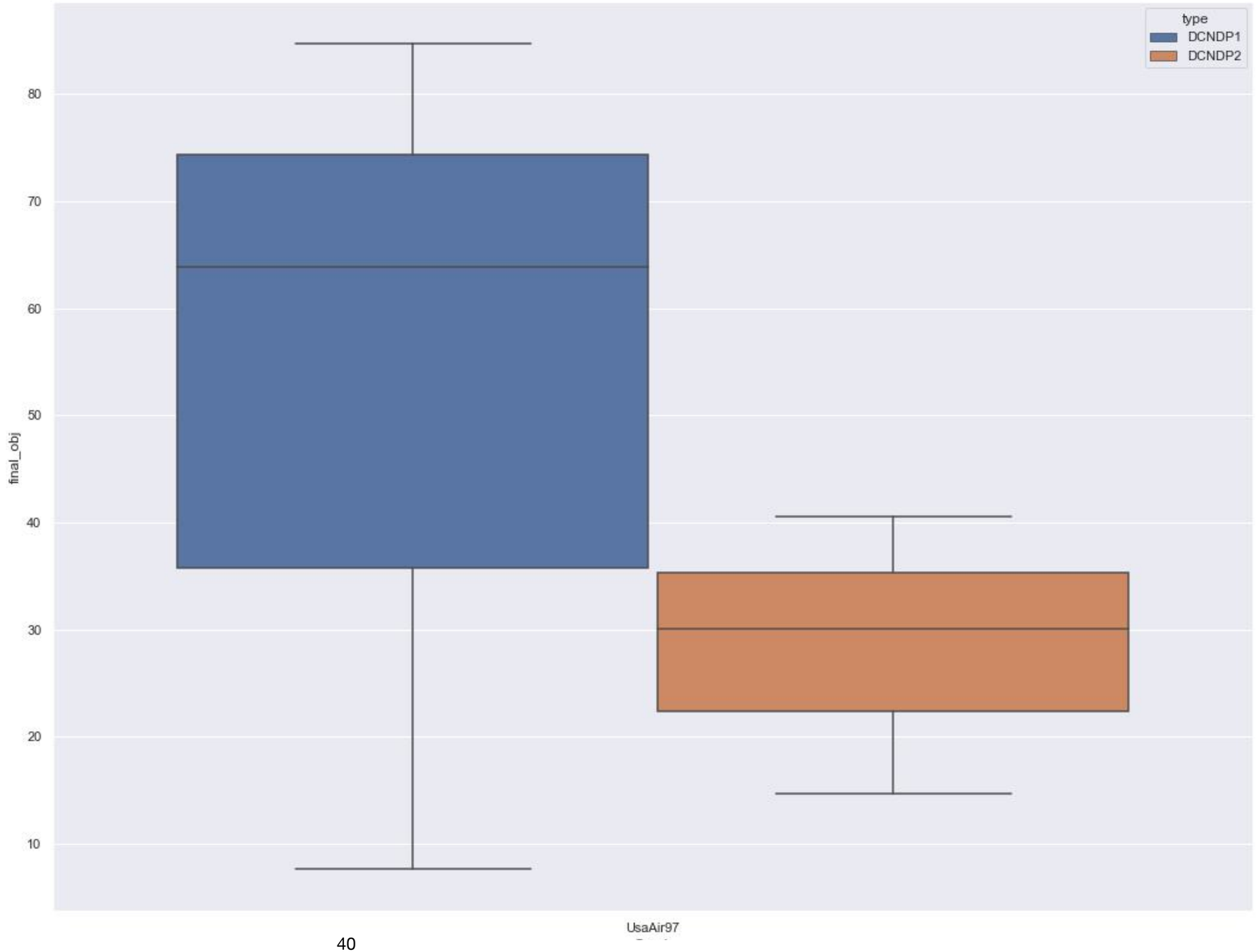
Big real world test.
Callback time UsaAir97



Experiments

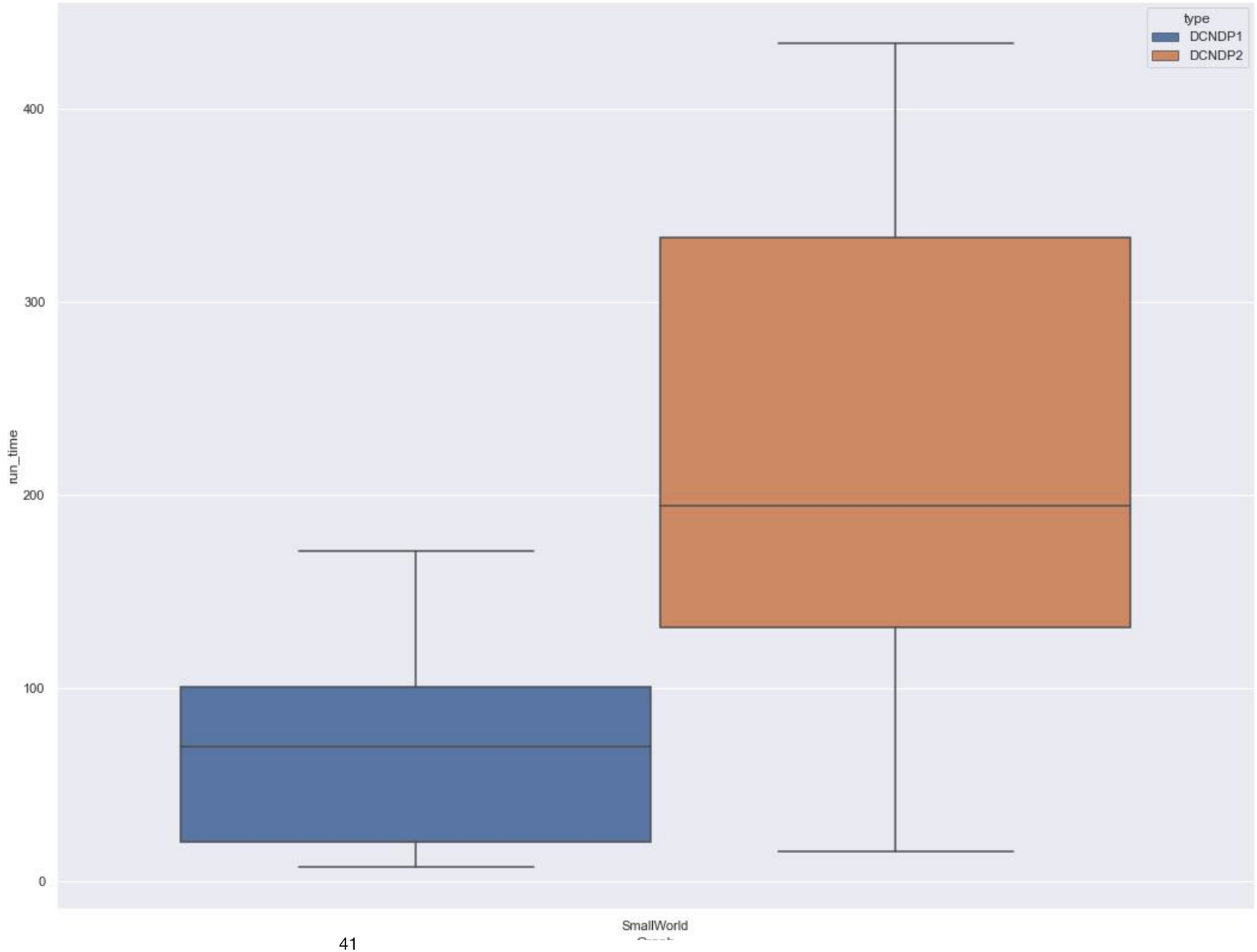
Big real world test.

%obj UsaAir97



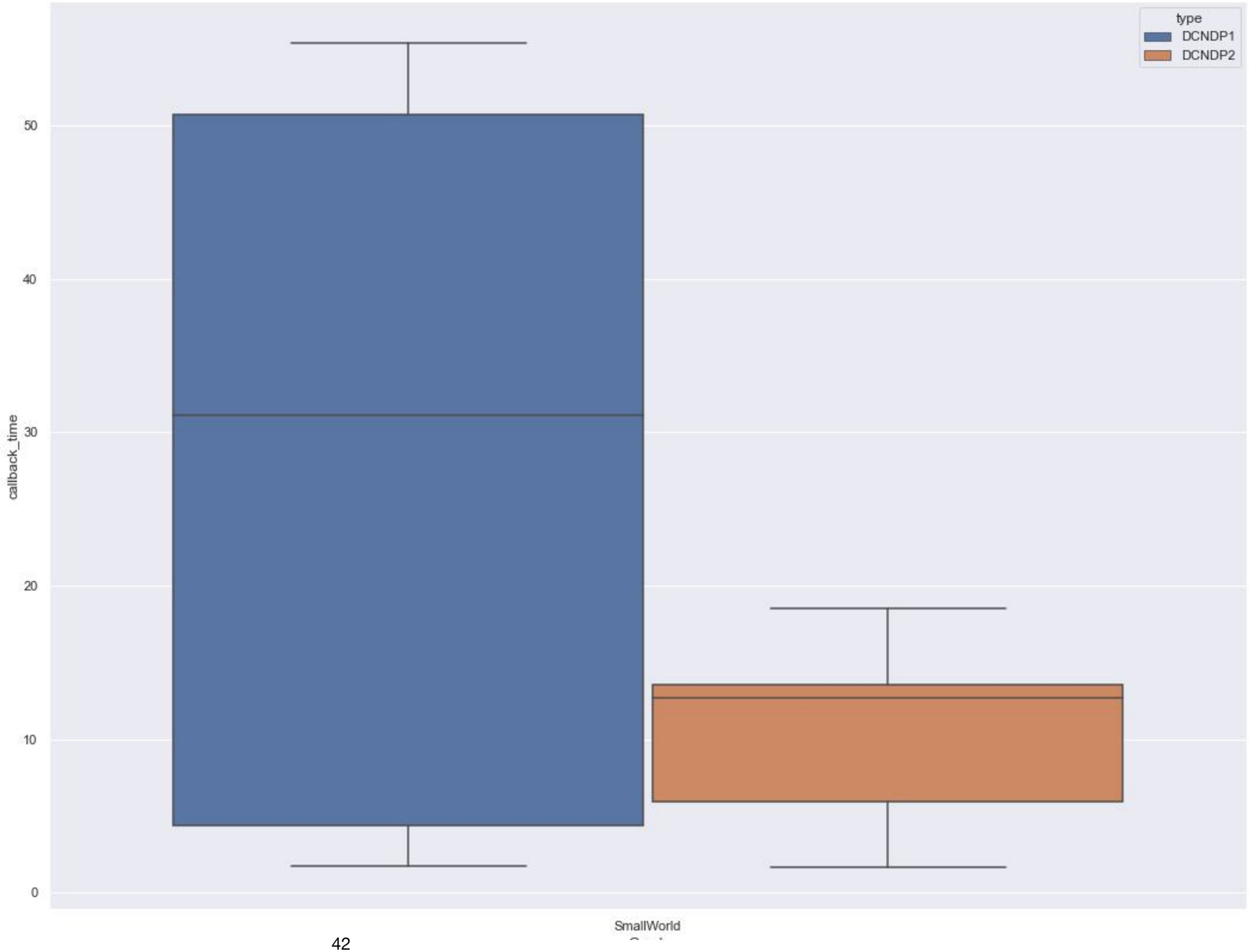
Experiments

Big real world test.
Runtime SmallWorld



Experiments

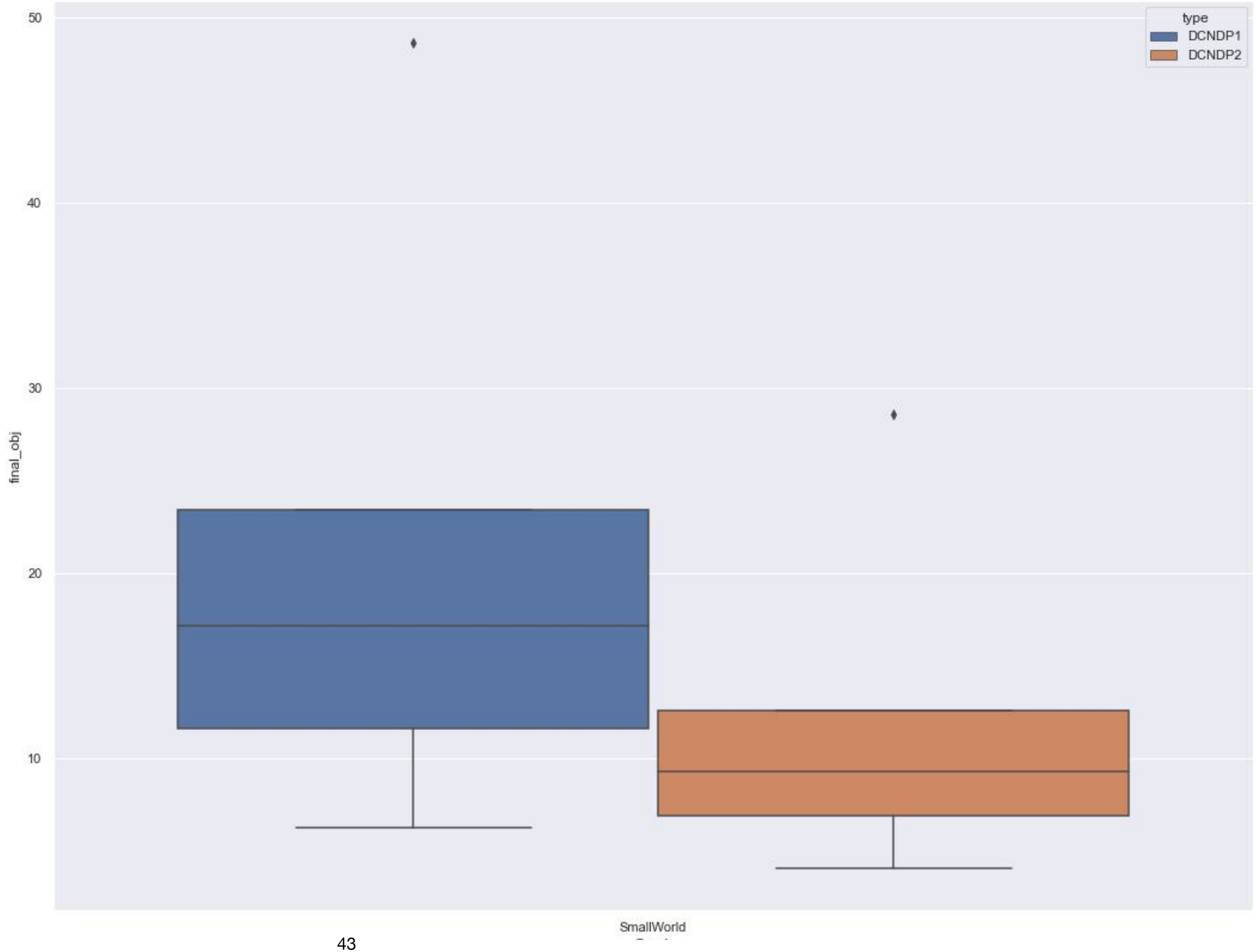
Big real world test.
Callback time
SmallWorld



Experiments

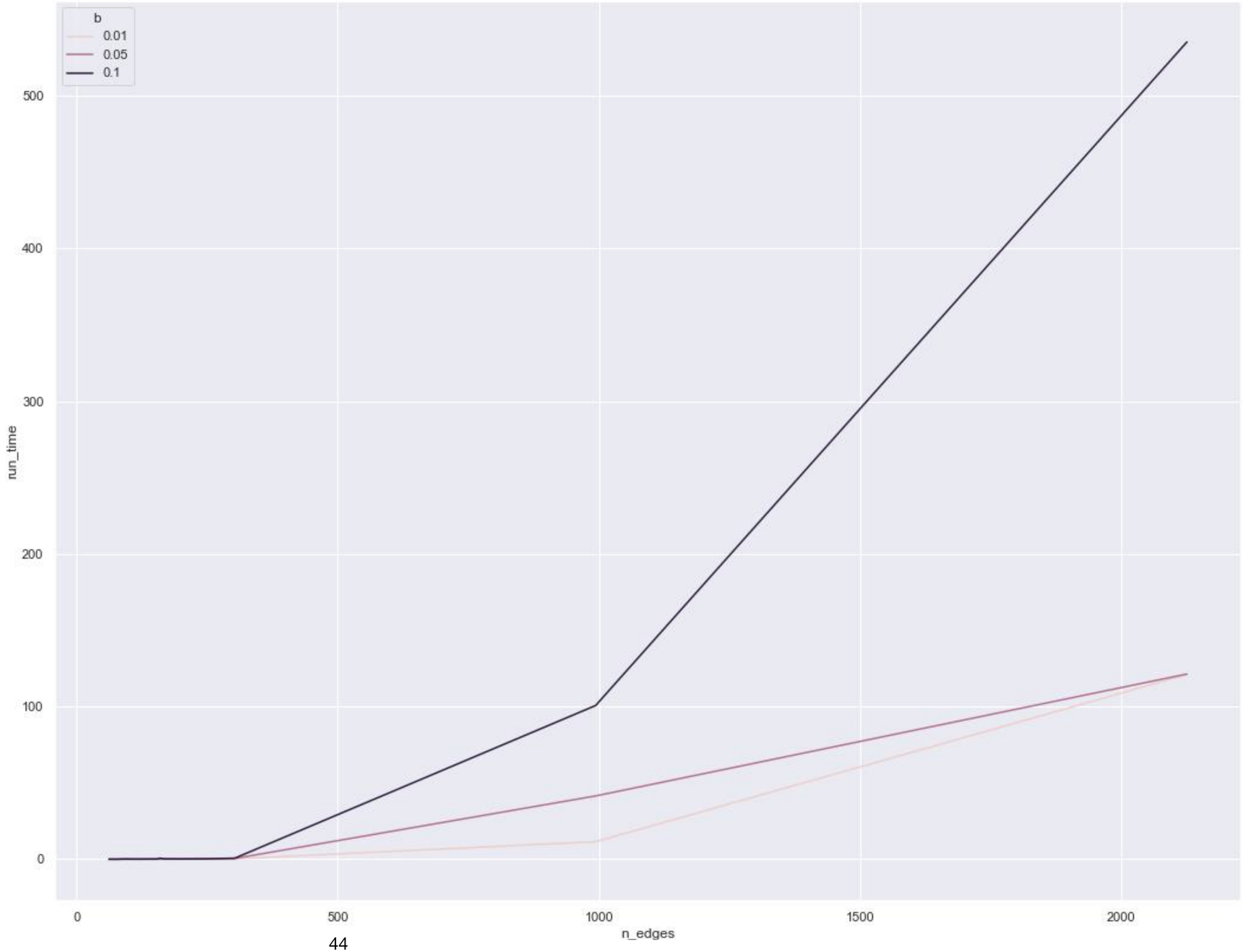
Big real world test.

%obj



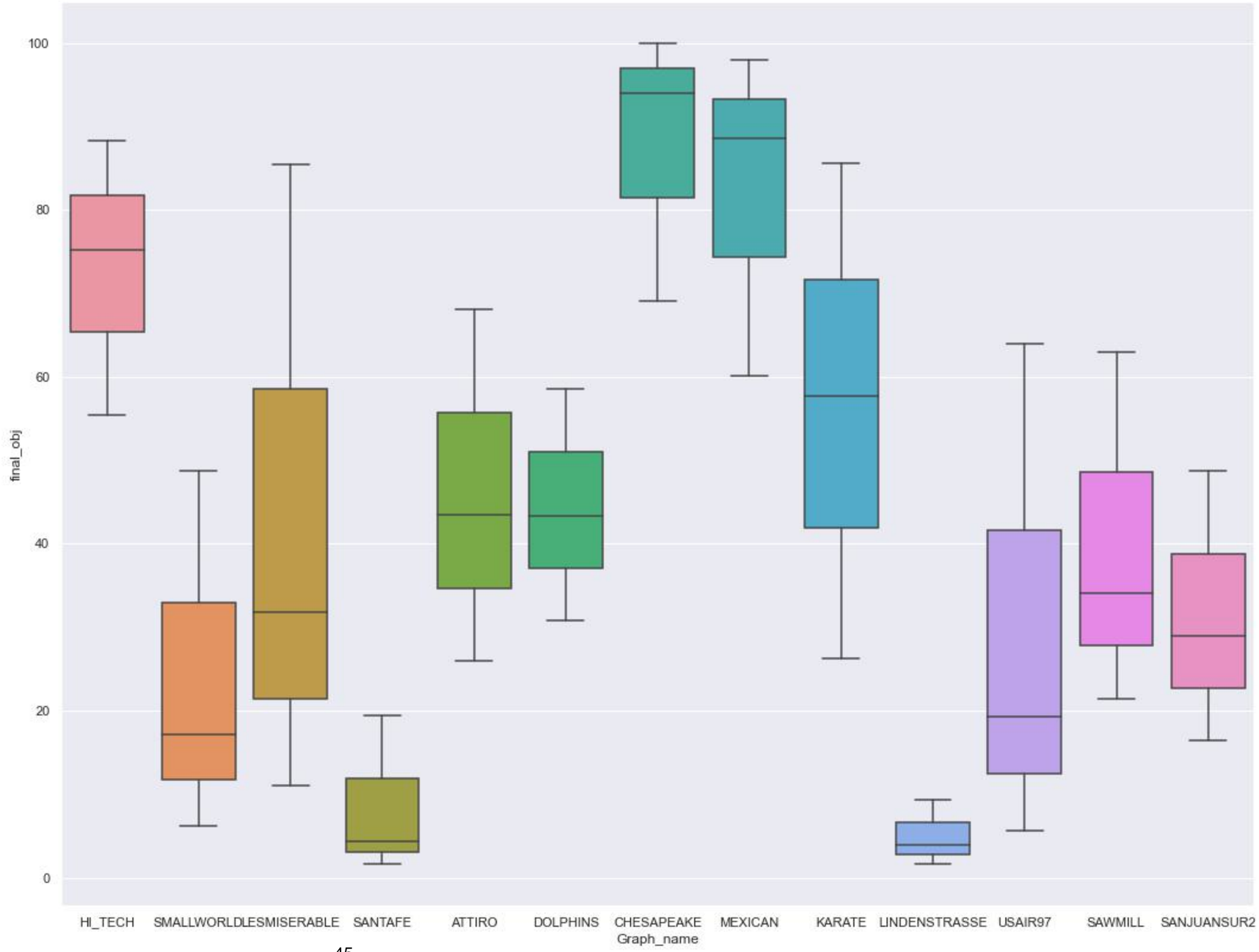
Experiments

DCNDP_1 on real graphs at various B



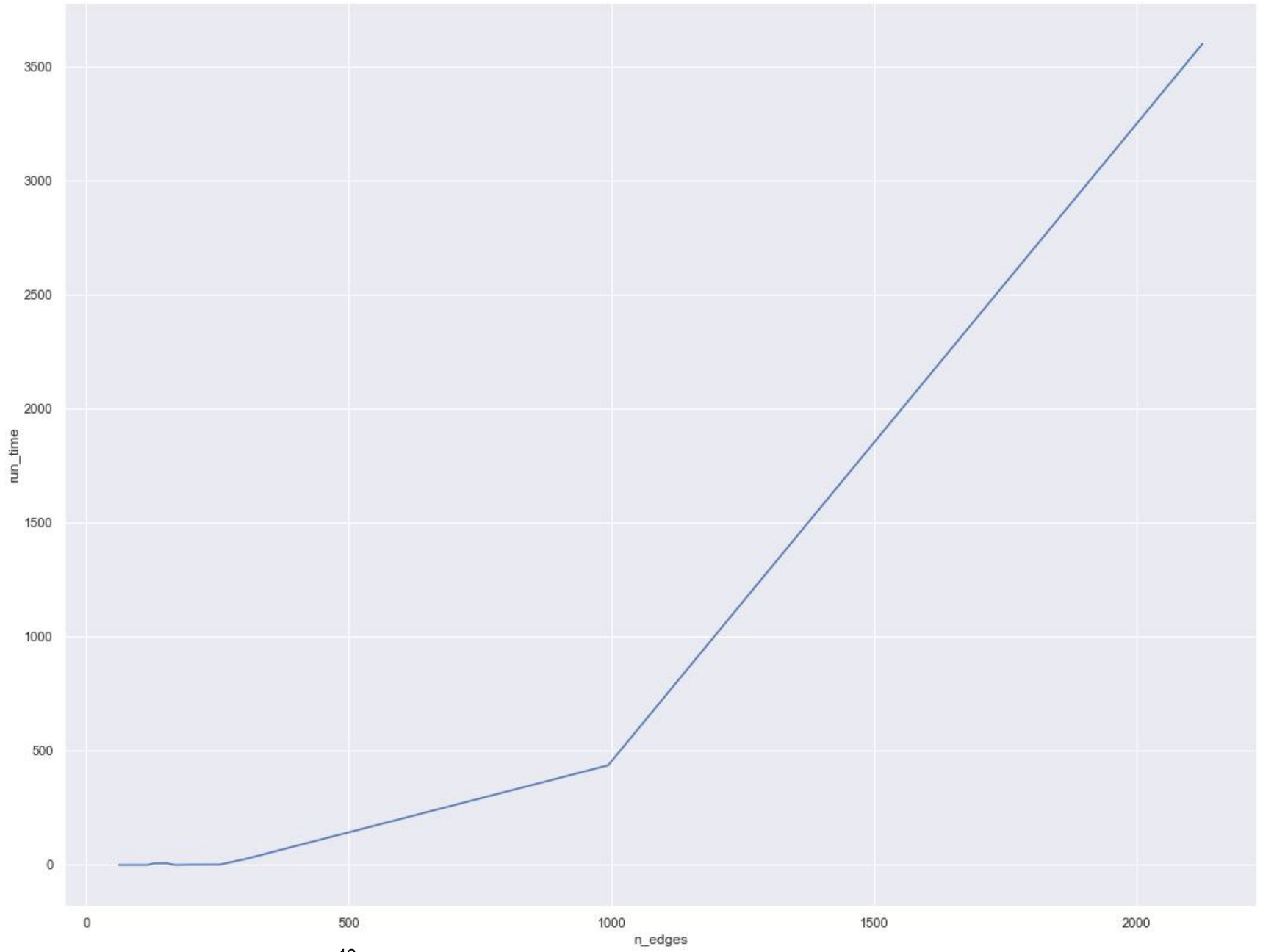
Experiments

DCNDP_1 on real graphs at various B Gap% finalObj



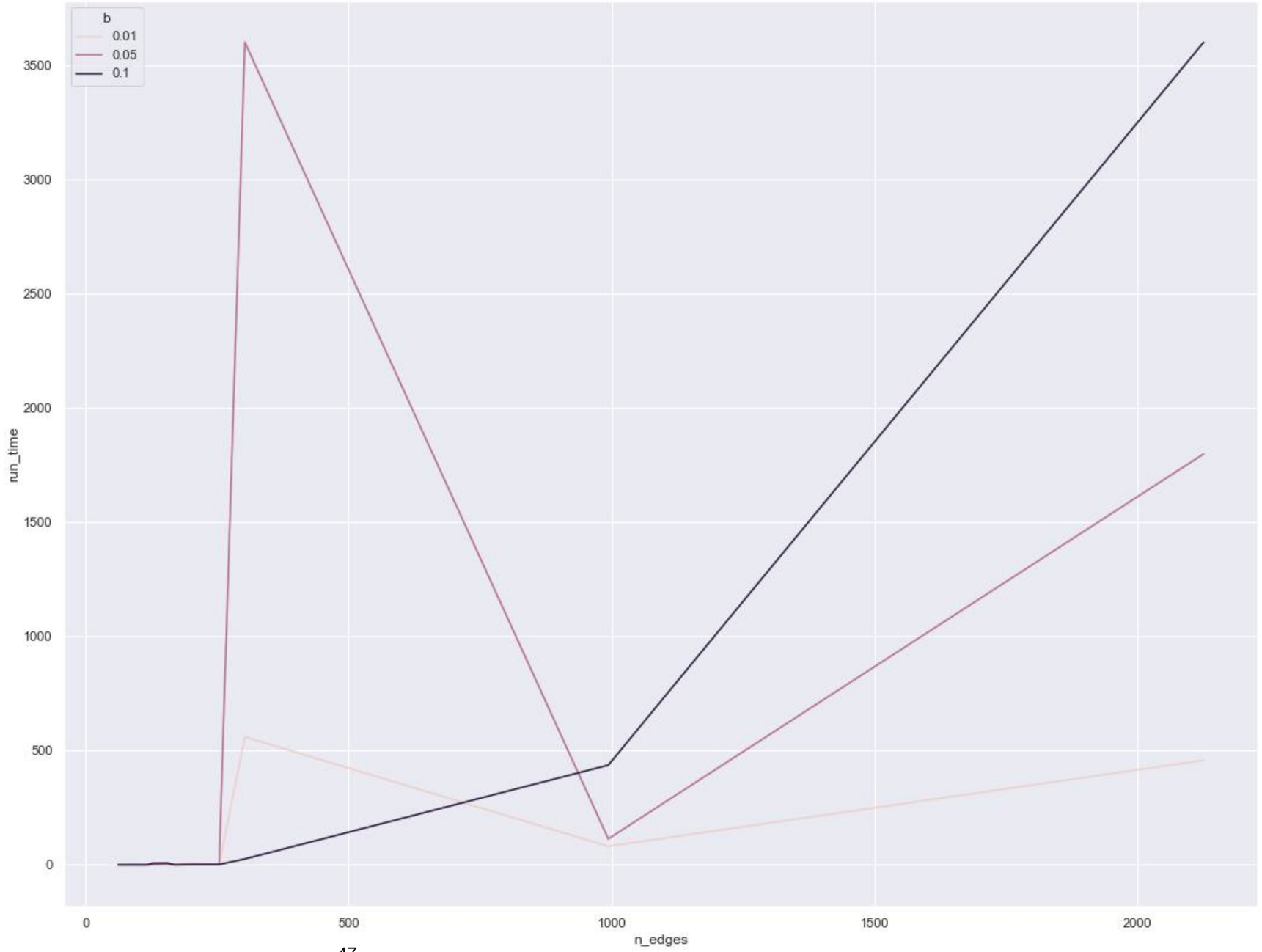
Experiments

DCNDP_2 on real graphs with B = 0.1N



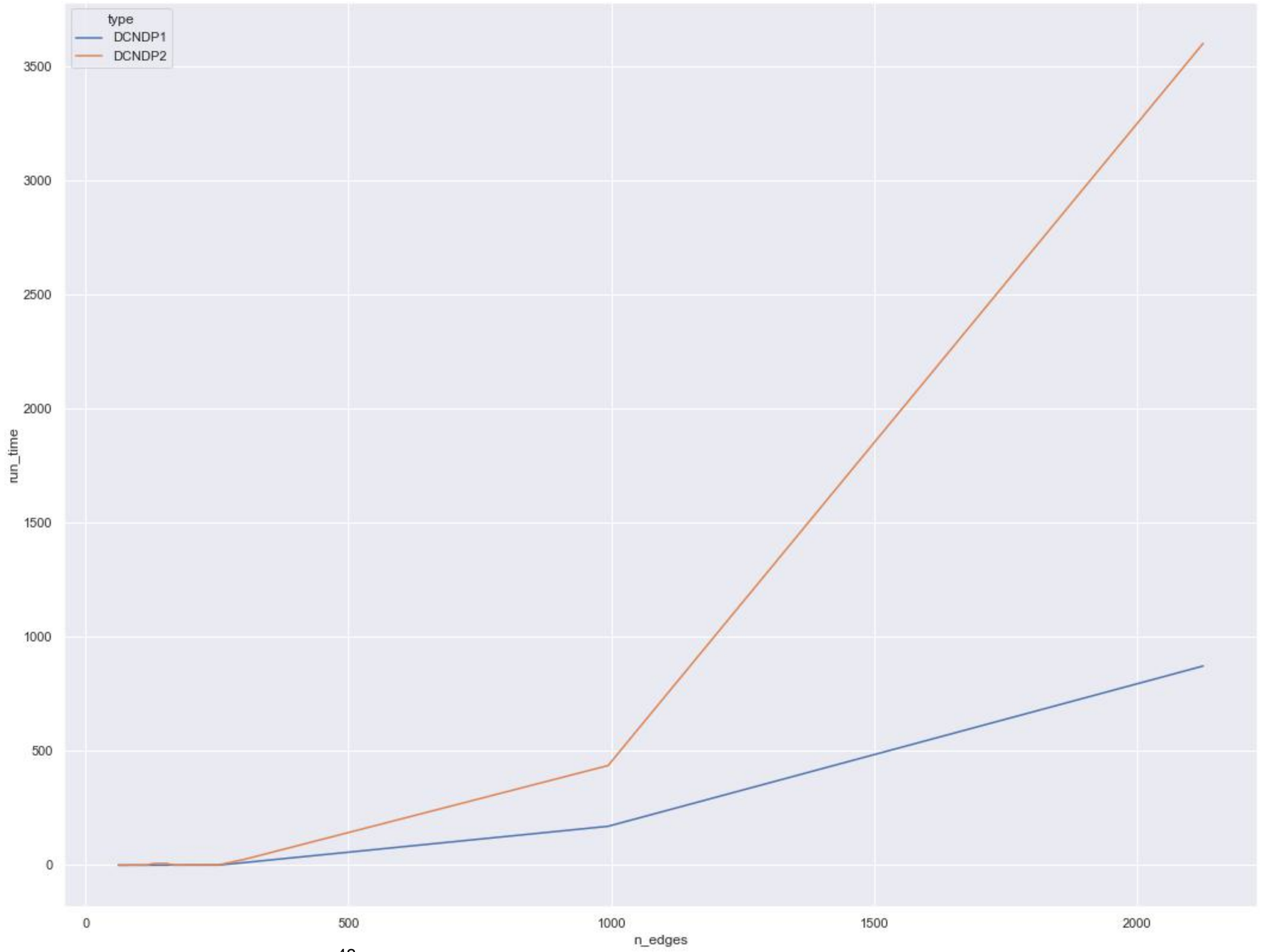
Experiments

DCNDP_2 on real graphs with various B



Experiments

DCNDP 1 vs 2 on real graphs with B = 0.1N



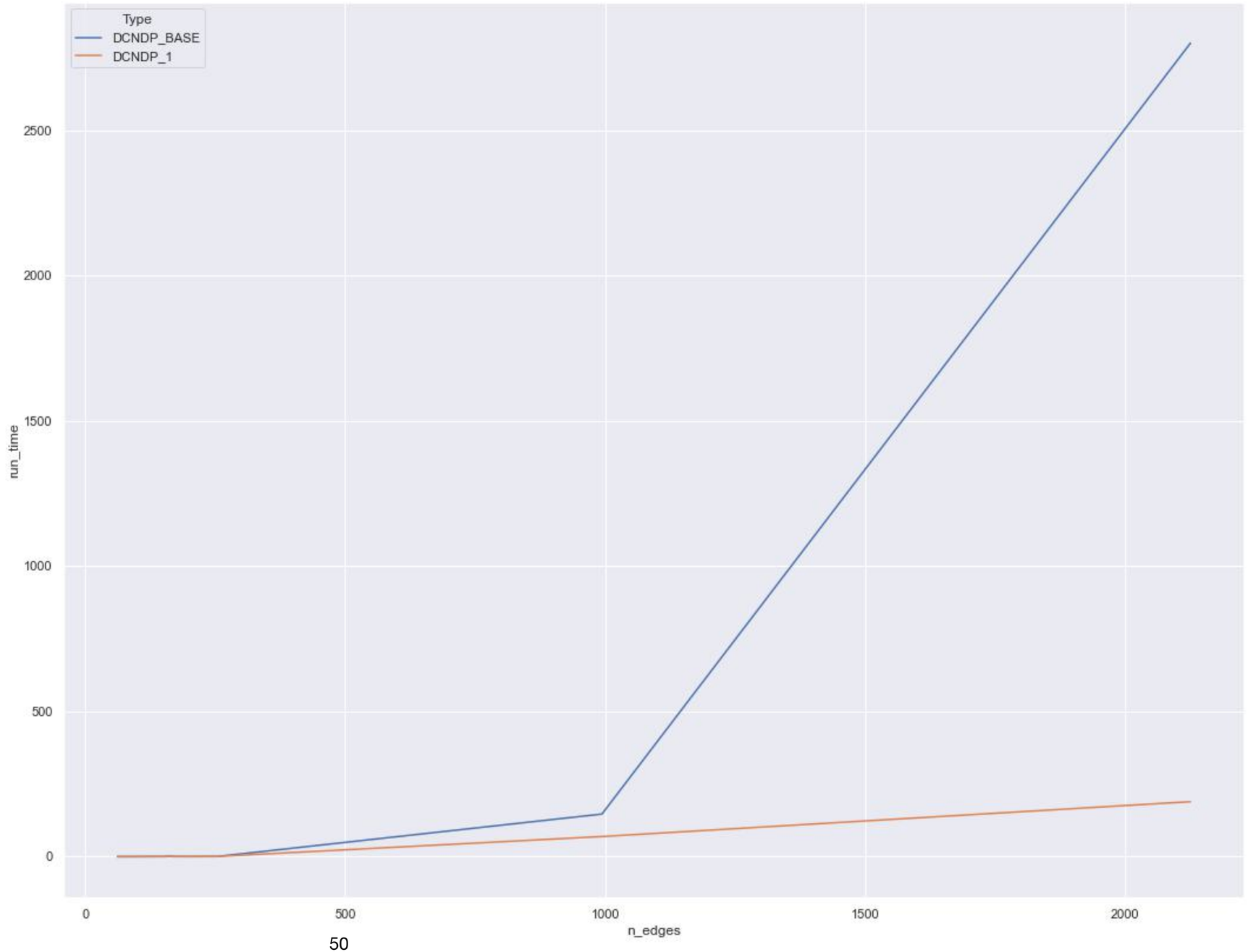
Experiments

DCNDP_BASE vs DCNDP_1 with B = 0.05N & 0.1N

				B = 0.05N		B = 0.1N	
				DCNDP BASE	DCNDP_1	DCNDP BASE	DCNDP_1
graph	n_nodes	n_edges	diameter	run_time (s)	run_time (s)	run_time (s)	run_time (s)
Hi-Tech	33	91	5	0,12	0.12	0.59	0,36
Karate	34	78	5	0,16	0.07	0.11	0,05
Mexican	35	117	4	0,31	0.16	0.33	0,16
Sawmill	36	62	8	0,08	0.07	0.08	0,04
Chesapeake	39	170	3	0,6	0,32	1.36	0,55
Dolphins	62	159	8	1,91	0,88	1.91	1,22
Lesmiserable	77	254	5	0,52	0,73	0.97	0,57
Santafe	118	200	12	0,2	0,69	0.59	0,43
Sanjuansur	75	155	7	0,44	0,24	0.39	0,42
Attiro	59	128	8	0,39	0.16	0.67	0,35
UsAir97	332	2126	6	2800,56	189,18	>3600	872,71
SmallWorld	233	994	4	146,52	69,2	>3600	170,52

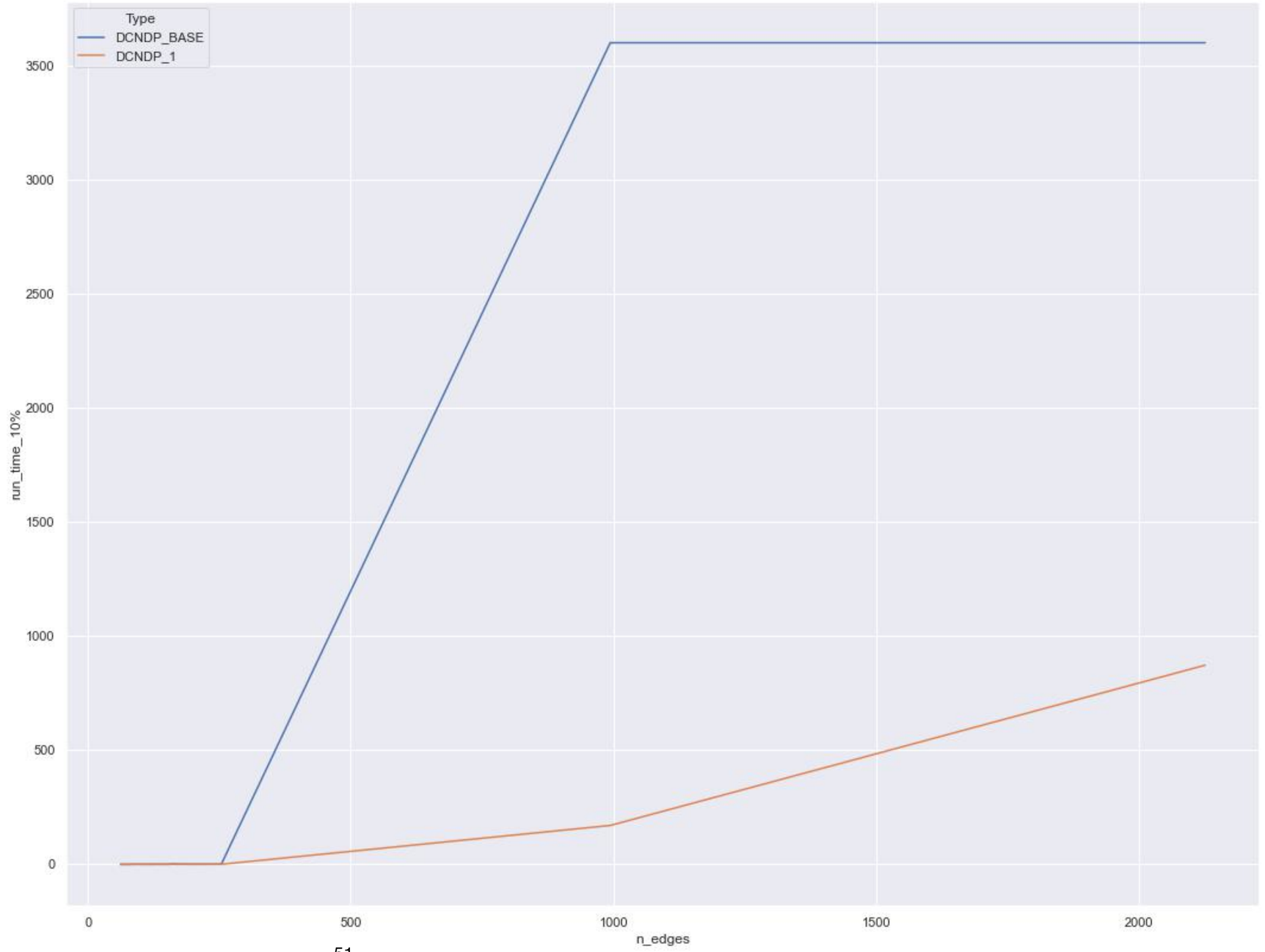
Experiments

DCNDP_BASE vs DCNDP_1 with B = 0.05N



Experiments

DCNDP_BASE vs DCNDP_1 with B = 0.1N



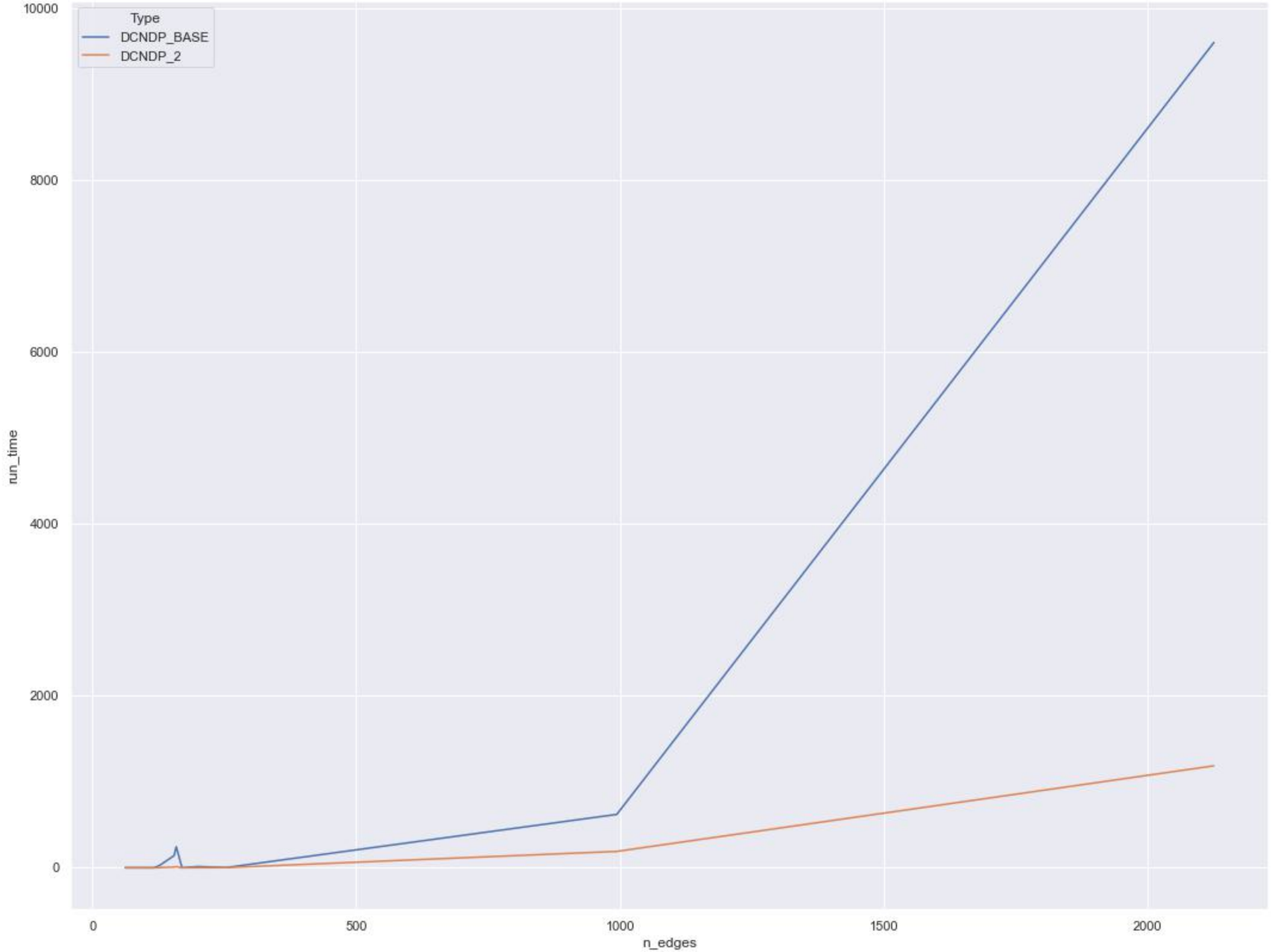
Experiments

DCNDP_BASE vs DCNDP_ with B = 0.05N & 0.1N

				B = 0.05N		B = 0.1N	
				DCNDP BASE	DCNDP_1	DCNDP BASE	DCNDP_1
graph	n_nodes	n_edges	diameter	run_time (s)	run_time (s)	run_time (s)	run_time (s)
Hi-Tech	33	91	5	0,47	0,36	2,38	0,4
Karate	34	78	5	0,31	0,11	0,38	0,16
Mexican	35	117	4	0,36	0,27	0,92	0,21
Sawmill	36	62	8	0,75	0,23	0,49	0,09
Chesapeake	39	170	3	0,27	0,25	0,33	0,25
Dolphins	62	159	8	245	12,46	602,5	1,38
Lesmiserable	77	254	5	3,03	1,97	7,27	1,97
Santafe	118	200	12	14,1	1,24	21,39	1,24
Sanjuansur	75	155	7	142	8,18	216,9	7,9
Attiro	59	128	8	29	2,29	153,78	7,18
UsAir97	332	2126	6	M	1186	M	2100
SmallWorld	233	994	4	620,7	190,4	>3600	730

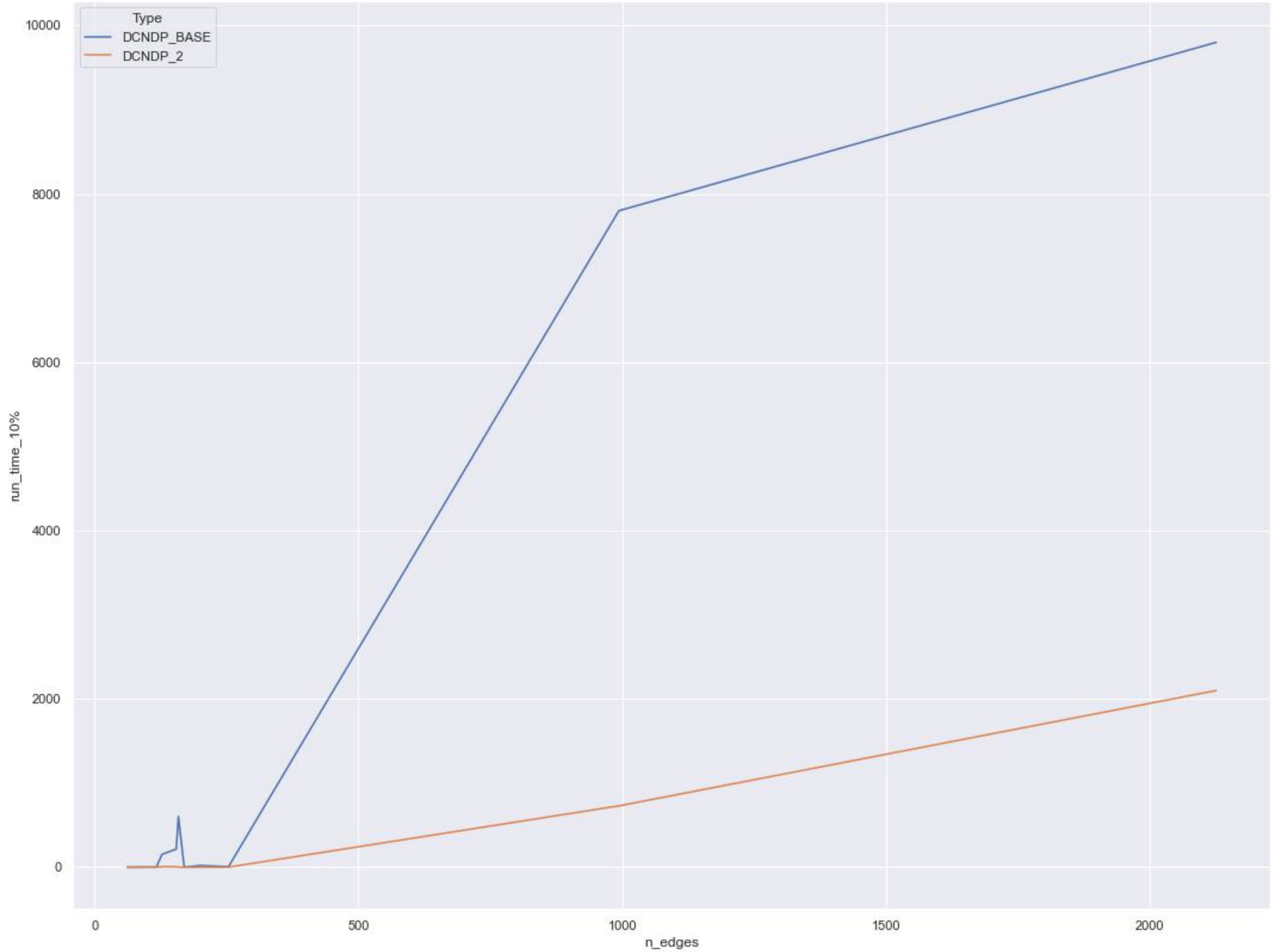
Experiments

DCNDP_BASE vs DCNDP_2 with B = 0.05N



Experiments

DCNDP_BASE vs DCNDP_2 with B = 0.1N



DCNDP for weighted graph :

Problem description and formulation

DCNDP_WEIGHTED

Problem definition

- Let $G = (V, E)$ be a given undirected weighted graph with a finite set V of nodes.
- For every $(\mathrm{i}, \mathrm{j}) \in \mathrm{E}$, we denote by w_{ij} a positive weight on (i, j) interpreted as the length of (i, j)
- For any pair of nodes i and j, the length of any given path between i and j is the sum of all edge weights along that path
- We assume that all edge weights w_{ij} are positive integers which is not too restrictive for most real-world applications

.

DCNDP_WEIGHTED

Formulations

- 5.5-5.10 are based on the same idea as those of constraints 3.21-3.26
- 5.3-5.4 enchance the formulations in a similar version as the shortest path-based enhancements described for the base formulation
- 5.5 instead of the hop-based lengt of path P , we now consider the edge-weighted length of path P represented by d

$$\text{minimise} \quad \sum_{i,j \in V: i<j} \left(f(1) y_{ij}^1 + \sum_{\ell=2}^{L} f(\ell) \left(y_{ij}^{\ell} - y_{ij}^{\ell-1} \right) \right) \tag{5.1}$$

$$\text{s.t.} \quad y_{ij}^{\ell} = 0, \quad \forall (i,j) \in E,\ i<j,\ \ell \in \left\{ 1, \ldots, \min_{t \neq j:(i,t) \in E} \{ w_{it}, w_{ij} - 1 \} \right\} \tag{5.3}$$

$$y_{ij}^{\ell} = 0, \quad \forall (i,j) \notin E,\ i<j,\ \ell \in \left\{ 1, \ldots, \min_{t:(i,t) \in E} \{ w_{it} \} \right\} \tag{5.4}$$

$$\sum_{r \in V(P)} x_r + y_{ij}^{d} \geq 1, \quad \forall P \in \mathcal{P}_L(i,j),\ i,j \in V,\ i<j \tag{5.5}$$

$$y_{ij}^{\ell} \leq y_{ij}^{\ell+1}, \quad \forall (i,j) \notin E,\ i<j,\ \ell \in \{2, \ldots, L-1\} \tag{5.6}$$

$$y_{ij}^{\ell} = y_{ij}^{(w_{ij})}, \quad \forall (i,j) \in E,\ i<j,\ \ell \in \{w_{ij}+1, \ldots, L\} \tag{5.7}$$

$$\sum_{i \in V} x_i \leq B \tag{5.8}$$

$$y_{ij}^{\ell} \in \{0,1\}, \quad \forall (i,j) \in V,\ i<j,\ \ell \in \{1, \ldots, L\} \tag{5.9}$$

$$x_i \in \{0,1\}, \quad \forall i \in V \tag{5.10}$$

DCNDP_WEIGHTED

New Compact Formulations

- 5.11 minimises the connectivity of the input graph with respect to a specified distance-based connectivity function $\mathrm{f}(\,.\,) \geq \mathrm{d}$
- 5.12-5.15 admits any non-negative non increasing distance function such as the first 3 classes of the DCNDP

$$\text{minimise} \quad \sum_{i,j \in V : i<j} y_{ij} \tag{5.11}$$

$$\text{s.t.} \quad \sum_{r \in V(P)} f(d) x_r + y_{ij} \geq f(d), \quad \forall P \in \mathcal{P}_L(i,j),\ (i,j) \in V,\ i<j \tag{5.12}$$

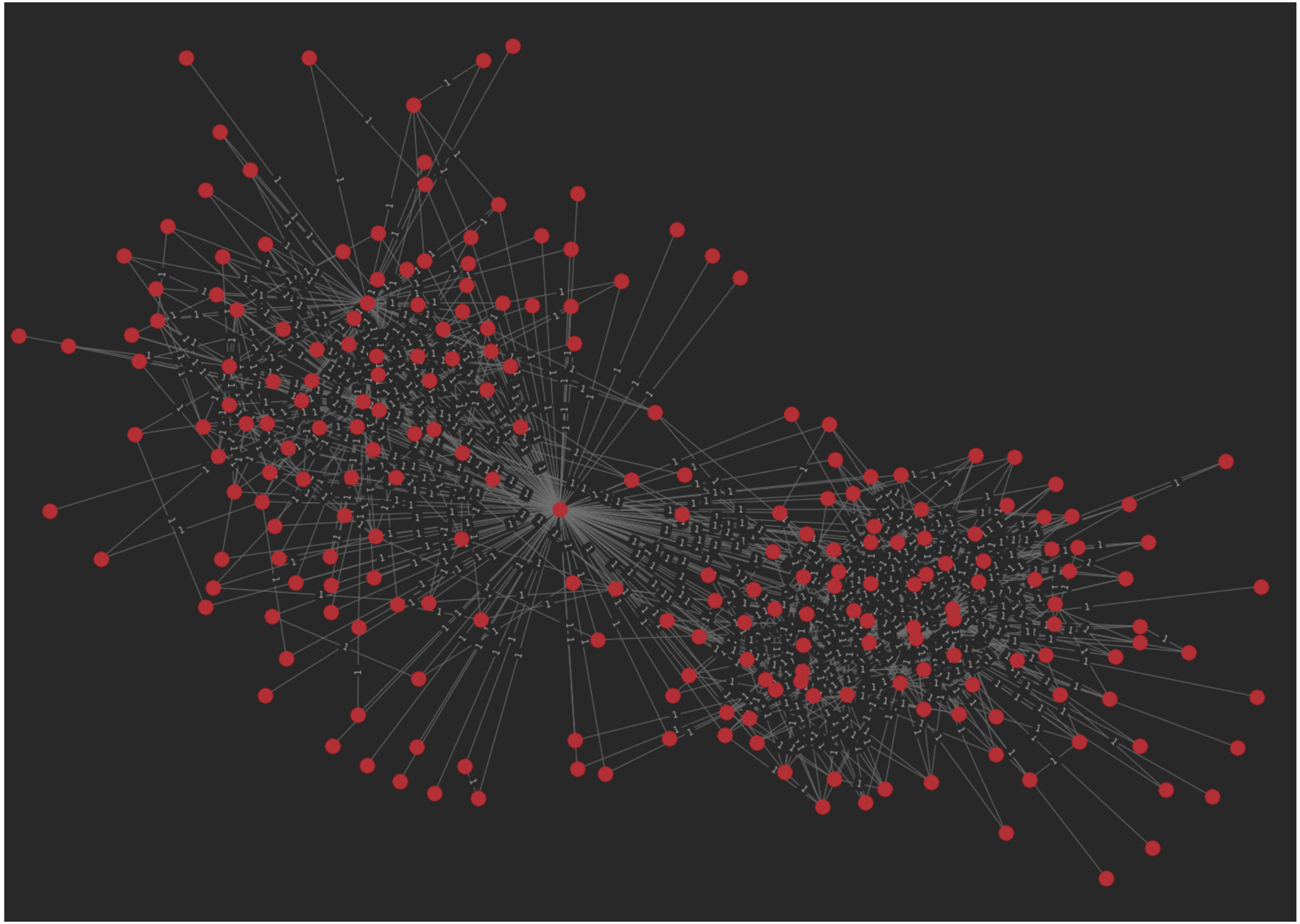
$$\sum_{i \in V} x_i \leq B \tag{5.13}$$

$$x_i \in \{0,1\}, \quad \forall i \in V \tag{5.14}$$

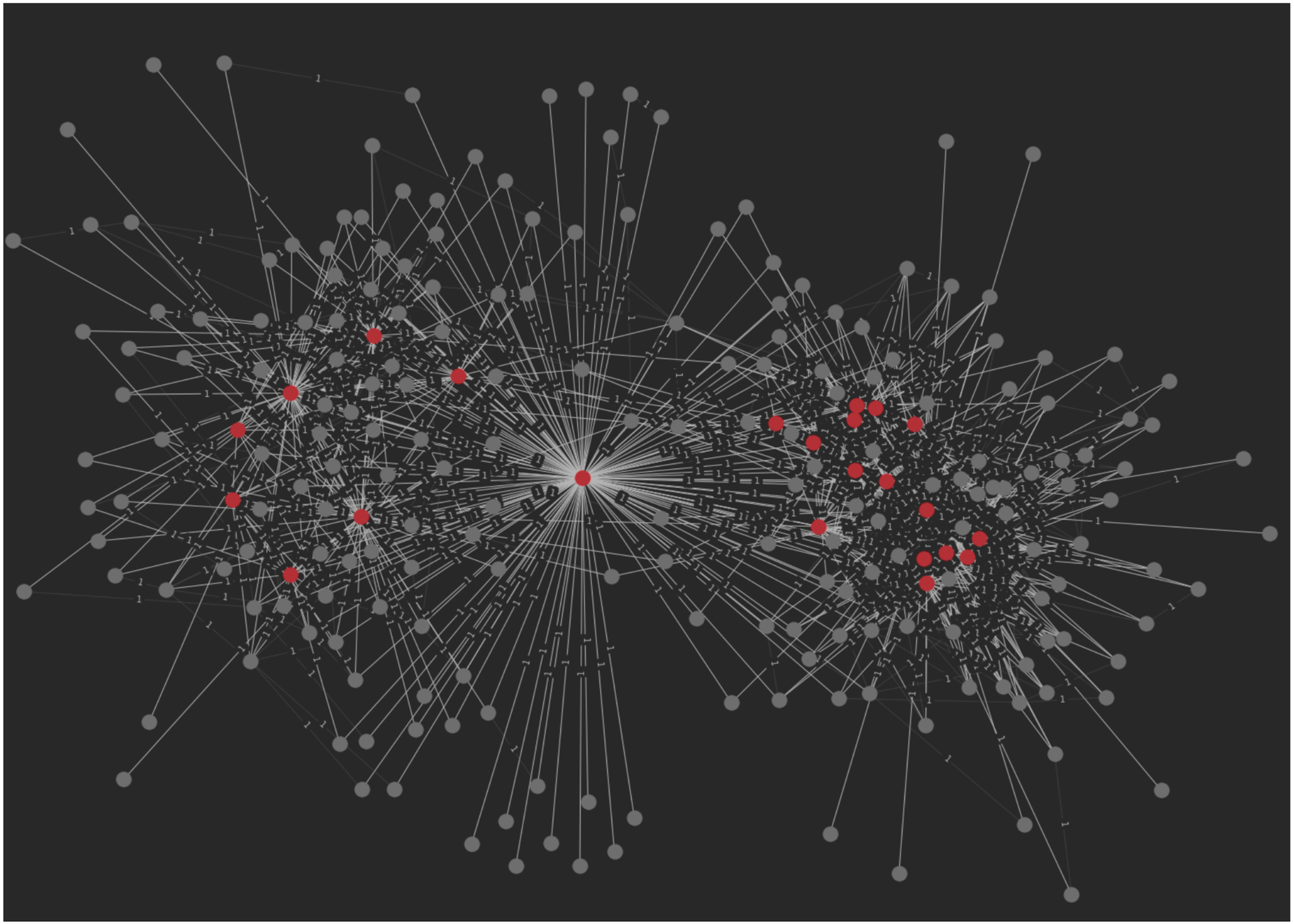
$$y_{ij} \geq 0, \quad \forall (i,j) \in V,\ i<j \tag{5.15}$$

DCNDP WEIGHTED

- DCNDP example with UpperBound B = 6
- Graph : SmallWorld
- First image is the original graph
- Second image is graph without critical nodes after optimization

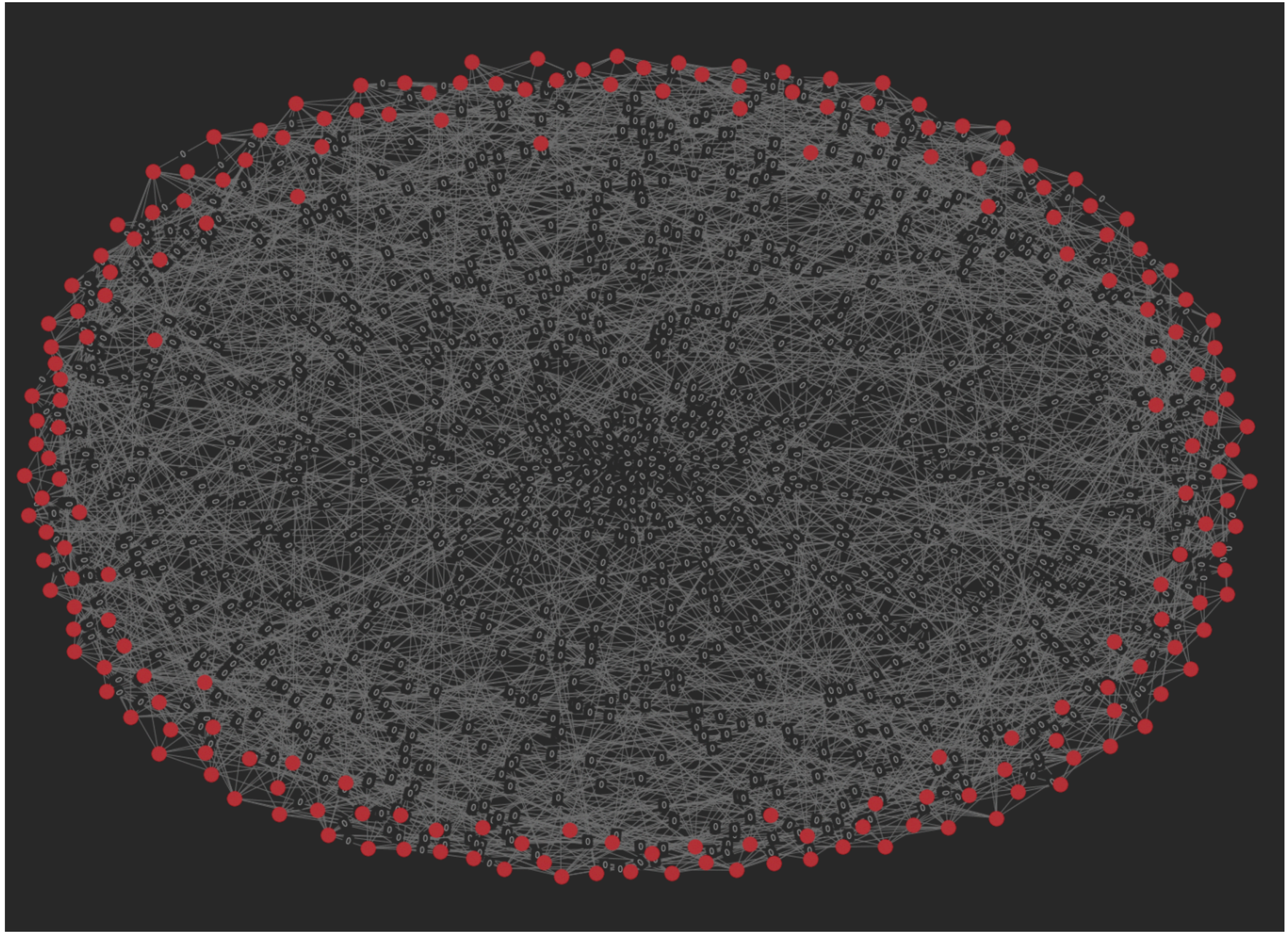


DCNDP WEIGHTED

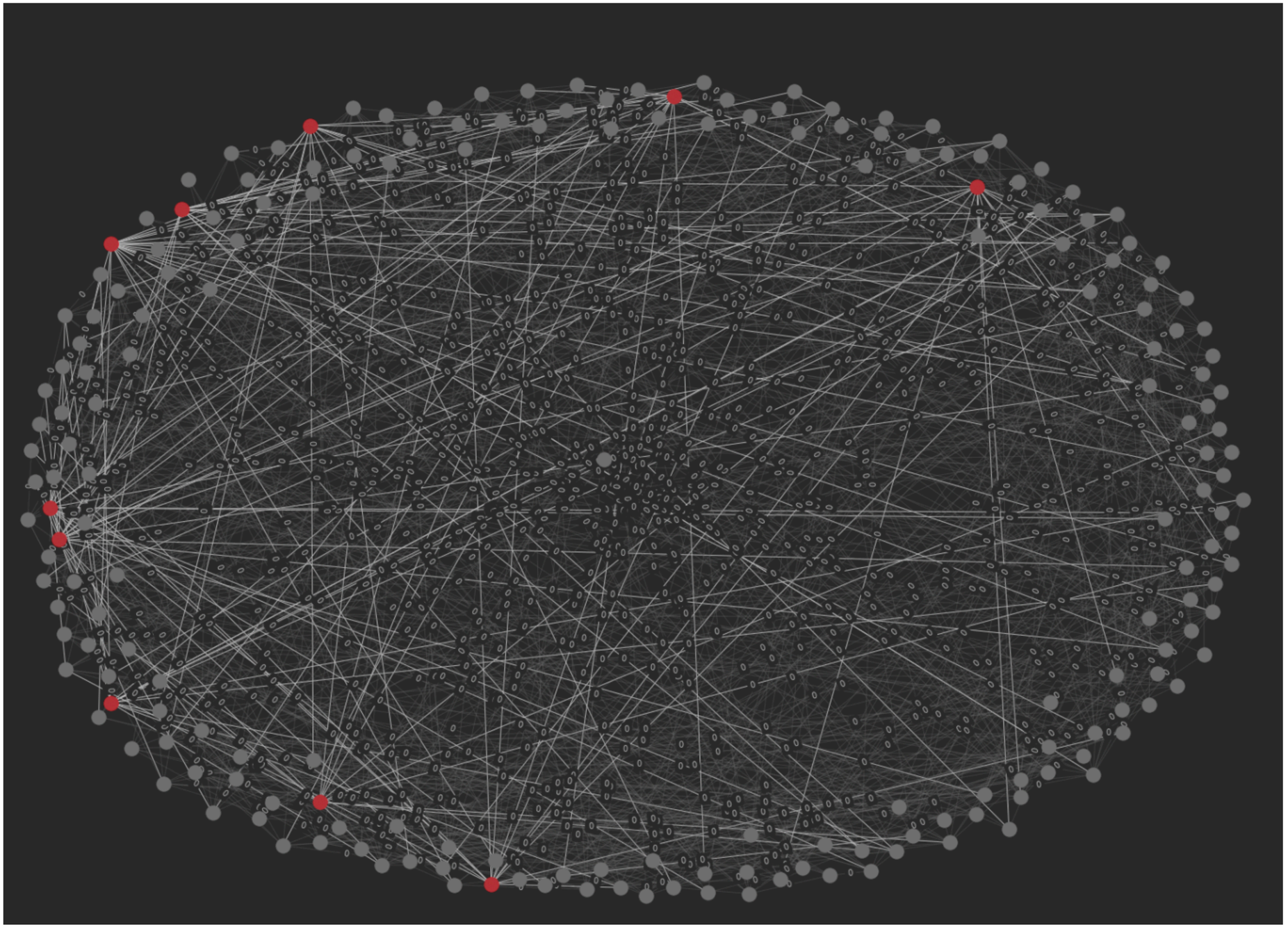


DCNDP WEIGHTED

- DCNDP example with UpperBound B = 6
- Graph : ER2
- First image is the original graph
- Second image is graph without critical nodes after optimization



DCNDP WEIGHTED



Computational experiments

Experiments

DCNDP_WEIGHTED B=0.1N

Graph	n_nodes	n_edges	diameter	%final_obj	n_vars_sol	distance-based pairwise connectivity	Status	callback_time (s)	run_time (s)
Santafe	118	200	12	1,84	7021	127,3	2	0,46	0,49
Sanjuansur	75	155	7	14,51	2850	402,7	2	1,16	5,31
SmallWorld	233	994	4	6,03	272261	1630	2	12,21	202,95
BA2	100	900	3	32,8	5050	1627	2	11,93	24,78
ER2	200	1032	4	21,51	20100	4281	2	72,42	2032

Thank You