Caracterización estructural de instancias

Brenda Yaneth Sotelo Benítez
5705

30 de abril de 2019

Introducción

En este trabajo se realiza la selección de un algoritmo generador, un algoritmo de flujo máximo y de acomodo que sean adecuados para alguna aplicación específica para generar cinco instancias y establecer si características en los vértices como la distribución de grado, coeficiente de agrupamiento, centralidad de cercanía y de carga, excentricidad y PageRank afectan en el tiempo de ejecución o el valor óptimo en el algoritmo seleccionado, así como analizar que vértices resultan ser buenas fuentes o sumideros y cuáles sería mejor no usar como ninguno si se busca tener un flujo alto, pero no batallar con el tiempo de ejecución.

Los instancias se obtuvieron por medio de la librería Networkx [1] y Matplotlib [2] de `Python` [3] para la generación de grafos y guardar imágenes en el formato *eps* respectivamente. Además se hace uso de la librería Numpy [4] para la suma de tiempos, cálculo de tiempos promedios y desviaciones estándar y Pandas [5] para el tratamiento de datos. El código empleado se obtuvo consultando documentación oficial [6].

Se presenta una breve descripción de los algoritmos seleccionados y las características estructurales de los vértices, visualización de los grafos, fragmentos relevantes de código y una sección de resultados donde se realiza una comparación de distintas variables numéricas a través de grupos, por ejemplo el tiempo de ejecución y las características para vértices.

Las pruebas se han realizado en una PC con procesador Intel Core i3 2.00 GHz y 8.00 GB de RAM.

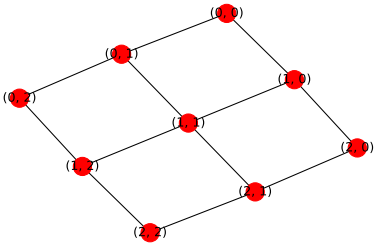
1. Instancias

Una red de distribución de agua tiene gran relevancia tanto en la industria como en la sociedad y puede ser representada mediante un grafo formado por aristas que representan tuberías y vértices, puntos de extracción y fuentes de abastecimiento como son los embalses y depósitos [7].

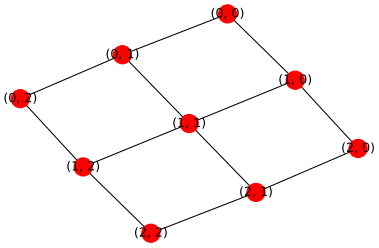
Por tal razón se ha seleccionado el generador `grid_graph` para hacer la representación de cinco instancias donde los vértices son puentes de extracción y fuentes de abastecimiento y las aristas tuberías, a las cuales se les han asignado pesos positivos normalmente distribuidos con media $\mu = 10$ y desviación estándar $\sigma = 3$. También se ha seleccionado el algoritmo `maximum_flow_value` y cada una de las instancias fueron visualizadas con el algoritmo de acomodo `kamada_kawai_layout` que produce un resultado entendible acorde a la aplicación. Las características de los algoritmos mencionados se pueden encontrar en la práctica dos y cuatro del repositorio de Sotelo [8].

En la figura 1 se muestran las instancias de orden 9 y 16 que se generaron con el algoritmo generador y de acomodo.

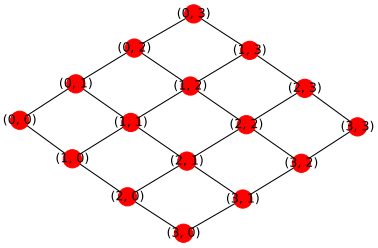
```
1     def __init__(self,x1,x2,x3):
2         self.dim_x=x1
3         self.dim_y=x2
4         self.num=x3
5
6     def crear(self):
7         self.G= nx.grid_graph(dim=[self.dim_x,self.dim_y])
8         self.pos=nx.kamada_kawai_layout(self.G)
9
10    def graficar(self):
11        nx.draw_networkx(self.G,self.pos)
```



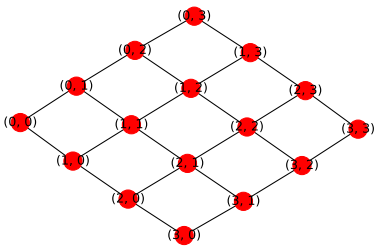
(a) Instancia 1



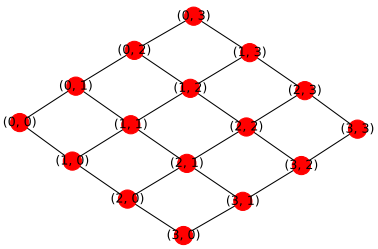
(b) Instancia 2



(c) Instancia 3



(d) Instancia 4



(e) Instancia 5

Figura 1: *Instancias generadas por el algoritmo grid_graph*

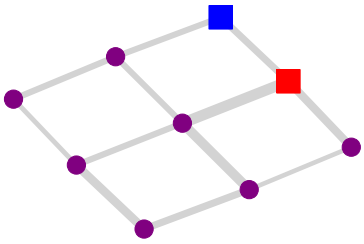
Posteriormente en la figura 2 se les asigna capacidad a las aristas, cuyo valor se representa en el grosor de las aristas. El vértice cuadrado azul representa el vértice fuente, el vértice cuadrado rojo representa el sumidero y el resto de los vértices tienen color morado.

En el cuadro 1 se indica el vértice fuente y sumidero de cada una de las instancias ya que en las siguientes visualizaciones no se muestra el número de nodo.

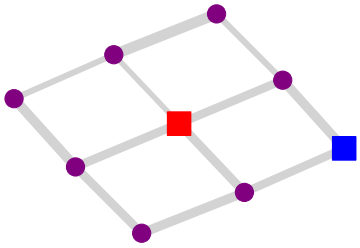
Cuadro 1: *Vértices fuente y sumidero*

Instancia	Fuente	Sumidero
1	(0,0)	(1,0)
2	(2,0)	(1,1)
3	(3,2)	(1,1)
4	(2,1)	(3,0)
5	(1,3)	(2,0)

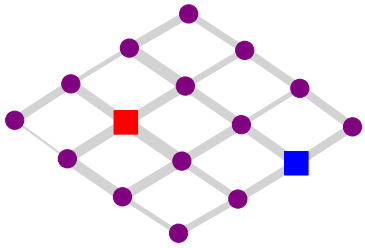
```
1     def Asignarpesos(self, media, desviacion):
2         self.media=media
3         self.desviacion=desviacion
4         K=[i for i in self.G.edges()]
5         Pesos=[round (max(0.1,random.gauss(self.media,self.
      desviacion)),2) for i in range(len(K))]
6         e=[]
7         for i in range(len(self.G.edges())):
8             a=(K[i][0],K[i][1],Pesos[i])
9             e.append(a)
10        self.G.add_weighted_edges_from(e)
11
12    def solucion(self):
13        #Seleccionar nodo fuente y nodo destino
14        buscar=True
15        while buscar:
16
17            origen1=random.randint(0, self.dim_x-1)
18            origen2=random.randint(0, self.dim_y-1)
19            destino1=random.randint(0, self.dim_x-1)
20            destino2=random.randint(0, self.dim_y-1)
```



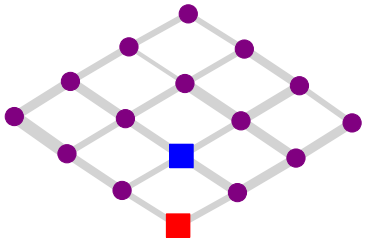
(a) Instancia 1



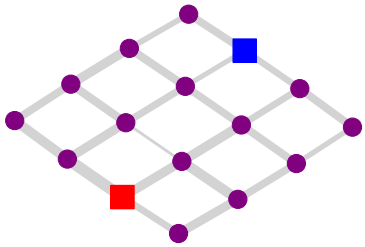
(b) Instancia 2



(c) Instancia 3



(d) Instancia 4

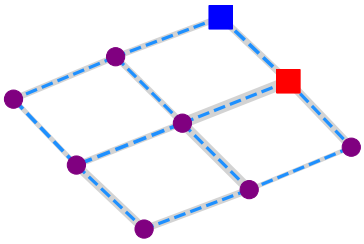


(e) Instancia 5

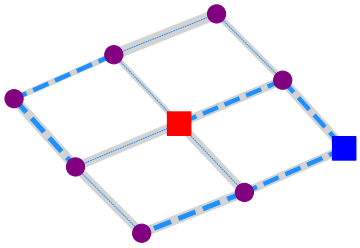
Figura 2: *Instancias con capacidad*

Por último, se representa en la figura 3 el flujo correspondiente a cada arista de color celeste, manteniendo la idea de que el grosor representa la cantidad de flujo que pasa por cada una de ellas.

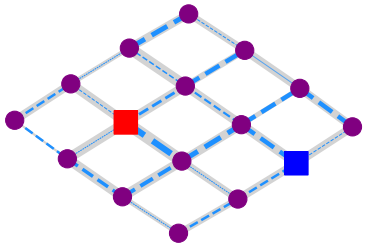
```
        flow_value, flow_dict = nx.maximum_flow(self.G, self.inicio, self.final, capacity='weight')
        self.objetivo=flow_value
        print(self.objetivo)
        #Aqui indica el flujo que pasa por cada arista
        flujo=[]
        pesos=nx.get_edge_attributes(self.G,'weight')
        pesos1=[pesos[i] for i in pesos]

        for i in flow_dict.keys():
            for j in flow_dict[i].keys():
                if flow_dict[i][j]>0:
                    flujo.append(flow_dict[i][j])

        self.grosor_capacidad=[i*0.75 for i in pesos1]
        self.grosor_flujo=[i*0.4 for i in flujo]

        nx.draw_networkx_nodes(self.G,self.pos,node_color='purple')
        nx.draw_networkx_nodes(self.G,self.pos,nodelist=[(origen1,origen2)],node_color='b', node_shape='s',node_size=500)
        nx.draw_networkx_nodes(self.G,self.pos,nodelist=[(destino1,destino2)],node_color='r', node_shape='s',node_size=500)
        nx.draw_networkx_edges(self.G,self.pos,edge_color='lightgray',width=self.grosor_capacidad)
        plt.axis('off')
        plot_margin = 0.05
        x0, x1, y0, y1 = plt.axis()
        plt.axis((x0 - plot_margin,x1 + plot_margin,y0 - plot_margin,y1 + plot_margin))
        plt.tight_layout()
        imagen2="I2_"+str(self.num)+".eps"
        plt.savefig(imagen2)
        plt.show()

        nx.draw_networkx_nodes(self.G,self.pos,node_color='purple')
        nx.draw_networkx_nodes(self.G,self.pos,nodelist=[(origen1,origen2)],node_color='b', node_shape='s',node_size=500)
        nx.draw_networkx_nodes(self.G,self.pos,nodelist=[(destino1,destino2)],node_color='r', node_shape='s',node_size=500)
        nx.draw_networkx_edges(self.G,self.pos,edge_color='lightgray',width=self.grosor_capacidad)
        nx.draw_networkx_edges(self.G,self.pos,edge_color='dodgerblue',width=self.grosor_flujo, style='--')
```



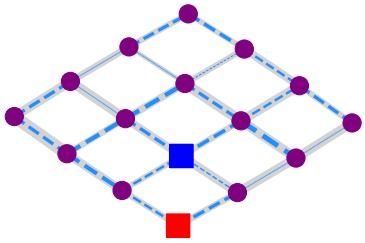
(a) Instancia 1



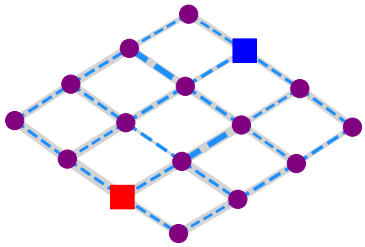
(b) Instancia 2



(c) Instancia 3



(d) Instancia 4



(e) Instancia 5

Figura 3: *Instancias con solución*

2. Características estructurales de los vértices

Distribución de grado

Dado un grafo de tamaño n, el grado de un vértice i se define como el número total de aristas que inciden en él, mientras que la distribución de grado se define como la probabilidad de que un vértice i tenga exactamente k conexiones con otros vértices.

`Degree_centrality`. Calcula la centralidad de grado para los vértices y recibe como único parámetro un grafo.

```
1        A1 = nx.degree_centrality(self.G)
```

Coeficiente de agrupamiento

El coeficiente de agrupamiento mide el nivel de agrupamiento que existe entorno a un vértice, que se calcula como el número total de aristas que conectan a los vecinos más cercanos entre el número máximo de aristas posibles entre todos los vecinos más cercanos. Se puede ver como la probabilidad de que dos vértices vecinos a un vértice i sean vecinos entre sí.

`Clustering`. Este algoritmo calcula el coeficiente de agrupamiento de los vértices y recibe como parámetros un grafo, lista de nodos para los que se desea calcular el coeficiente y peso de las aristas. Estos últimos dos parámetros son opcionales por lo que de no ser incluidos se toman por defecto todos los vértices del grafo y cada arista recibe un peso de uno.

```
1        A3 = nx.clustering(self.G)
```

Centralidad de cercanía

La centralidad de cercanía indica que tan cerca está un vértice i del resto de los vértices del grafo. Se calcula como el promedio de las distancias más cortas desde el vértice i a los demás vértices. En algunos casos, el recíproco de la distancia promedio se utiliza como la medida de centralidad de cercanía para evitar problemas con las distancias en los grafos no conexos y en estos casos los valores más altos indican una centralidad más alta.

`Closeness_centrality`. Este algoritmo calcula la centralidad de cercanía de los vértices, tomando el recíproco del promedio de las distancias más cortas y

normalizando. Recibe como parámetros un grafo, el vértice para el que se quiere encontrar el valor si se desea para uno en específico, la distancia y la opción de normalizar los coeficientes.

```
1        A2 = nx.closeness_centrality(self.G)
```

Centralidad de carga

La centralidad de carga de un vértice i se define como la fracción de todos los caminos más cortos que pasan a través de ese vértice.

`Load_centrality`. Recibe como parámetro principal un grafo y parámetros opcionales la normalización de los valores, el peso y el corte que si se especifica solo considera rutas de longitud menor o igual al corte.

```
1        A4 = nx.load_centrality(self.G)
```

Excentricidad

Se define como la distancia máxima desde un vértice i a a todos los demás vértices del grafo.

`Eccentricity`. Retorna la excentricidad de los nodos de un grafo y recibe como parámetros un grafo y un vértice en particular para el que se desea calcular la excentricidad.

```
1        A5 = nx.eccentricity(self.G)
```

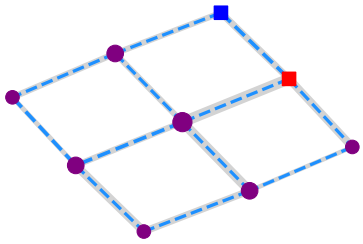
PageRank

Explora e indica la categorización de vértices en redes con base en su importancia.

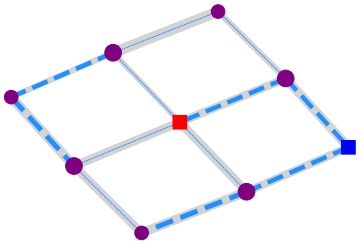
`Pagerank`. Retorna el PageRank de los vértices, donde se calcula una clasificación de los vértices de un grafo en función de la estructura de las conexiones entrantes. Recibe como parámetros un grafo, parametro de atenuación, un número máximo de iteraciones para resolver valores propios, toleancia de error, valor de inicio de la iteración para cada vértice y el peso de cada arista.

```
1        A6 = nx.pagerank(self.G)
```

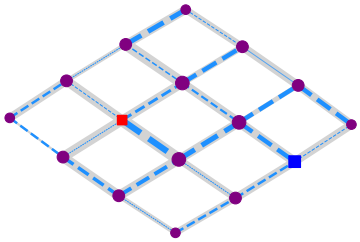
En la figura 4 se incluyen las visualizaciones de las instancias añadiendo la variación de tamaño de los vértices con las características antes mencionadas, para determinar cuales resultan ser relevantes. Se puede observar que para instancia los vértices de las esquinas tienen menor tamaño a comparación de los demás, debido posiblemente a la estructura que se tiene con el generador, las capacidadades y el flujo.



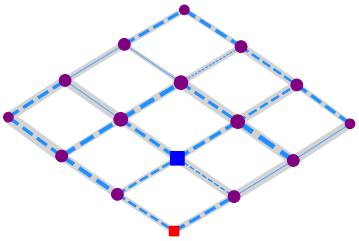
(a) Instancia 1



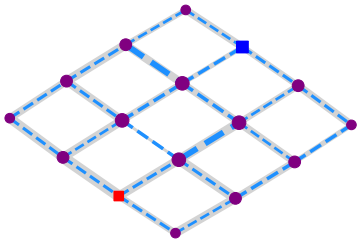
(b) Instancia 2



(c) Instancia 3



(d) Instancia 4



(e) Instancia 5

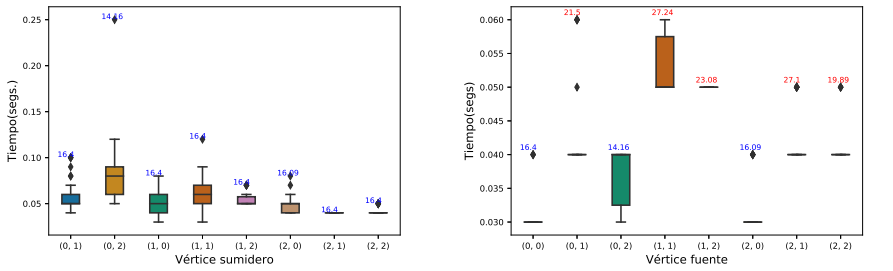
Figura 4: *Instancias con caracterización*

3. Resultados

A continuación se presenta la comparación del tiempo de ejecución variando el vértice fuente y sumidero mediante diagramas de caja y bigote, identificando con rojo los casos en los que el valor de la función objetivo mejora y con azul los que no tienen mejora o empeoran.

El valor de la función objetivo de la instancia 1 es 16.4 y en la figura 5 se observa que al variar el vértice sumidero no se encuentran mejoras, por lo que si busca es tener un tiempo menor, se pueden elegir los valores con el mismo valor objetivo que tardan menos, como lo son el (2,1) y el (2,2).

Por otra parte, al variar el vértice fuente, los vértices (0,1), (1,1), (1,2), (2,1), (2,2) presentan una mejora, siendo el (1,1) el que tiene mayor valor de la función objetivo pero que también consume mucho tiempo a comparación de los otros, mientras el vértice (2,1) presenta un valor objetivo similar con un tiempo de ejecución menor.

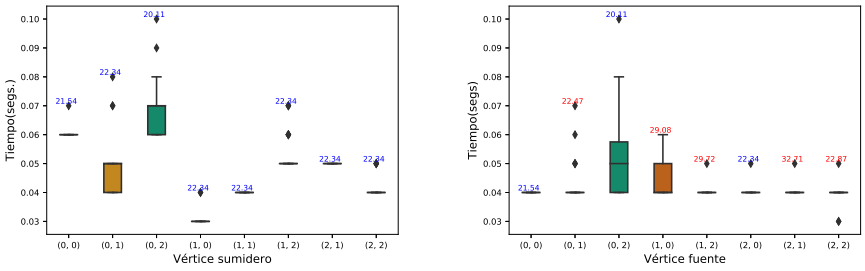


(a) Tiempo de ejecución variando sumidero (b) Tiempo de ejecución variando fuente

Figura 5: *Instancia 1*

El valor objetivo de la instancia 2 es 22.34 y en la figura 6 se observa que al variar el vértice sumidero no se encuentran mejoras, pero si lo que se busca es conservar el valor de la función objetivo con menor tiempo, se consideraría el vértice (1,0).

Por otra parte, al variar el vértice fuente, los vértices que presentan mejora son el (0,1), (1,0), (1,2), (2,1) y (2,2), de los cuales el de mayor valor objetivo y menor tiempo de ejecución es el nodo (2,1).

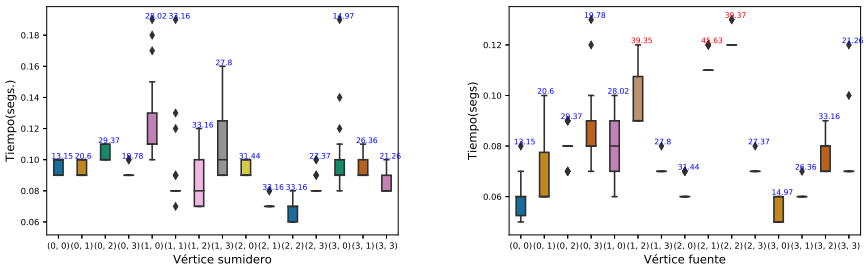


(a) Tiempo de ejecución variando sumidero (b) Tiempo de ejecución variando fuente

Figura 6: *Instancia 2*

El valor objetivo de la instancia 3 es 33.16 y en la figura 7 se observa que al variar el vértice sumidero no se encuentran mejoras, pero si lo que se busca es conservar el valor de la función objetivo con menor tiempo se considera elegir el vértice (2,2).

Por otra parte, los vértices que presentan mejora son el (1,2), (2,1) y (2,2), de los cuales el vértice con mayor valor objetivo es el (2,1) pero con la desventaja de que consume mucho tiempo, mientras que el de menor tiempo es el (1,2).



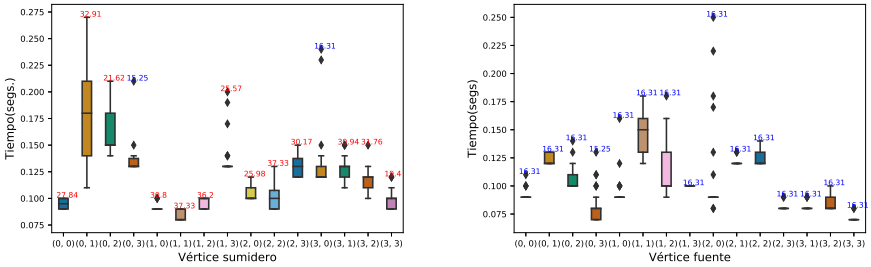
(a) Tiempo de ejecución variando sumidero (b) Tiempo de ejecución variando fuente

Figura 7: *Instancia 3*

El valor objetivo de la instancia 4 es 16.31 y en la figura 8 se muestra que hay mejoras en el valor objetivo de la instancia variando el vértice sumidero original, siendo el (1,1) el de mayor valor objetivo y menor tiempo de ejecución.

Por otra parte, se observa que al variar el vértice fuente no se encuentran mejo-

ras, pero si lo que se busca es conservar el valor objetivo con menor tiempo se considera elegir el vértice (3,3).

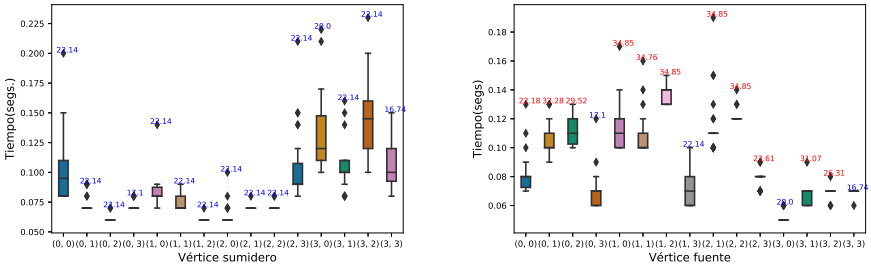


(a) Tiempo de ejecución variando sumidero (b) Tiempo de ejecución variando fuente

Figura 8: *Instancia 4*

El valor objetivo de la instancia 5 es 22.14 y en la figura 9 se observa que al variar el vértice sumidero no se encuentran mejoras, pero si lo que se busca es conservar el valor objetivo con menor tiempo se considera elegir el vértice (0,2) y (1,2).

Por otra parte los vértices que no presentan mejora son el (0,3), (1,3), (3,0) y (3,3), siendo el (1,0) y (1,1) los que presentan mayor valor objetivo y menor tiempo de ejecución .

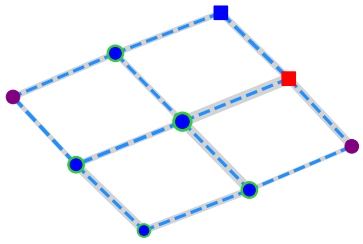


(a) Tiempo de ejecución variando sumidero (b) Tiempo de ejecución variando fuente

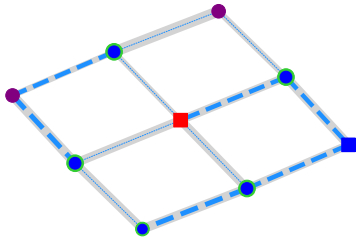
Figura 9: *Instancia 5*

En la figura 10 se visualiza cada una las caracterizaciones en la instancia y donde si se tiene un buen vértice fuente pero no sumidero se representa con

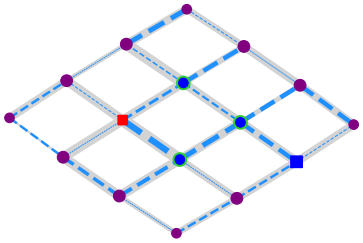
azul y verde limón, aquellos que son mejor vértice sumidero pero no fuente se representa con color rojo y amarillo mientras que los que son mejor fuente y sumidero se representan con el color verde oscuro. Para las instancias 1, 2 y 5 se observa que tienen vértices que son buena opción para considerar como vértices fuente si se quiere mejorar el valor de la función objetivo, mientras que la instancia 2 se tiene que vértices son buena opción como vértices sumidero.



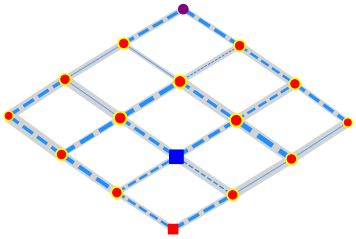
(a) Instancia 1



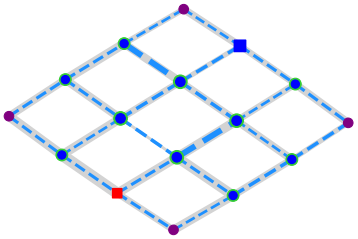
(b) Instancia 2



(c) Instancia 3



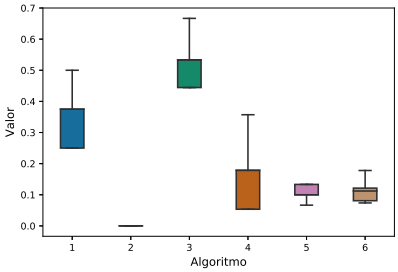
(d) Instancia 4



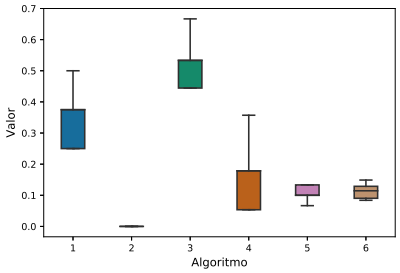
(e) Instancia 5

Figura 10: *Instancias con caracterización*

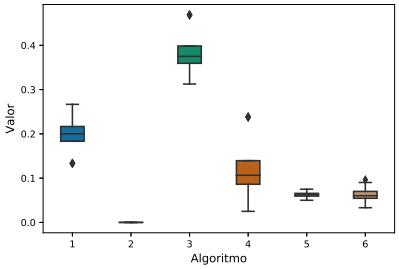
Por último, en la figura 11 se comparan los algoritmos `degree_centrality`, `clustering`, `closeness_centrality`, `load_centrality`, `eccentricity` y `pagerank` para cada instancia, esperando determinar que características pudieran ser relevantes para la solución.



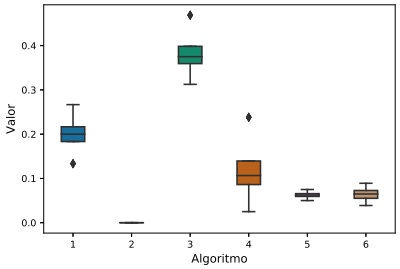
(a) Instancia 1



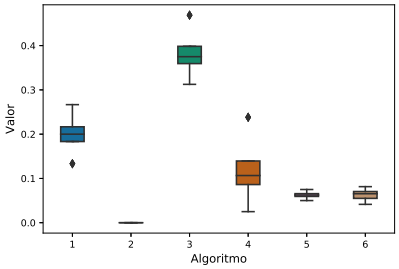
(b) Instancia 2



(c) Instancia 3



(d) Instancia 4



(e) Instancia 5

Figura 11: *Valores obtenidos con los algoritmos*

De acuerdo a los resultados obtenidos de los diagramas de caja y bigote se

obtienen las siguientes conclusiones generales:

1. Se observa que el `closeness_centrality` es el que tiene valores más altos, seguido por `degree_centrality`, `load_centrality`, `pagerank`, `eccentricity` y `clustering` teniendo valor cero en todas las instancias.
2. El algoritmo generador se comporta muy similar sin importar la cantidad de vértices y siempre se obtienen valores similares de las características, por lo que podría decirse que el valor de la función objetivo no depende de las características sino de una buena elección del vértice fuente y sumidero.
3. Tener un vértice fuente en las orillas y un vértice sumidero en el centro asi como ambos en el centro, hace que se obtenga un mejor valor objetivo, ya que los vértices del centro parecen tener mejores valores de las características que los vértices de las orillas y esquinas tal y como se muestra en la figura 10.

Referencias

[1] NetworkX developers Versión 2.0. `https://networkx.github.io/`.

[2] The Matplotlib development team Versión 3.0.2. `https://matplotlib.org/`.

[3] Python Software Foundation Versión 3.7.2. `https://www.python.org/`.

[4] NumPy developers Versión 1.16.2. `https://networkx.github.io`.

[5] Augspurger T. and the pandas core team Versión 0.24.2. `https://pandas.pydata.org/`.

[6] NetworkX developers con última actualización el 19 de Septiembre 2018. `https://networkx.github.io/documentation/networkx-1.10/reference/drawing.html`.

[7] Oscar Tomas Vegas Niño and Velitchko Martínez Alzamora, Fernando y Tzatchkov. Aplicación de la teoría de grafos a la identificación de subsistemas hidráulicos en redes de distribución de agua. 09 2016.

[8] Sotelo B. Repositorio optimización flujo en redes. `https://github.com/BrendaSotelo/Flujo_Redes_BSotelo`.