

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.

”

Applicazione del critical node detection problem

Computational Biology

- Il critical node detection problem è stato applicato per studiare le interazione fra le proteine nei network biologici
- Organismi biologici, quali batteri, sono composti da sets di proteine interconnesse cui interazioni formano un network di interazioni proteina-proteina dove i nodi sono rappresentati dalle proteine e gli archi dalle interazioni
- Nel contesto di tali interazioni, i nodi critici sono rappresentati dalle proteine imprescindibili per la connettività del network
- Boginski & Commander (2009) utilizzarono una versione modificata del CNP (il CC-CNP) per identificare le proteine cui rimozione avrebbe distrutto l'interazione primaria portando alla neutralizzazione di organismi quali batteri
- Gli autori proposero inoltre che l'applicazione del CNDP in questo tipo di network sia potenzialmente utile nel drug design

Controllo dei contagi

- Il Critical Node Detection trova applicazione nel contagion control
- Tagliare o limitare le connessioni fra nodi di una network si rivela necessario per contenere l'incremento di eventi indesiderati
- Il classico CNDP è stato proposto come strategia molto efficiente per l'immunizzazione di una popolazione contro malattie
- Considerata la difficoltà di vaccinare un intera popolazione, immunizzare o isolare i nodi critici identificati fornisce una via poco costosa ed effettiva per dimezzare o eliminare la propagazione dell'infezione

Problem description and base formulation

DCNDP

Problem definition

- Sia $G = (V, E)$ un semplice grafo non pesato e indiretto con un set finito V di nodi e un set finito $E \subseteq V \times V$ di archi dove $n := |V|$ and $m := |E|$
- Per un qualsiasi nodo i , denotiamo con $\mathcal{N}_G(i)$ il set dei neighbor di i
- Il nodo i si dice di essere connesso ad un nodo j se esiste un path fra essi. Uno shortest path fra i e j è un path contenente il minor numero di archi fra tutti i paths che collegano i and j .
- Il numero di archi in uno shortest path fra i e j in G è noto come la hop-distance
- La massima distanza fra ogni coppia di nodi è conosciuta come diametro di G
- Il goal del DCNDP è quello di trovare un subset di nodi $R \subset V$ con $|R| \leq B$, cui rimozione minimizzi una determinata funzione di distance-based connectivity $f(d)$

DCNDP

Classi di distance-based critical node detection problem

Class 1. Minimizzare il numero di nodi connessi da un path di lunghezza al più k :

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

dove k è un intero positivo rappresentante la distanza di cut-off. Tale problema assumerà il nome di DCNDP_1

Class 2. Minimizzare Harary index o efficienza di un dato grafo:

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

Questa metrica si basa sull'assunzione che in un network di comunicazione la communication efficiency fra nodi è inversamente proporzionale alla distanza fra loro. Tale problema assumere la denominazione di DCNDP_2

DCNDP

Premessa

- Associato con ogni nodo $i \in V$, definire una variabile x_i cosicché :

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

- Connectivity variable:

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

New integer programming formulations

DCNDP_2

New Integer Programming Formulations

- Considerato che solo path fino a lunghezza L hanno importanza in questo caso, tenendo traccia della lunghezza dei path fino a lunghezza L , si garantisce che i nodi (i, j) sono disconnessi ad una certa distanza L se e solo se almeno uno dei nodi fra i path presi come candidates fra i e j viene eliminato.
- Tale idea è stata utilizzata all'interno del paper per proporre un nuovo modello basato sulla connessione dei cammini fra nodi e come tale potrebbe comportare una crescita esponenziale dei vincoli.
- Tuttavia, solo alcuni di questi vincoli sarebbero attivi nella formulazione, quindi possono essere trattati come vincoli lazy da separare in tempo polinomiale risolvendo uno shortest path problem
- Seguendo quanto proposto da Veremyev, il vincolo di integralità della variabile di connettività può essere rilassato.

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

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

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

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

$$\sum_{i \in V} x_i \leq B \quad (3.24)$$

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

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

DCNDP_1

Formulation and relaxation

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

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

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

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

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

(3.28)

- Per il problema DCNDP_1, la funzione obiettivo non fa un uso esplicito della lunghezza dei path indicizzata da l nella variable di connettività

(3.29)

- Questo ha permesso di eliminare tale indice nella variabile di connettività $y_{i,j}^l$ usata nel modello proposto precedentemente

(3.30)

- Soluzioni ottenute più efficienti computazionalmente

(3.31)

- Introduzione di una nuova variabile di connettività:

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

DCNDP_2

Spiegazione & Implementazione : Funzione Obiettivo

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

```
# objective
obj = LinExpr(0)
for j in input_graph.nodes():
    for i in input_graph.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]))
```

- La funzione obiettivo minimizza una certa funzione di distance-based connectivity $f(\cdot)$

DCNDP_2

Spiegazione & Implementazione : vincolo 3.21

$$st. \quad \sum_{r \in V(P)} x_r + y_{ij}^{|P|} \geq 1, \quad \forall P \in P_l(i, j), \quad (i, j) \in V, \quad i < j$$

- Il vincolo (3.21) garantisce che la coppia di nodi (i, j) siano *L – distance disconnected* se e solo se almeno uno dei nodi fra tutti i paths di lunghezza minore o uguale ad L che collega i nodi i e j , sia eliminato.
- Considerato che potenzialmente il modello potrebbe essere composto da molteplici vincoli, alcuni dei quali ridondanti, viene modellato esplicitamente il vincolo di non ridondanza $y_{ij}^l + x_i + x_j \geq 1$

DCNDP_2

Spiegazione & Implementazione : vincolo 3.22

$$y_{ij}^{l-1} \leq y_{ij}^l, \quad \forall (i,j) \in V, \quad i < j, \quad l \in [2\dots L]$$

```
for j in input_graph.nodes():
    for i in input_graph.nodes():
        if i not in input_graph.neighbors(j) and i < j:
            for l in range(1, L):
                model.addConstr(u_connect[i, j, l] <= u_connect[i, j, l + 1], name="3.22")
```

- Il vincolo (3.22) garantisce che esiste un path di lunghezza l , se esiste un path di lunghezza $l - 1$

DCNDP_2

Spiegazione & Implementazione : vincolo 3.23

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

```
for (i, j) in input_graph.edges():
    if i < j:
        for l in range(2, L + 1):
            model.addConstr(u_connect[i, j, 1] == u_connect[i, j, l], name="3.23_2")
    else: # that is j < i
        for l in range(2, L + 1):
            model.addConstr(u_connect[j, i, 1] == u_connect[j, i, l], name="3.23_2")
```

- Il vincolo (3.23) garantisce che $y_{ij}^l = 1$ per tutte le coppie di nodi adiacenti (i,j) , se nessuno dei nodi i o j è eliminato

DCNDP_2

Spiegazione & Implementazione : vincolo 3.24

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

```
# constraint on number of critical nodes
model.addConstr(sum((x_delete[j]) for j in input_graph.nodes()) <= C, name="3.24")
```

Il vincolo (3.24) rappresenta il vincolo di budget che va a limitare la cardinalità del set di nodi critici

DCNDP_1

Spiegazione & Implementazione

$$\text{Min.} \sum_{i,j \in V: i < j} y_{ij}$$

```
# objective
obj = LinExpr(0)
for j in input_graph.nodes():
    for i in input_graph.nodes():
        if i < j:
            obj.add(u_connect[i, j])
```

La funzione obiettivo (3.28) minimizza le coppie di nodi collegati da path composti da al più k hop-distance

Solution Methods

DCNDP

Solution Methods

- Uso di una custom routine per separare i lazy cuts violati
- Tale approccio consiste nella generazione di breadth-first-search tree per ogni nodo $v \in V$
- Il percorso univoco creato dall'albero BFS fino al nodo di root fornisce il vincolo esposto nel modello proposto prima (3.21 o 3.29)
- Per il DCNDP_1, considerato che siamo interessati ai paths di lunghezza non superiore a k , fermiamo l'attraversamento del tree alla profondità ricercata
- Per DCNDP_2, continuiamo l'attraversamento fino a profondità L . All'interno del BFS, identifichiamo i path dal nodo root r fino ai nodi a livello $l = [1...L]$ e se tale path è volato, lo aggiungiamo come taglio
- La generazione di tali BFS trees risulta essere molto più efficiente dell'approccio tradizionale
- BFS tree risulta essere generato con un costo $O(|E|)$ mentre risolvere il problema tradizionalmente avrà un costo pari a $O(|E| + |V| \log |V|)$

BFS

- L'algoritmo inizia selezionando un nodo r dal set di nodi candidate roots e costruisce il BFS fino alla profondità desiderata.
- L'algoritmo segue la struttura standard del BFS, con l'unica differenza di tener traccia della profondità di ogni nodo scoperto (linea 6 & 10)
- Per ogni nodo t che viene visitato, l'algoritmo procede in 1 delle 3 situazioni possibili, dipendendo dal livello $l[t]$ in cui si trova:
 - $l[t] = 1$ implica che il nodo t è un direct neighbour del nodo radice r , ovvero $(r, t) \in E$.
 - $1 < l[t] \leq L$, implica che $(r, t) \notin E$ ma la hop distance fra il nodo radice r e t è minore di o uguale ad L . Questo rappresenta il caso di interesse. Quindi non solo aggiungiamo t alla coda Q e marchiamo il nodo come visitato, ma verificheremo anche se le diseguaglianze (3.21 or 3.29) del corrispondente shortest path P_{rt} sono violate (lines 14-17) e nell'eventualità che lo siano, aggiungiamo le stesse alla coda Q_c .
 - $l[t] > L$, implica che tutti i nodi raggiungibili con hop distance L dal nodo root r , siano stati esplorati

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      $s \leftarrow Q.\text{remove}$ ;                  // retrieve the first element in queue  $Q$ 
        /* Explore node  $s$ , where  $\delta_s$  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          $Q.\text{add}(t)$ ;                      // 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          $Q.\text{add}(t)$ ;
            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.  $\sum_{i \in V(P_{rt})} \tilde{x}_i + \tilde{y}_{rt}^{l[t]} < 1$ ) then
            18            $Q_c.\text{add}(\sum_{i \in V(P_{rt})} x_i + y_{rt} \geq 1)$ 
            19           (resp.  $Q_c.\text{add}(\sum_{i \in V(P_{rt})} x_i + y_{rt}^{l[t]} \geq 1)$ );
            20         end
            21       else
            22          $Q \leftarrow \emptyset$ ;
            23         break;
    
```

DCNDP

BFS

- Differente implementazione per soluzioni intere e non.
- Per ogni soluzione incombente, le variabili assumono valore 0 o 1.
- Considerato che i vincoli 3.21 e 3.29 vengono violati solo quando la somma dei LHS è < di 1, solo i nodi $\bar{x} = 0$ di ogni soluzione incombente potenzialmente violerebbero il vincolo del cammino.
- Nella soluzione corrente intera, il set di nodi candidati root, consiste solo in nodi con valore $\bar{x} = 0$
- Seguendo questo concetto, nell'esplorare un nodo nello step 9, viene limitato ai soli neighbor non visitati cui valori nella soluzione incombente siano pari a 0
- Questo permette di identificare e aggiungere tutti i vincoli violati non sprecando tempo in altri branches

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      $s \leftarrow Q.\text{remove}$ ;                      // retrieve the first element in queue  $Q$ 
9     /* Explore node  $s$ , where  $\delta_s$  denotes neighbours of  $s$  */ 
10    for  $t \in \delta_s \setminus T$  do
11       $l[t] = l[s] + 1$ ;
12      if  $l[t] = 1$  then
13         $Q.\text{add}(t)$ ;                            // add  $t$  to queue  $Q$ 
14         $T \leftarrow T \cup \{t\}$ ;                      // mark  $t$  as visited
15      else if  $l[t] \in [2, L]$  then
16         $Q.\text{add}(t)$ ;
17         $T \leftarrow T \cup \{t\}$ ;                      // mark  $t$  as visited
18        /* check violation of (3.29) (resp. (3.21) for DCNDP-2b)
           for the path  $P_{rt}$  */ 
19        if  $\sum_{i \in V(P_{rt})} \tilde{x}_i + \tilde{y}_{rt} < 1$  (resp.  $\sum_{i \in V(P_{rt})} \tilde{x}_i + \tilde{y}_{rt}^{l[t]} < 1$ ) then
20           $Q_c.\text{add}(\sum_{i \in V(P_{rt})} x_i + y_{rt} \geq 1)$ 
           (resp.  $Q_c.\text{add}(\sum_{i \in V(P_{rt})} x_i + y_{rt}^{l[t]} \geq 1)$ );
21        end
22      else
23         $Q \leftarrow \emptyset$ ;
24      break;

```

BFS

- Nel caso delle soluzioni non intere, il problema di separazione consiste nel risolvere un problema di shortest path con pesi degli archi in un grafo di supporto.
- Il peso degli archi è dato dal valore del valore del rilassamento LP.
- BFS creato basandosi sul valore del rilassamento LP
- Nodo root r con minor valore LP scelto come primo nodo nella generazione
- Nodi esplorati in ordine crescente di valore LP per i vicini, i vicini con peso minore verranno visitati per primi

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      $s \leftarrow Q.\text{remove}$ ;                      // retrieve the first element in queue  $Q$ 
9     /* Explore node  $s$ , where  $\delta_s$  denotes neighbours of  $s$  */
10    for  $t \in \delta_s \setminus T$  do
11       $l[t] = l[s] + 1$ ;
12      if  $l[t] = 1$  then
13         $Q.\text{add}(t)$ ;                            // add  $t$  to queue  $Q$ 
14       $T \leftarrow T \cup \{t\}$ ;                      // mark  $t$  as visited
15      else if  $l[t] \in [2, L]$  then
16         $Q.\text{add}(t)$ ;
17         $T \leftarrow T \cup \{t\}$ ;                      // mark  $t$  as visited
18        /* check violation of (3.29) (resp. (3.21) for DCNDP-2b)
           for the path  $P_{rt}$  */
19        if  $\sum_{i \in V(P_{rt})} \tilde{x}_i + \tilde{y}_{rt} < 1$  (resp.  $\sum_{i \in V(P_{rt})} \tilde{x}_i + \tilde{y}_{rt}^{l[t]} < 1$ ) then
20           $Q_c.\text{add}(\sum_{i \in V(P_{rt})} x_i + y_{rt} \geq 1)$ 
           (resp.  $Q_c.\text{add}(\sum_{i \in V(P_{rt})} x_i + y_{rt}^{l[t]} \geq 1)$ );
21        end
22      else
23         $Q \leftarrow \emptyset$ ;
24      break;

```

Callback & BFS

```

def first_depth_k_bfs(input_graph: nx.Graph, cost: dict, connect: dict, k_value: int, cut_limit: int, model: 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_level = levels[vertex] + 1
                    # direct neighbours of root node
                    if new_level == 1:
                        predecessor[neighbour] = vertex
                        levels[neighbour] = new_level
                        # mark neighbours of node as visited to avoid revisiting
                        visited.add(neighbour)
                        # add neighbours of node to queue
                        queue.append(neighbour)

```

```

# new_level in range(2,k)
elif new_level in range(2, k_value):
    predecessor[neighbour] = vertex
    levels[neighbour] = new_level
    # mark neighbours of node as visited to avoid revisiting
    visited.add(neighbour)
    # add neighbours of node to queue
    queue.append(neighbour)
    i = min([root, neighbour])
    j = max([root, neighbour])
    if connect[(i, j)] < 1 - 1e-5:
        model.cbLazy(
            model._x_delete[neighbour] + model._x_delete[vertex] +
            model._x_delete[root] + model._u_connect[i, j] >= 1)
        cut_count += 1

```

```

def cut(model, where):
    global bound_check, node_count, bound_save
    cost = {}
    connect = {}

    if where == GRB.Callback.MIPSOL:
        # get MIPSOL_OBJBDN = Current best objective bound
        bound_save = model.cbGet(GRB.callback.MIPSOL_OBJBDN)
        # get MIPSOL_NODCNT = Current explored node count
        current_explored_node_count = model.cbGet(GRB.Callback.MIPSOL_NODCNT)
        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:
                    connect[(i, j)] = abs(model.cbGetSolution(model._u_connect[i, j]))
    if current_explored_node_count == 0:
        first_depth_k_bfs(G, cost, connect, k, GRB.INFINITY, model)
    else:
        first_depth_k_bfs(G, cost, connect, k, 150, model)

```

```

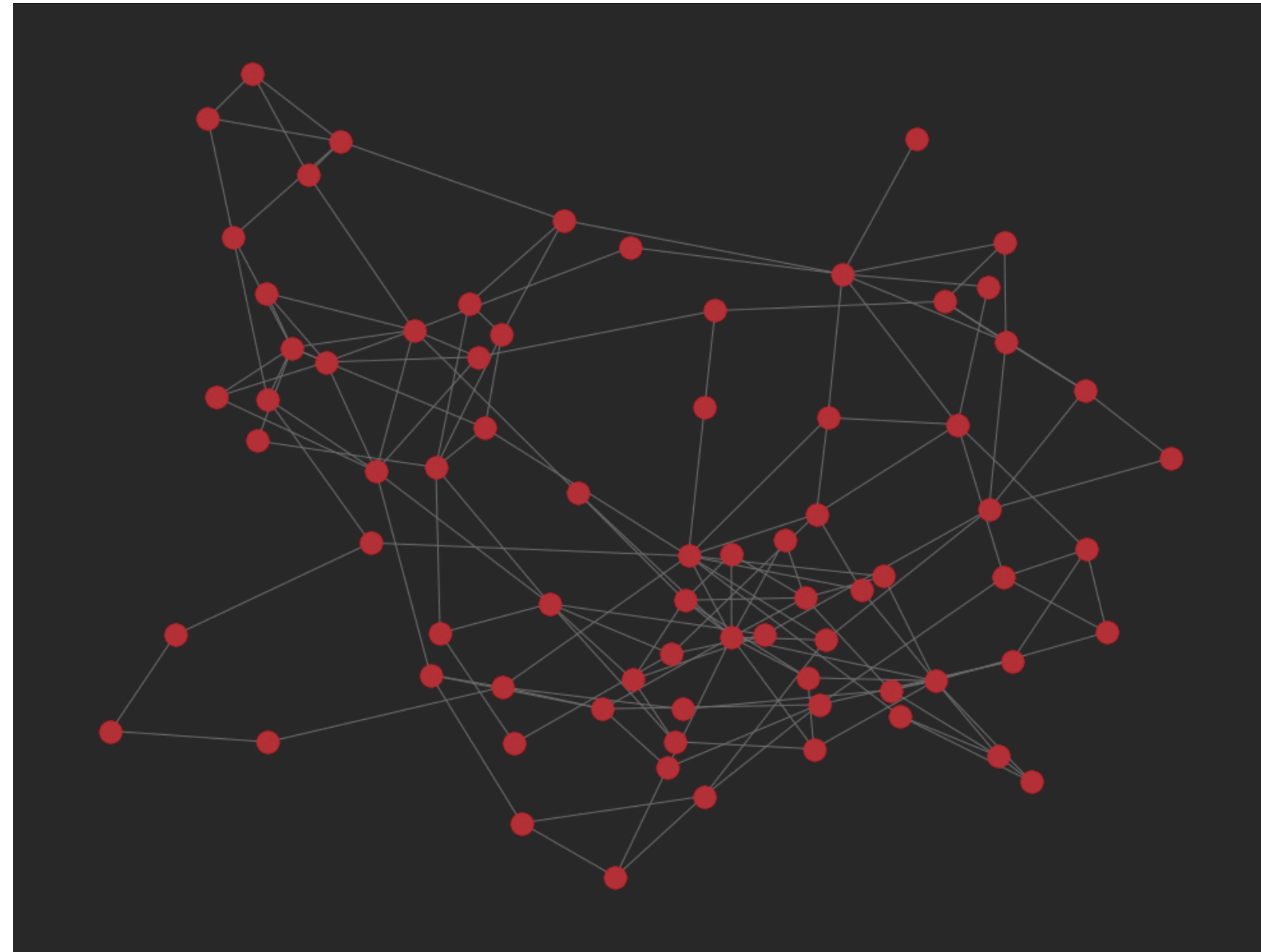
# Currently exploring a MIP node
elif where == GRB.Callback.MIPNODE: # if relaxed solution
    # Optimization status of current MIP node if it is OPTIMAL
    if model.cbGet(GRB.Callback.MIPNODE_STATUS) == GRB.Status.OPTIMAL:
        # get current best objective bound
        current_best_obj_bound = model.cbGet(GRB.callback.MIPNODE_OBJBND)
        # get current explored node count
        current_explored_node_count = int(model.cbGet(GRB.callback.MIPNODE_NODCNT))
        if bound_save == current_best_obj_bound:
            bound_check += 1
            if bound_check >= 5 and current_explored_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:
                    # retrieve values from the node relaxation solution at the current node
                    connect[(i, j)] = abs(model.cbgetNodeRel(model._u_connect[i, j]))

            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, k, 150, model)

```

DCNDP

- DCNDP_1 con $B=0.05n$
- Grafo : Sanjuansur



DCNDP



Computational experiments

Experiments

Hardware & Software

- Test sperimentali e computazionali eseguiti su Macbook Pro con Apple M1, 8GB di Ram e MacOS 12.4
- Modello e algoritmi scritti in Python 3.9, usando Gurobi 9.5.1 come suite di ottimizzazione.
- NetworkX usato per la generazione dei grafi artificiali e per la manipolazione sui grafi stessi
- 3 classi di random graph generator usati, predisposti da NetworkX:
 - Barabasi-Albert
 - Erdos-Renyi
 - Uniform

Experiments

Istanze di Test

- Le istanze di test sono state effettuate su modelli reali e su modelli creati artificialmente.
- Le istanze reali sono un subset dei network presenti nei datasets Pajek e UCINET datasets

Experiments

Istanze Reali

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 network
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

Istanze Synthetic

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

Parametri & Impostazioni

- Per DCNDP_1, dove minimizziamo il numero di coppie di nodi connessi da path di lunghezza al più K, il valore K è settato su K = 3.
- Per DCNDP_2, dove minimizzazione l'efficienza del grafo, il valore di L è uguale al diametro del grafo in input.
- Budget B fissato fra 1% e 10% per i grafi appartenenti al dataset dei real-world
- Budget B fissato fra 5% e 10% per i grafi appartenenti alle istanze generate artificialmente

Experiments

Premessa Risultati

- Gli esperimenti sono stati effettuati tenendo in considerazione le diverse dimensioni (nodi, archi) e classi di grafi con differenti densità di archi e diametri.
- In ogni tabella, insieme alle caratteristiche principali del grafo considerato, sono presenti il tempo di computazione impiegato espresso in s
- La colonna %InitObj rappresenta il valore dell'obiettivo iniziale del grafo in input:
 - Per DCNDP-1, questa è data dalla percentuale di coppie di nodi connessi da paths di lunghezza al più K
 - Per DCNDP-2, rappresenta l'efficiency della comunicazione iniziale del grafo in input
- La colonna %FinObj rappresenta il valore dell'obiettivo finale alla fine del processo di ottimizzazione
- La colonna call_back_time rappresenta il tempo computazionale espresso in secondi impiegato all'interno della routine BFS

Experiments

DCNDP_1 con B = 0.05n

graph	n_nodes	n_edges	diameter	init_obj	final_obj	n_vars_sol	distance-based pairwise connectivity	critical_node	callback_time (s)	run_time (s)
Hi-Tech	33	91	5	88,30%	75,19%	561	397	[2]	0,05	0,12
Karate	34	78	5	85,6%	57,75%	595	324	[1]	0,04	0,07
Mexican	35	117	4	98,0%	88,57%	630	527	[12]	0,07	0,16
Sawmill	36	62	8	63,0%	34,1%	666	215	[12]	0,03	0,07
Chesapeake	39	170	3	100,0%	93,93%	780	696	[39]	0,2	0,32
Dolphins	62	159	8	58,5%	43,36%	1953	820	[36, 40, 51]	0,48	0,88
Lesmiserable	77	254	5	85,4%	31,78%	3003	930	[11, 23, 27]	0,33	0,73
Santafe	118	200	12	32,9%	4,42%	7021	305	[1, 2, 12, 6, 24]	0,47	0,69
Sanjuansur	75	155	7	48,7%	28,94%	2850	803	[57, 34, 41]	0,13	0,24
Attiro	59	128	8	68,00%	43,42%	1770	743	[53, 35]	0,09	0,16
UsaAir97	332	2126	6	84,8%	19,33%	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%	27261	4629	[1, 3, 6, 14, 55, 215, 239, 240, 243, 252, 265]	31,1	69,2

Experiments

DCNDP_1 con B = 0.1n

graph	n_nodes	n_edges	diameter	init_obj	final_obj	n_vars_sol	distance-based pairwise connectivity	critical_node	callback_time (s)	run_time (s)
Hi-Tech	33	91	5	88,30%	55,49%	561	297	[2, 21, 19]	0,19	0,36
Karate	34	78	5	85,6%	26,2%	595	147	[1, 33, 34]	0,03	0,05
Mexican	35	117	4	98,0%	60,17%	630	358	[18, 10, 12]	0,06	0,16
Sawmill	36	62	8	63,0%	21,43%	666	135	[12, 36, 31]	0,02	0,04
Chesapeake	39	170	3	100,0%	69,1%	780	512	[39, 36, 38]	0,29	0,55
Dolphins	62	159	8	58,5%	30,83%	1953	583	[14, 17, 20, 29, 37, 40]	0,64	1,22
Lesmiserable	77	254	5	85,4%	11,04%	3003	323	[11, 23, 25, 26, 27, 48, 55]	0,36	0,57
Santafe	118	200	12	32,9%	1,68%	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%	2850	457	[57, 34, 41, 68, 72, 39, 40]	0,19	0,42
Attiro	59	128	8	68,00%	25,95%	1770	444	[53, 47, 51, 35, 42]	0,15	0,35
UsaAir97	332	2126	6	84,8%	5,64%	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%	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 con B = 0.05n

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

Experiments

DCNDP_2 con B = 0.1n

graph	n_nodes	n_edges	diameter	init_obj	final_obj	n_vars_sol	distance-based pairwise connectivity	critical_node	callback_time (s)	run_time (s)
Hi-Tech	33	91	5	50,8%	32,81%	2673	173,23	[53, 47, 51, 35, 8]	0,11	0,4
Karate	34	78	5	49,20%	16,69%	2839	93,65	[1, 33, 34]	0,04	0,16
Mexican	35	117	4	54,9%	36,59%	2415	217,67	[18, 10, 12]	0,06	0,21
Sawmill	36	62	8	39,9%	14,17%	5076	89,23	[12, 36, 27]	0,04	0,09
Chesapeake	39	170	3	60,3%	35,87%	2262	265,83	[39, 36, 38]	0,09	0,25
Dolphins	62	159	8	37,9%	18,63%	15190	352,83	[28, 30, 36, 40, 45, 51]	0,58	1,38
Lesmiserable	77	254	5	43,2%	7,88%	14707	230,58	[11, 23, 25, 26, 27, 48, 55]	0,41	1,97
Santafe	118	200	12	27,09%	1,39%	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%	19500	399,87	[3, 57, 34, 68, 70, 72, 46]	0,68	7,9
Attiro	59	128	8	38,9%	22,30%	13747	381,62	[53, 47, 51, 35, 8]	0,66	7,18

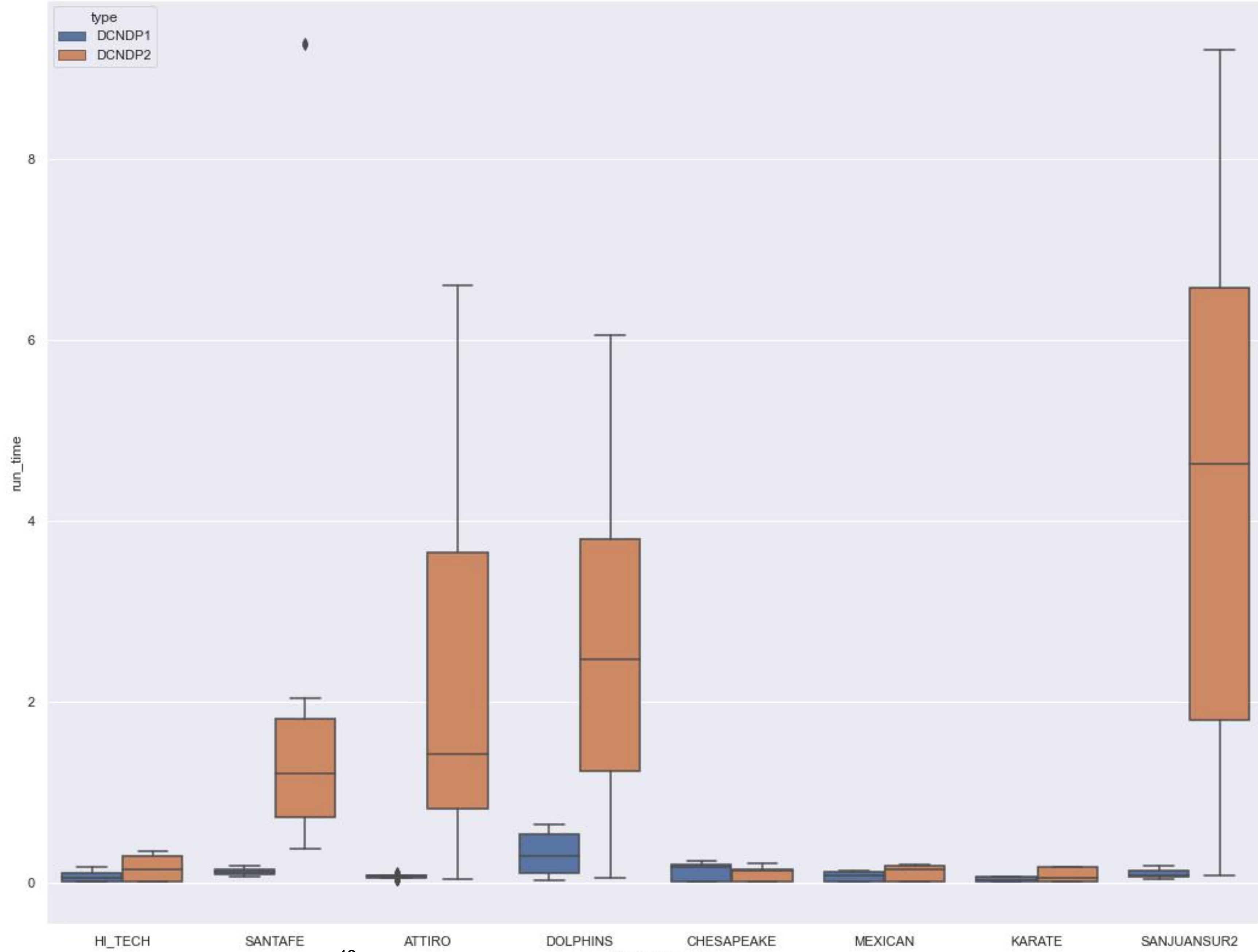
Experiments

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

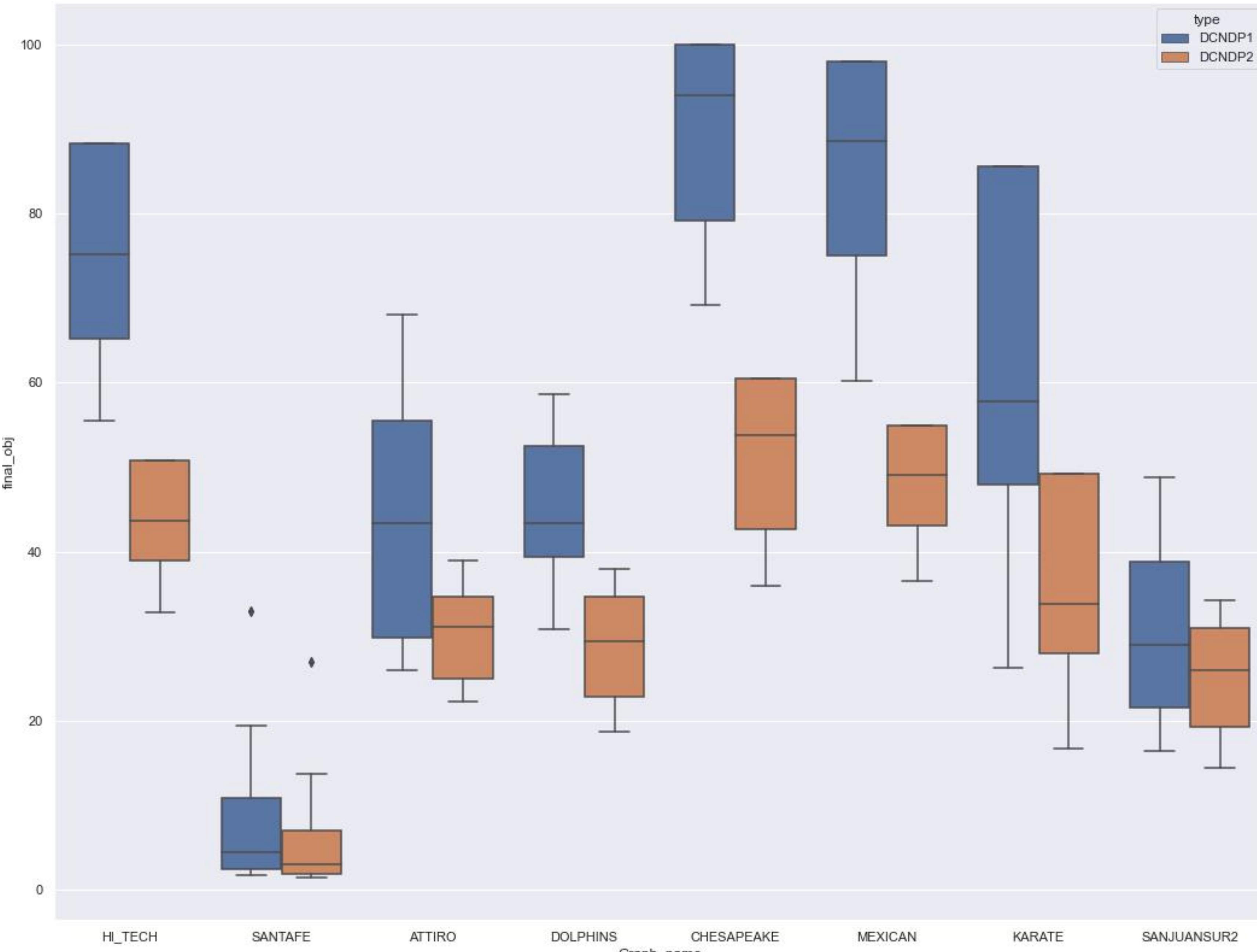
Experiments

DCNDP_1 vs DCNDP_2
run_time test su grafi
medi e piccoli



Experiments

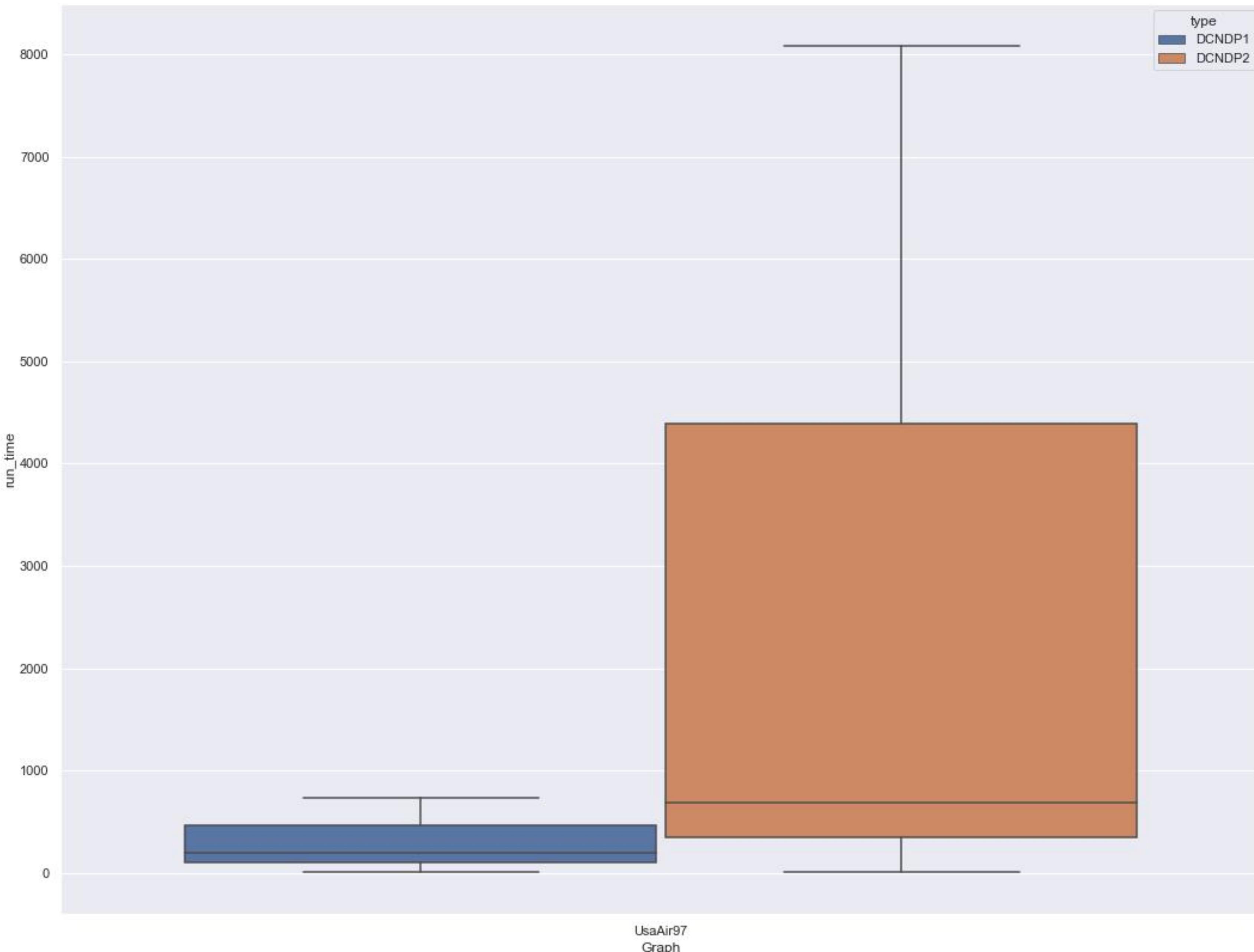
DCNDP_1 vs DCNDP_2 %gap



Experiments

UsaAir97 test.

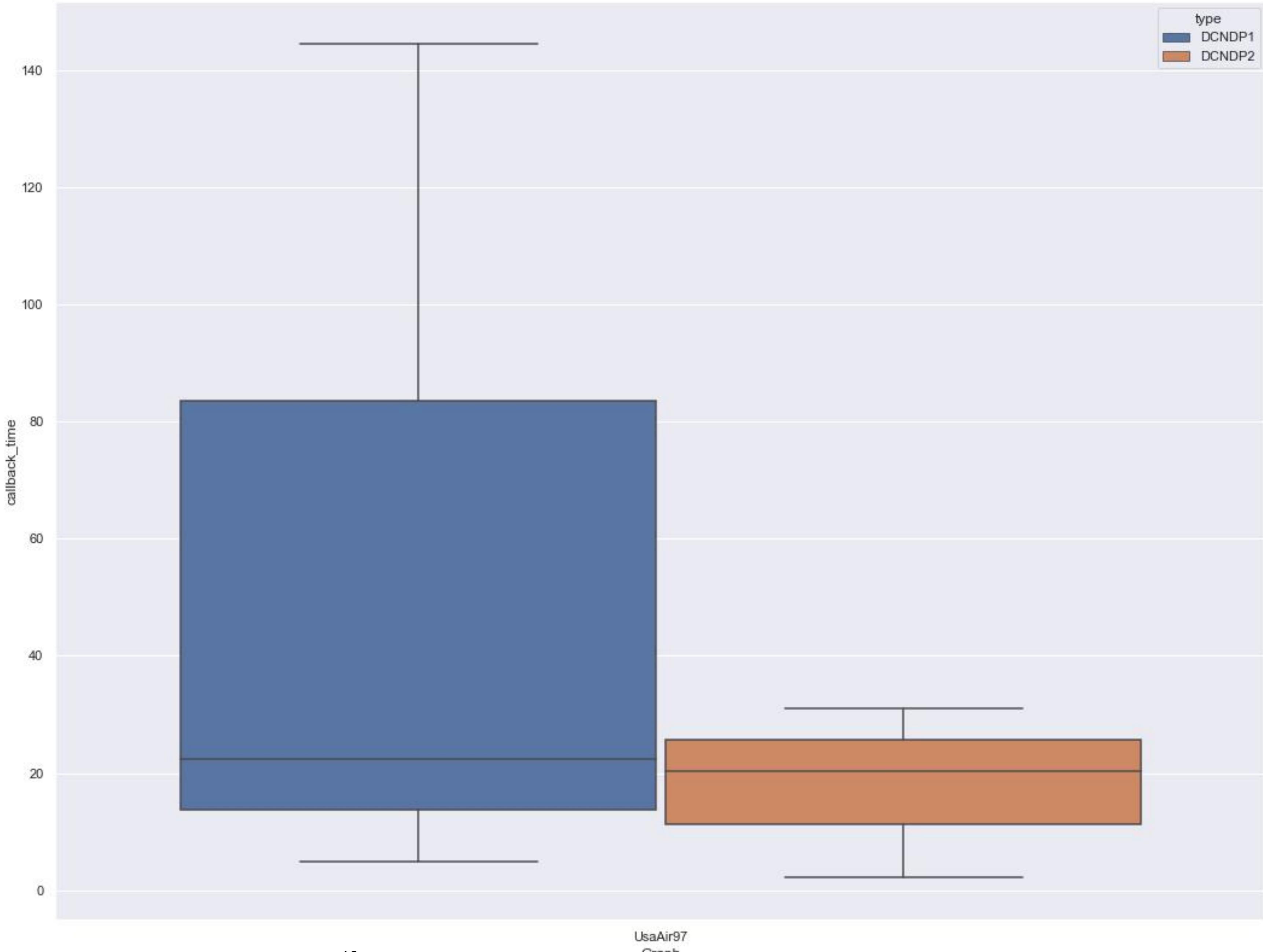
Runtime



Experiments

UsaAir97 test.

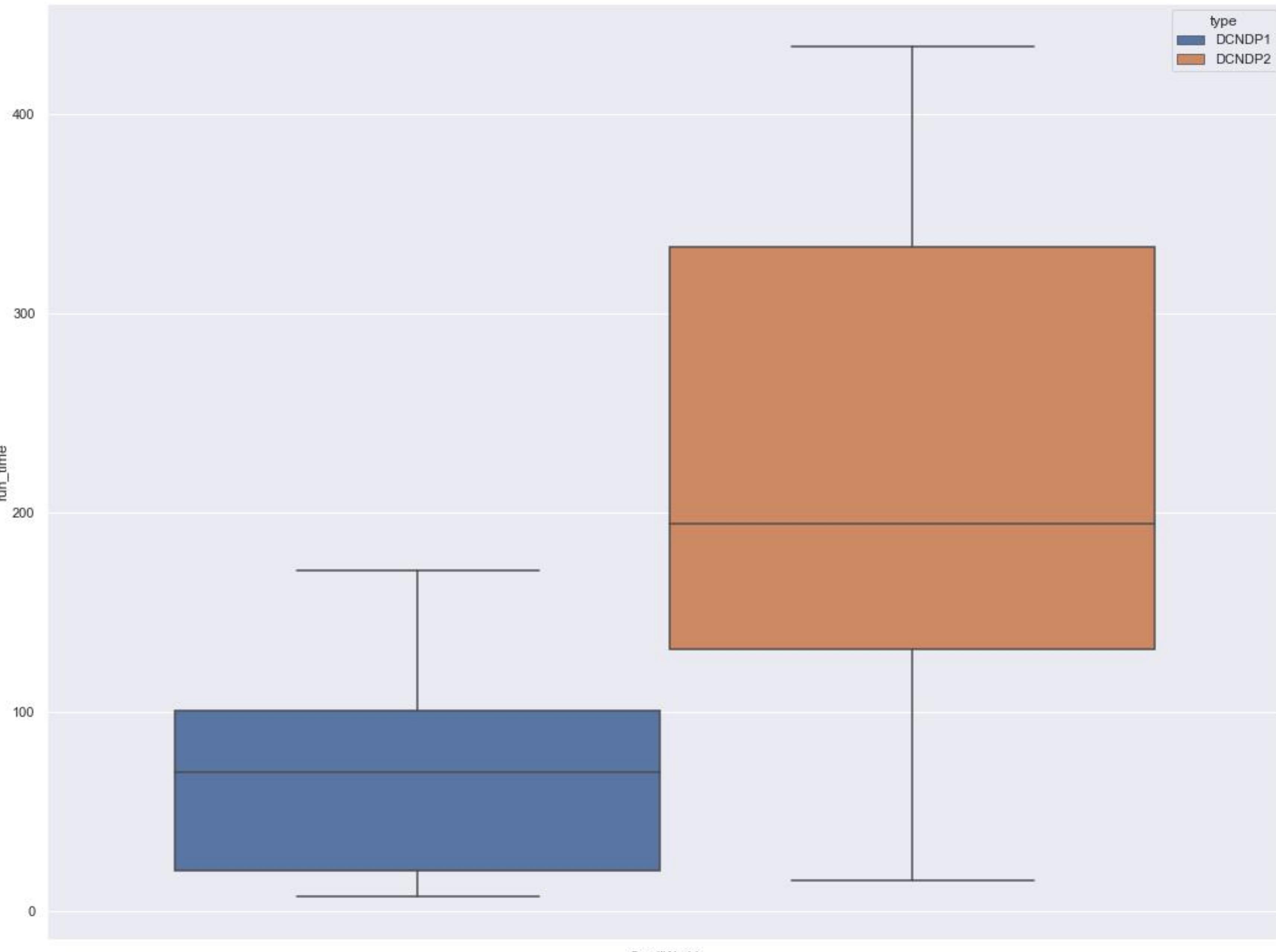
Callback time



Experiments

SmallWorld test.

Runtime



Experiments

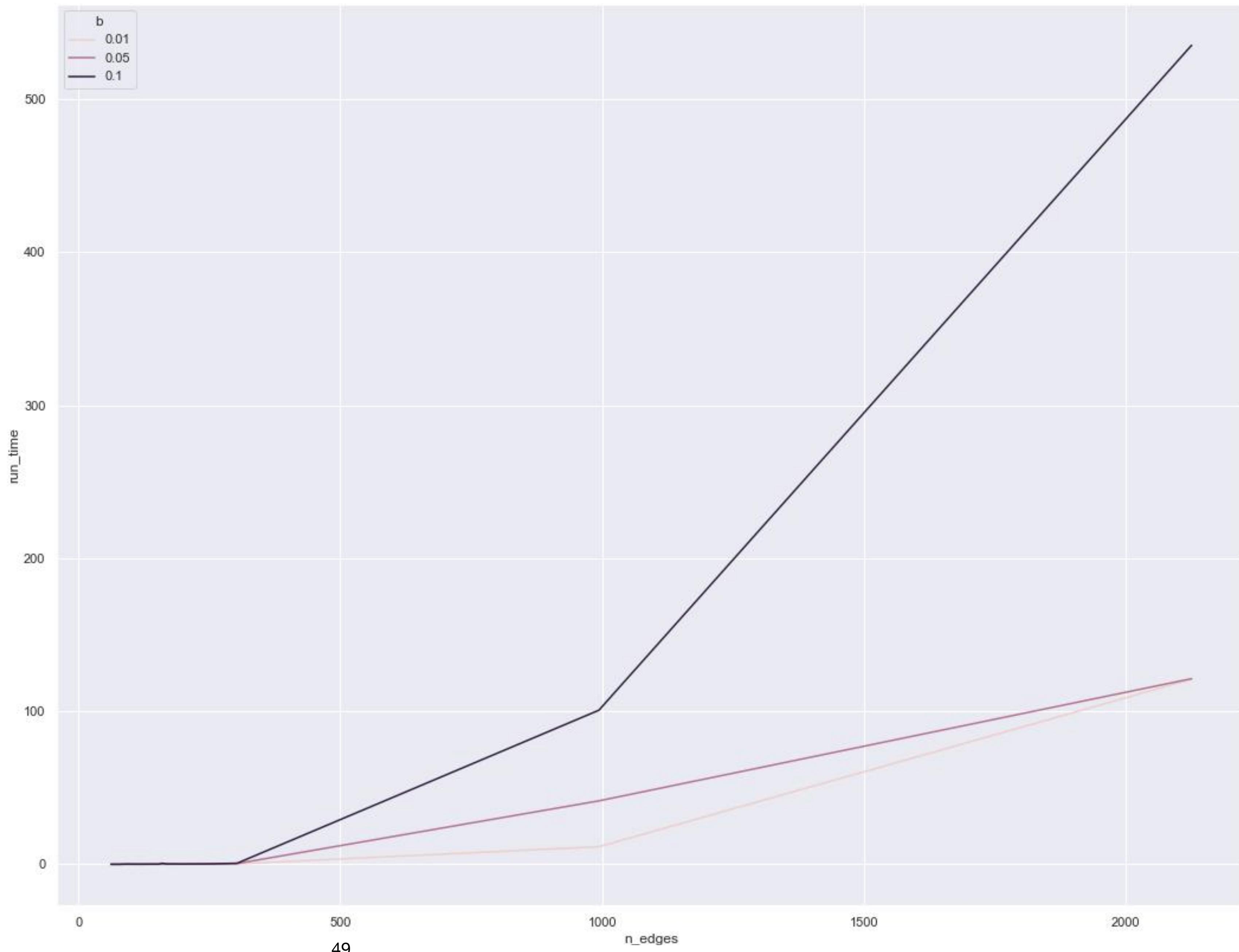
SmallWorld test.

Callback time



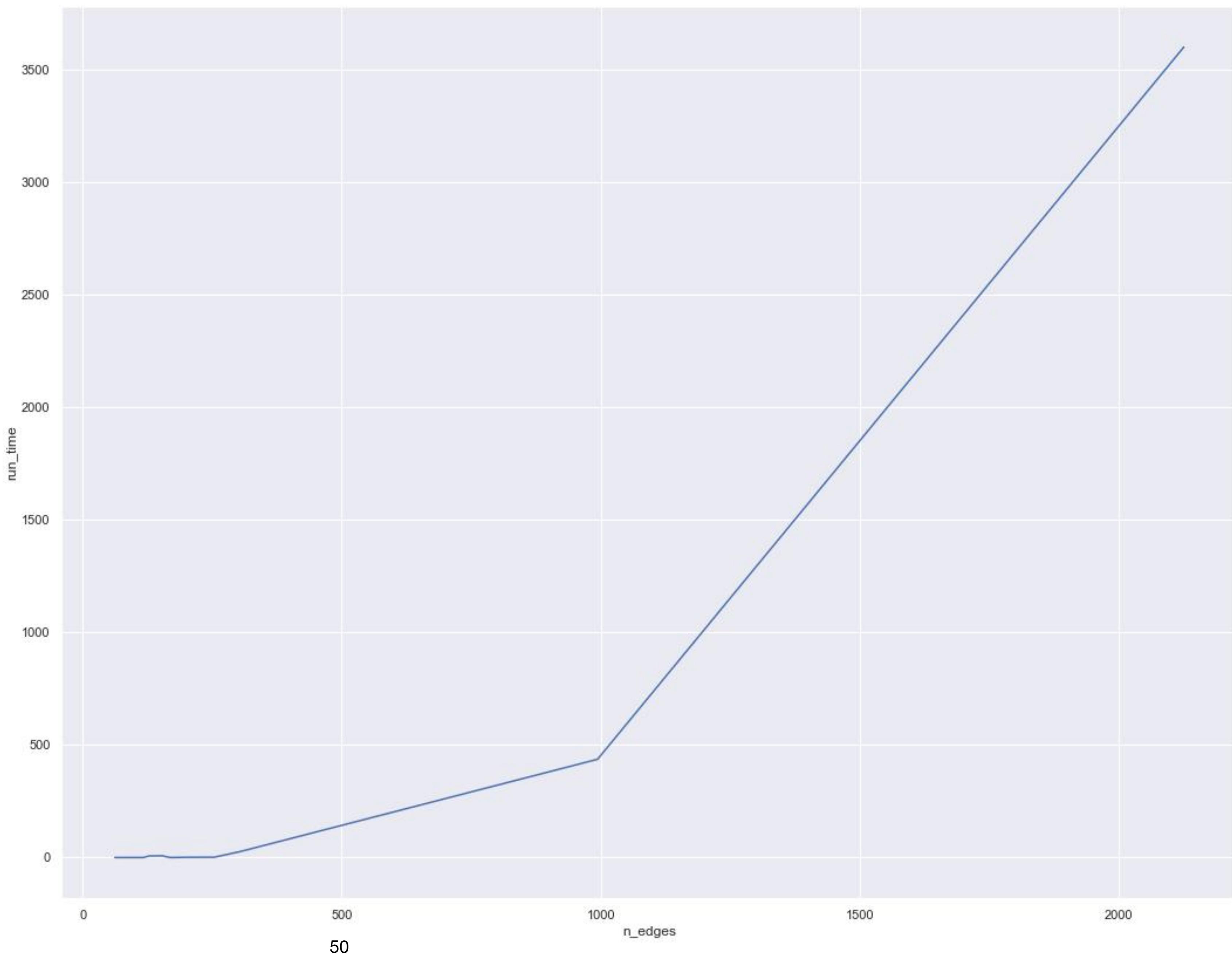
Experiments

DCNDP_1 su diversi threshold di B



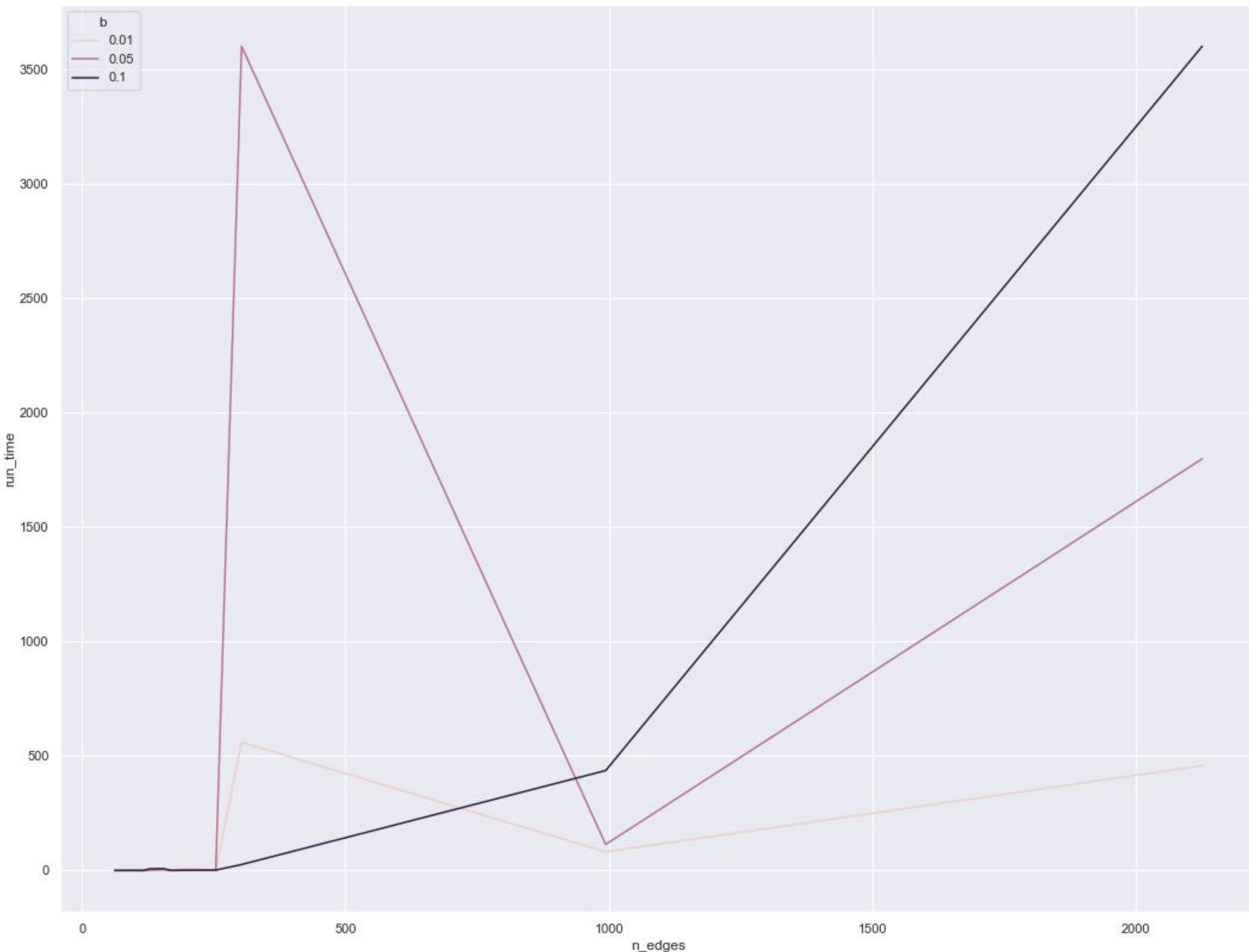
Experiments

DCNDP_2 con
B = 0.1N



Experiments

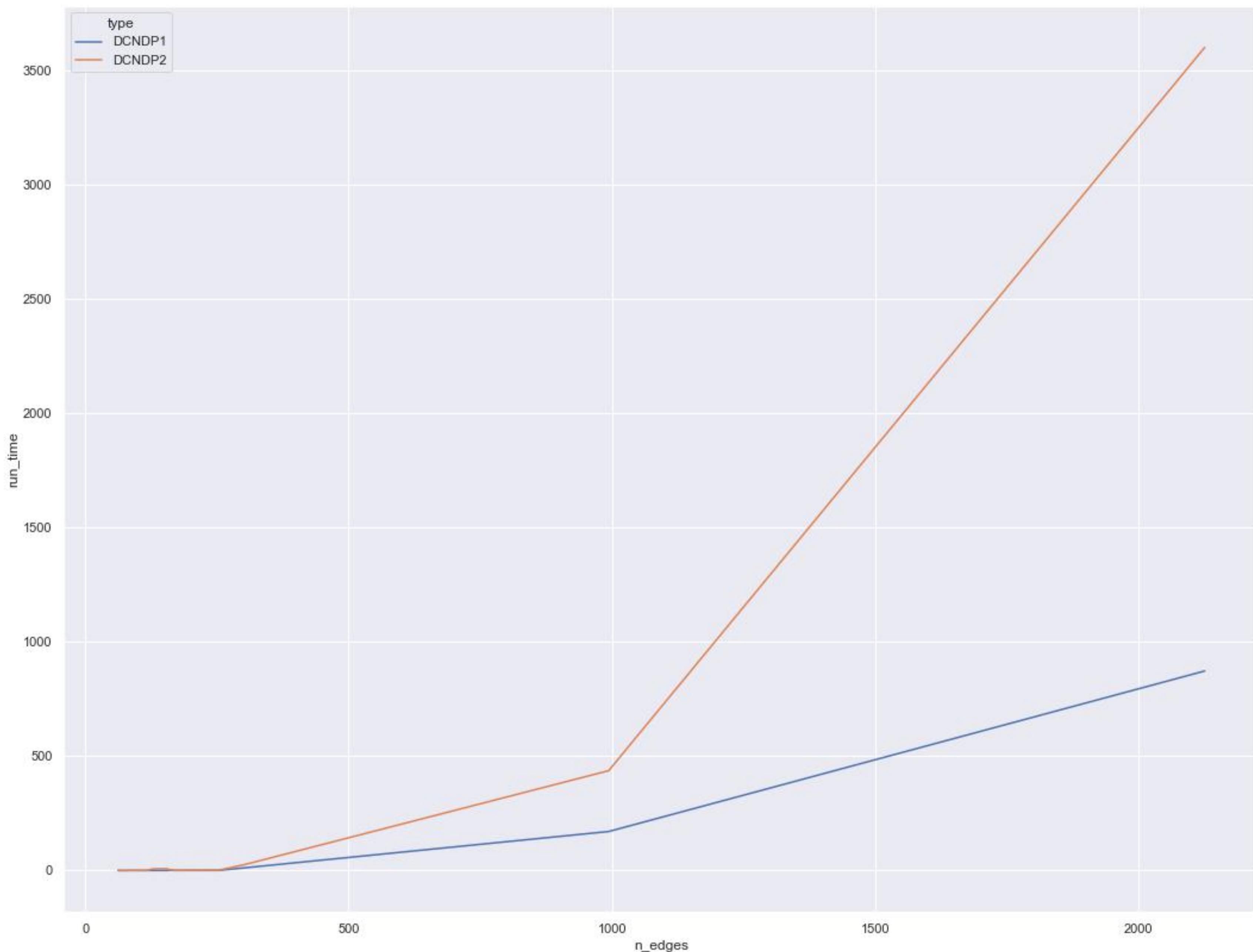
DCNDP_2 su diversi threshold di B



Experiments

DCNDP 1 vs 2 con

B = 0.1N



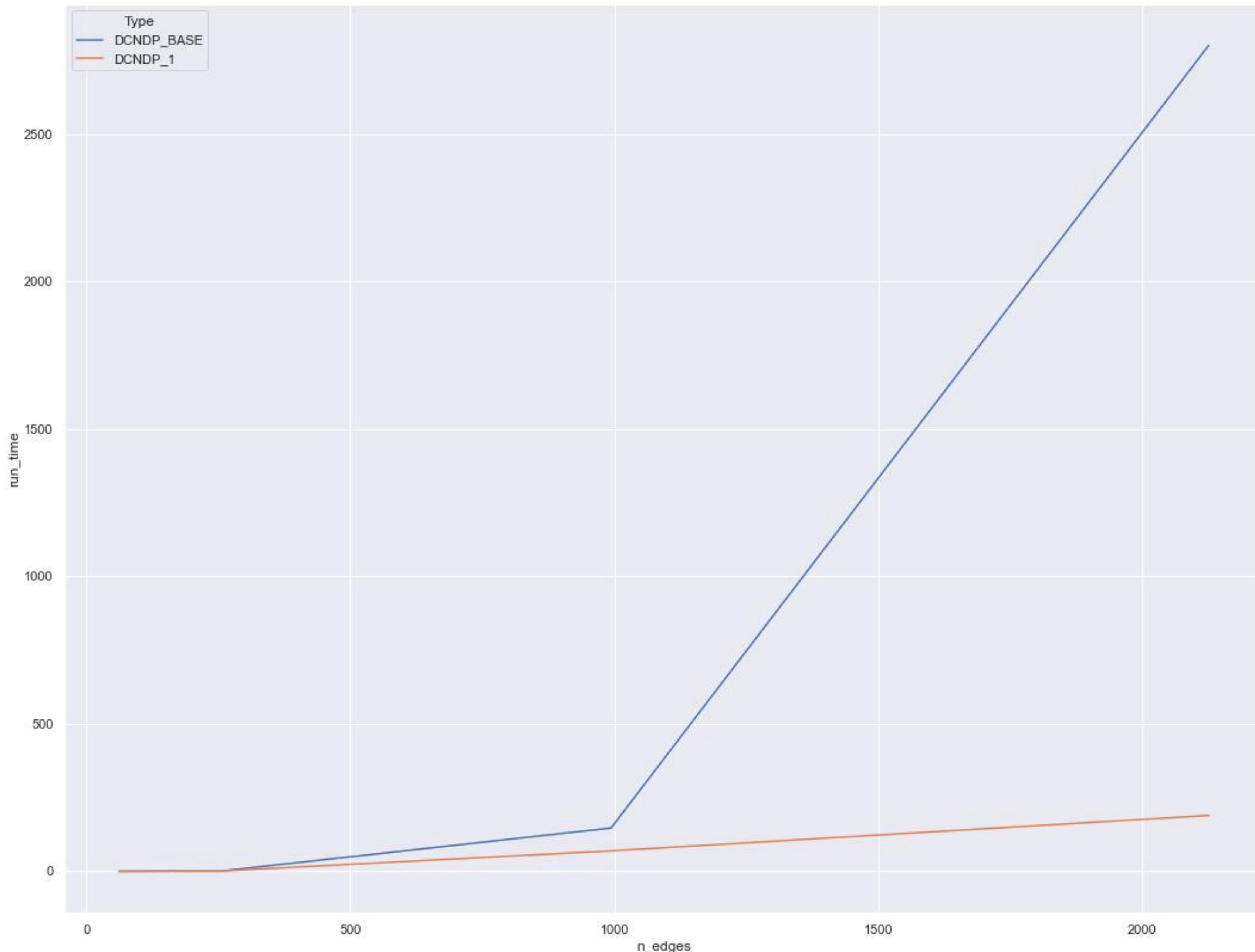
Experiments

DCNDP_BASE vs DCNDP_1 con 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	M	872,71
SmallWorld	233	994	4	146,52	69,2	M	170,52

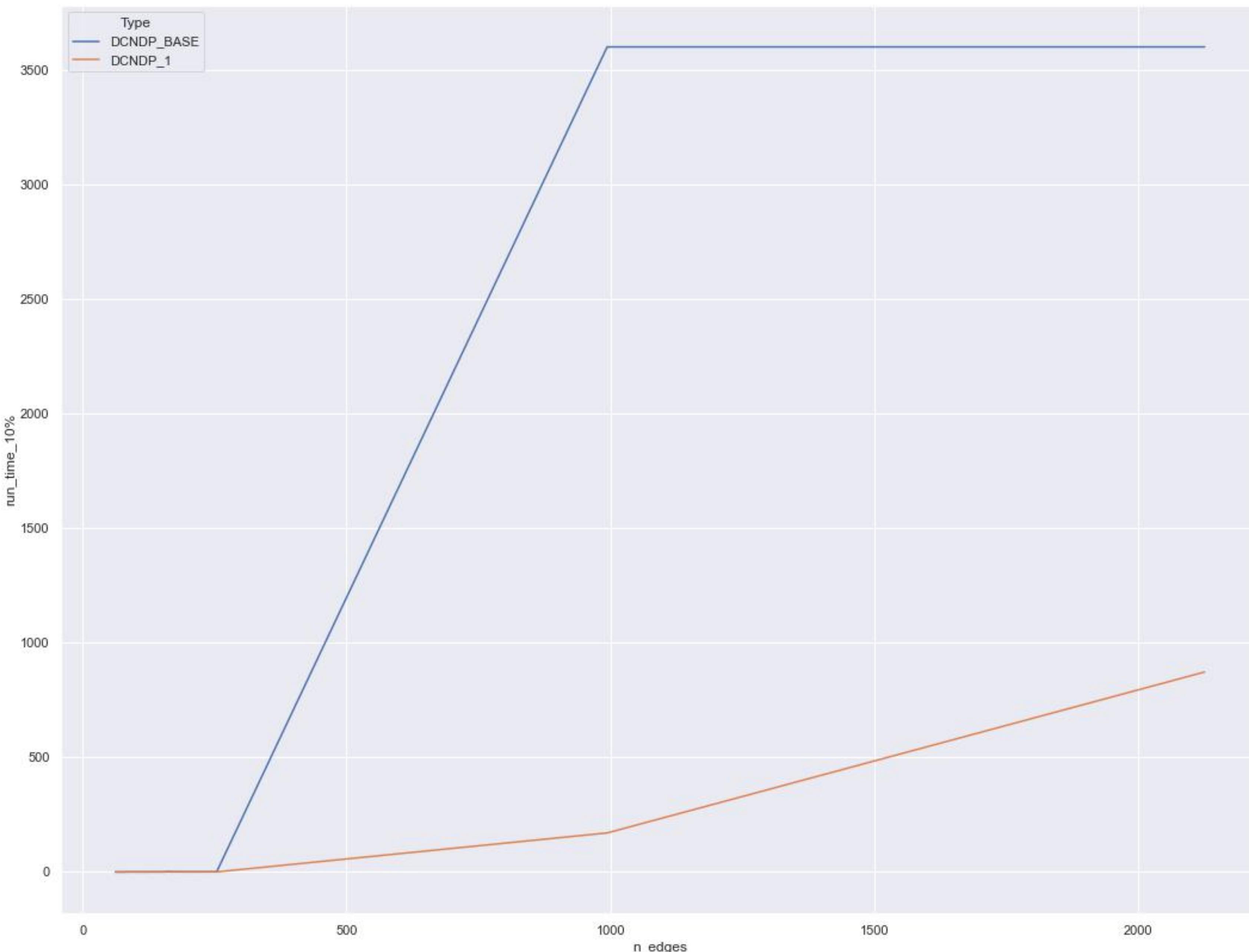
Experiments

**DCNDP_BASE vs
DCNDP_1 con B = 0.05N**



Experiments

**DCNDP_BASE vs
DCNDP_1 con B = 0.1N**



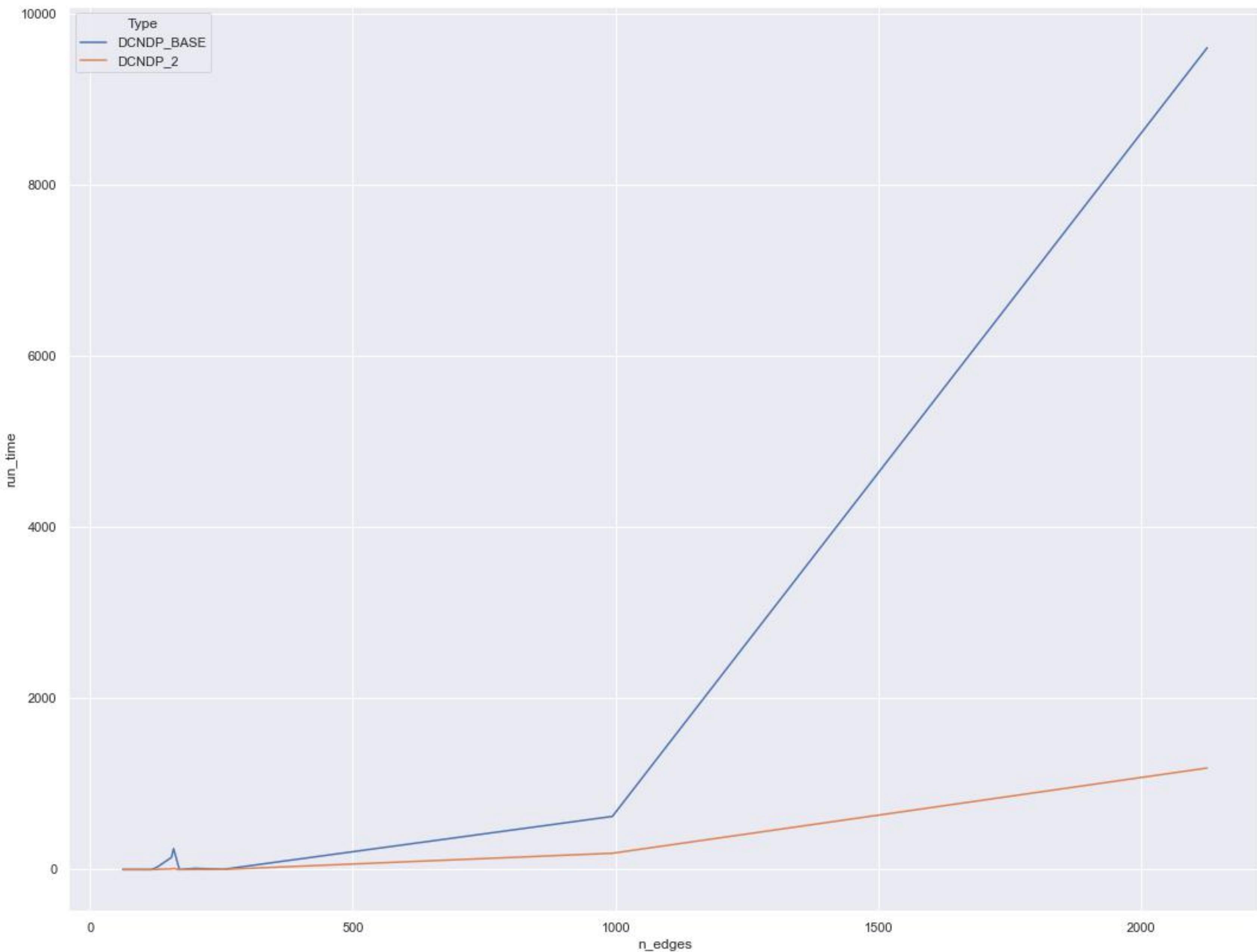
Experiments

DCNDP_BASE vs DCNDP_ con 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	M	730

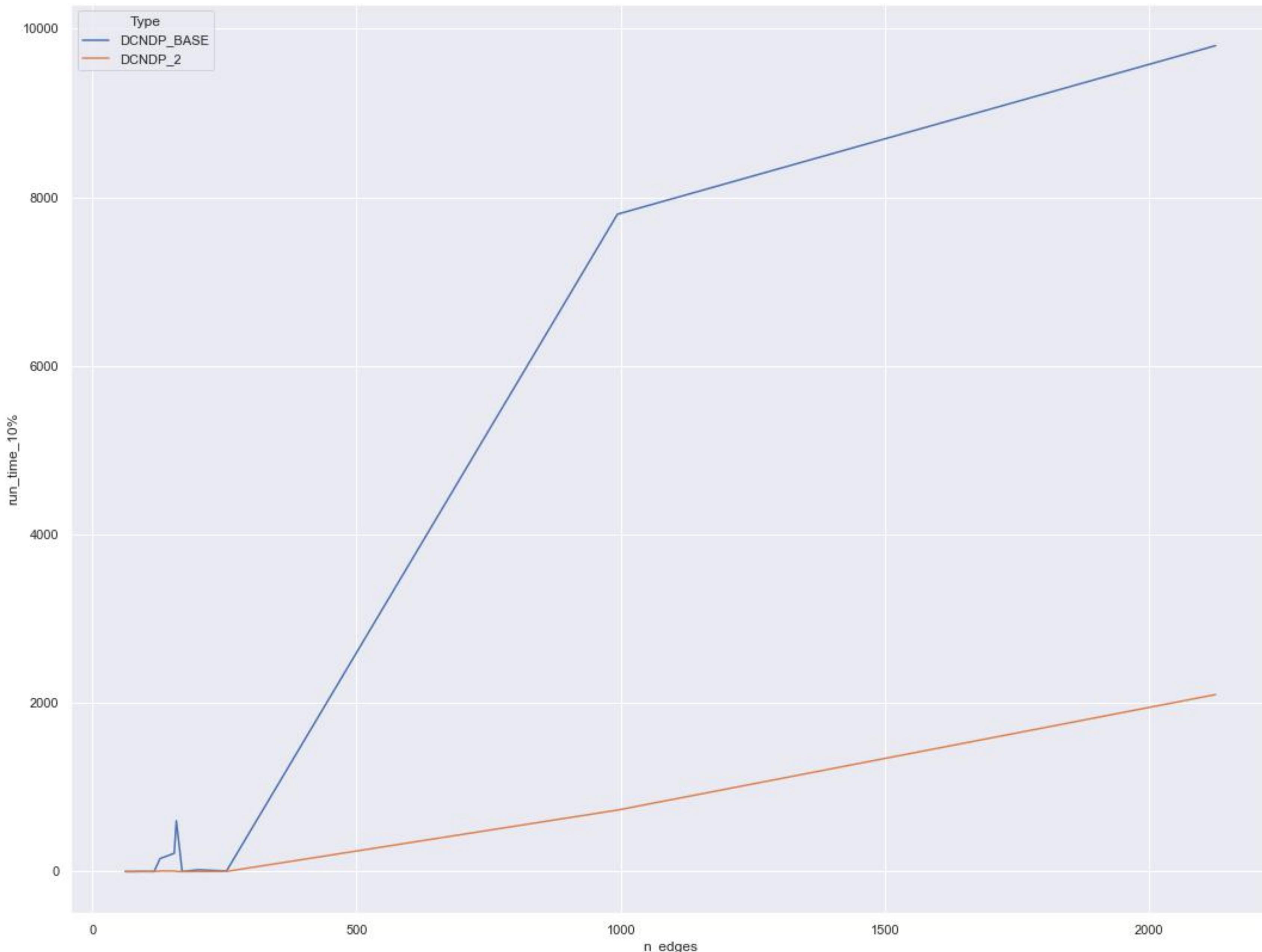
Experiments

**DCNDP_BASE vs
DCNDP_2 con B = 0.05N**



Experiments

**DCNDP_BASE vs
DCNDP_2 con B = 0.1N**



DCNDP for weighted graph :

Descrizione del problema e formulazione

DCNDP_WEIGHTED

Problem definition

- Sia $G = (V, E)$ un grafo pesato con un set finito V di nodi.
- Ad ogni coppia di nodi $(i, j) \in E$, denotiamo con w_{ij} un peso positivo associato a (i, j) interpretato come la lunghezza del path (i, j)
- Per ogni coppia di nodi i e j , la lunghezza di ogni path è data dalla somma dei pesi degli archi traversati durante il path
- Si assume che i pesi degli archi w_{ij} siano interi positivi che non è molto restrittiva come assunzione per la maggior parte delle applicazioni reali

DCNDP_WEIGHTED

New Compact Formulations

- (1) minimizza la connettività del grafo in input nel rispetto di una funzione di distance-based connectivity $f(\cdot) \geq d$ definita
- (2)-(5) ammette qualsiasi metrica non negativa di distanza come quelle definite per la classe 1 e classe 2.

$$\text{minimise} \quad \sum_{i,j \in V : i < j} y_{ij} \quad (1)$$

$$\text{s.t.} \quad \sum_{r \in V(P)} f(d)x_r + y_{ij} \geq f(d), \quad \forall P \in \mathcal{P}_L(i, j), (i, j) \in V, i < j \quad (2)$$

$$\sum_{i \in V} x_i \leq B \quad (3)$$

$$x_i \in \{0, 1\}, \quad \forall i \in V \quad (4)$$

$$y_{ij} \geq 0, \quad \forall (i, j) \in V, i < j \quad (5)$$

DCNDP_1_W

Explanation & Implementation

$$\text{Min.} \sum_{\substack{i,j \in V: i < j}} y_{ij}$$
$$\sum_{i \in V} x_i \leq B$$

```
# objective
obj = LinExpr(0)
for j in input_graph.nodes():
    for i in input_graph.nodes():
        if i < j:
            obj.add(u_connect[i, j])

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

- La funzione obiettivo minimizza la connettività del grafo in input nel rispetto di una funzione di distance-based connectivity $f(\cdot) \geq 0$

DCNDP_1_W

Explanation & Implementation

$$\sum_{r \in V(P)} f(d)x_r + y_{ij} \geq f(d), \quad \forall P \in P_l(i, j), \quad (i, j) \in V, i < j$$

- Questo vincolo indica che per ogni coppia di nodi i e j connessi da una path di lunghezza al più L , se nessuno dei nodi lungo il path è eliminato, allora la distance-based connectivity fra i e j è almeno $f(d)$ dove d è la lunghezza del path P

Computational experiments

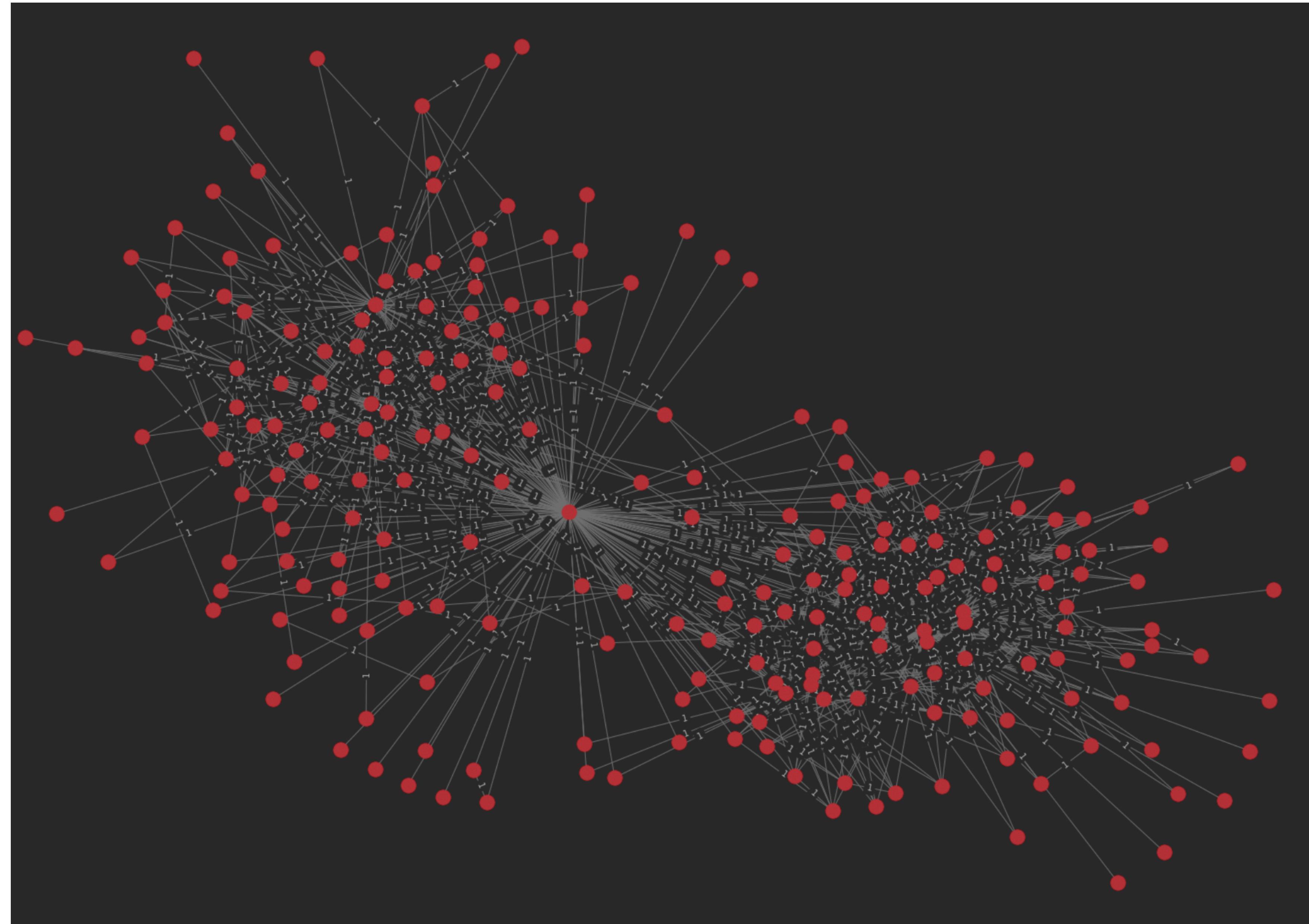
Experiments

Istanze di test

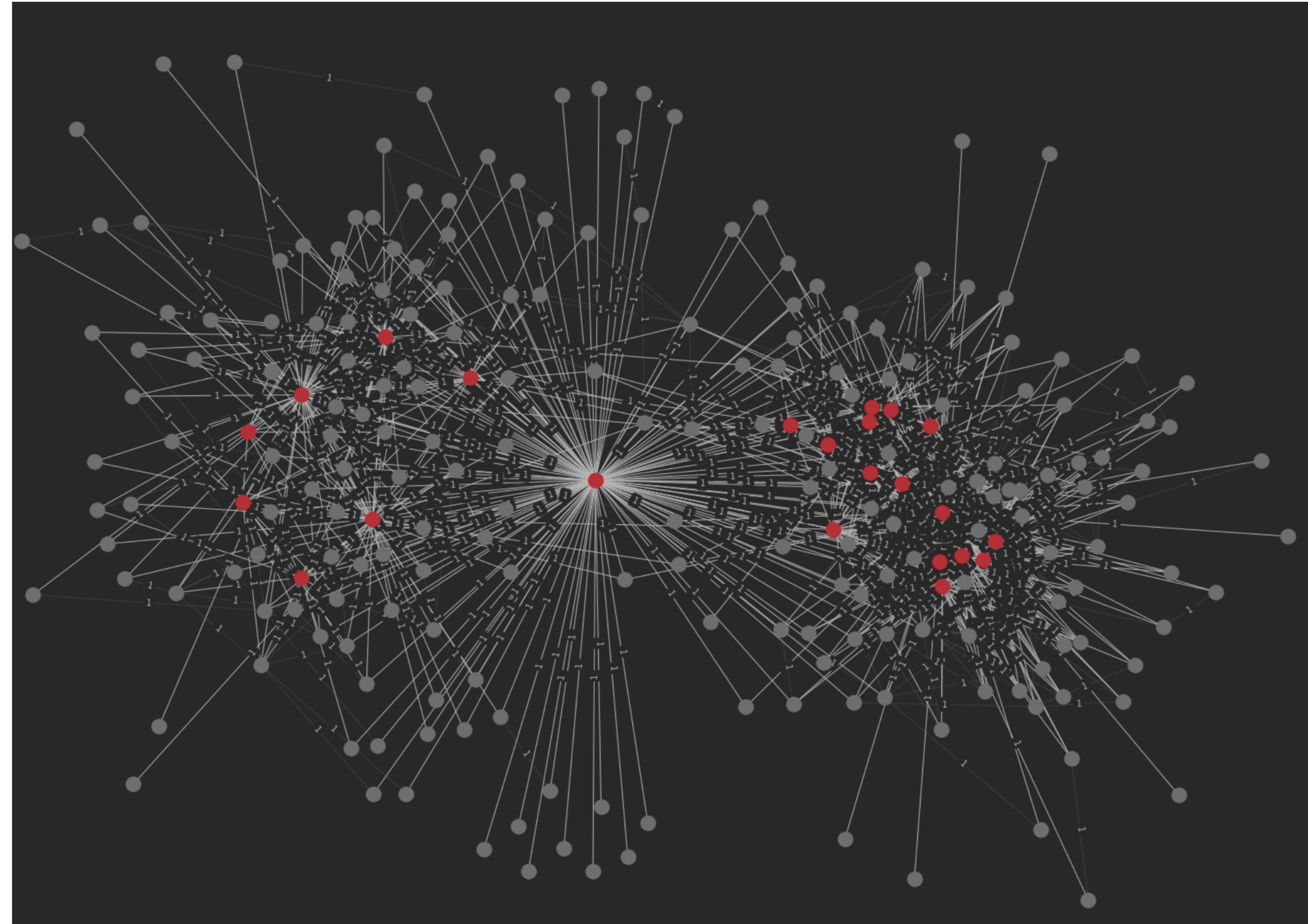
- Per le istanze di test, viene assegnato un peso piccolo agli archi che hanno un valore alto di betweenness centrality
- In maniera specifica :
 - Dati I valori di betweenness centrality b_{ij} degli archi (i,j) , settiamo $w_{ij} = \min(b, \bar{b})$
 - \bar{b} è l'intero più vicino ottenuto approssimando $0.1/b_{ij}$
 - b è dato da un upper-bound intero pre-stabilito. Per I test, sono utilizzati valori di b nel range $\{1,2,3,4,5,6\}$

DCNDP WEIGHTED

- Esempio DCNDP_W con UpperBound B = 6
- Grafo : SmallWorld
- Immagine del grafo in input
- Immagine del grafo con critical node rilevati

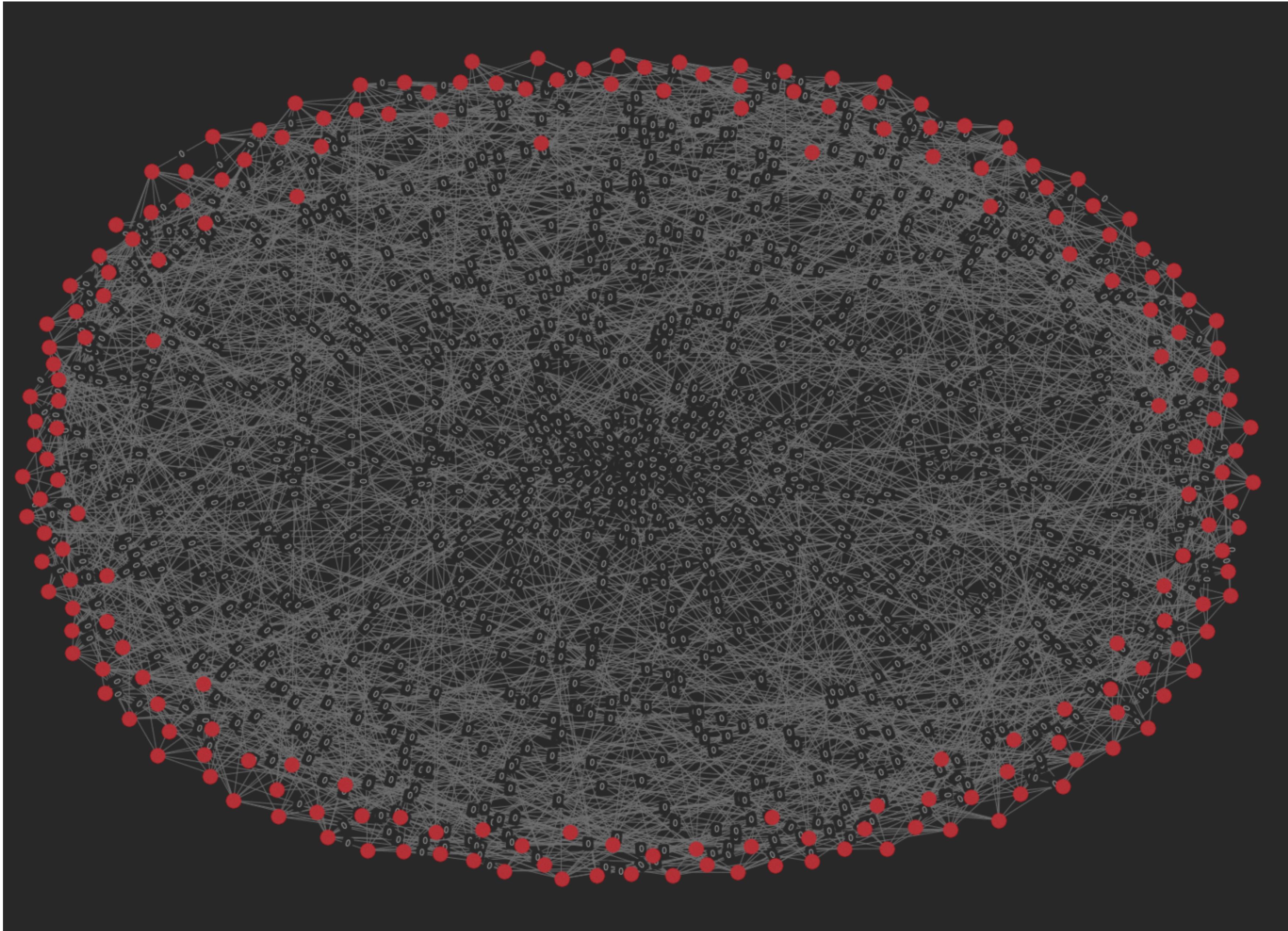


DCNDP WEIGHTED

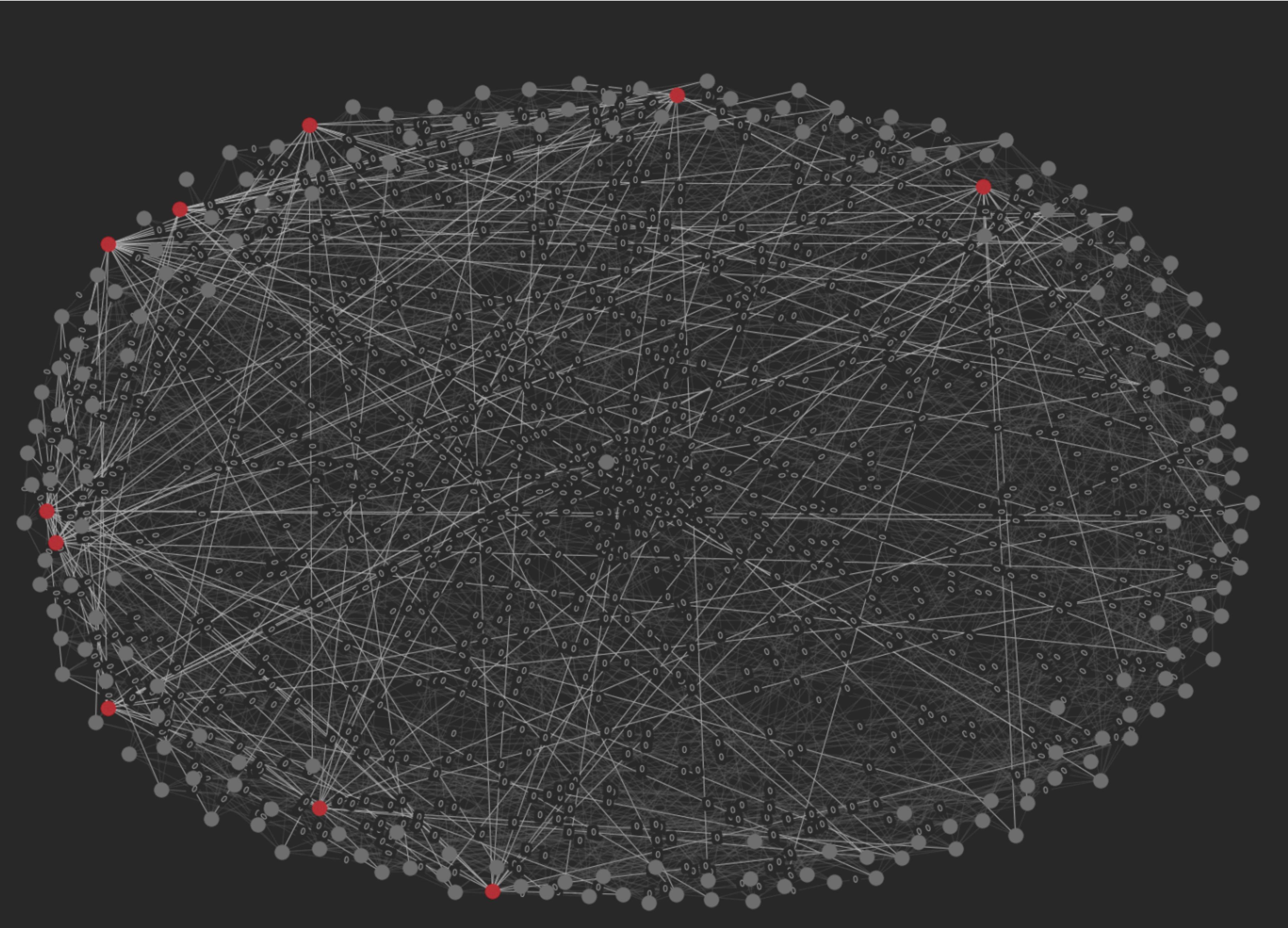


DCNDP WEIGHTED

- Esempio DCNDP_W con UpperBound B = 6
- Grafo : ER2
- Immagine del grafo in input
- Immagine del grafo con critical node rilevati



DCNDP WEIGHTED



Grazie

Reference

L. Gouveia, M. Leitner

Design of survivable networks with vulnerability constraints

L. Gouveia, M. Joyce-Moniz, M. Leitner

Branch-and-cut methods for the network design problem with vulnerability constraints

M. Di Summa, A. Grosso, M. Locatelli

Branch and cut algorithms for detecting critical nodes in undirected graphs

A. Veremyev, V. Boginski, E.L. Pasiliao

Exact identification of critical nodes in sparse networks via new compact formulations

B. Addis, M. Di Summa, A. Grosso

Identifying critical nodes in undirected graphs: Complexity results and polynomial algorithms for the case of bounded treewidth

A. Veremyev, O.A. Prokopyev, E.L. Pasiliao

An integer programming framework for critical elements detection in graphs

A. Veremyev, O.A. Prokopyev, E.L. Pasiliao

Critical nodes for distance-based connectivity and related problems in graphs

H. Salemi, A. Buchanan

Solving the distance-based critical node problem