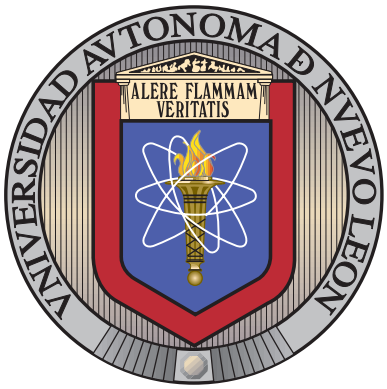
Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

División de Estudios de Posgrado



Detección de fugas en los procesos de una refinería

por

Ing. Luis Alejandro Benavides Vázquez

en opción al grado de

Maestría en Ciencias

en Ingeniería de Sistemas

San Nicolás de los Garza, Nuevo León febrero 2014

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

División de Estudios de Posgrado

Detección de fugas en los procesos de una refinería

por

Ing. Luis Alejandro Benavides Vázquez

en opción al grado de

Maestría en Ciencias

en Ingeniería de Sistemas

San Nicolás de los Garza, Nuevo León febrero 2014

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

División de Estudios de Posgrado

Los miembros del Comité de Tesis recomendamos que la Tesis «Detección de fugas en los procesos de una refinería», realizada por el Ing. Luis Alejandro Benavides Vázquez, con número de matrícula 1373078, sea aceptada para su defensa como opción al grado de Maestría en Ciencias en Ingeniería de Sistemas.

El Comité de Tesis

Dra. Yasmín A. Ríos Solís
Directora

Dr. Igor S. Litvinchev
Revisor

Dr. Mauricio Cabrera Ríos
Revisor

Vo. Bo.

Dr. Moisés Hinojosa Rivera
División de Estudios de Posgrado

San Nicolás de los Garza, Nuevo León, febrero 2014

A mi familia

Índice general

Agradecimientos

Deseo agradecer a la Universidad Autónoma de Nuevo León (U.A.N.L) la oportunidad que me brindó para realizar mis estudios.

A la Facultad de Ingeniería Mecánica y Eléctrica (F.I.M.E) por el apoyo brindado durante mis estudios de maestría.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico otorgado mediante una beca de estudios de tiempo completo.

Quedo agradecido al Posgrado de Maestría en Ingeniería de Sistemas (PISIS) por darme la oportunidadd de realizar mis estudios de maestría.

Quiero expresar mi agradecimiento a la Dra. Yasmín Águeda Ríos Solís, directora de mi tesis por haberme guiado durante todo este tiempo, también por darme consejos para mejorar en mis presentaciones y escritura los cuales me han ayudado a mejorar en la manera de exponer. Por brindarme su apoyo y su amistad.

Agradezco al Dr. Mauricio Cabrera Ríos por aceptarme en el grupo de investigación durante mi estancia en el Recinto Universitario de Mayagüez, Puerto Rico, y guiarme durante la misma además de brindarme su apoyo y su amistad. Agradezco de igual manera a los compañeros del Applied Optimization Group de Puerto Rico quienes me acogieron como un miembro más, a parte de darme consejos y apoyo. Especialmente las alumnas que se encuentran bajo la tutela del Dr. Mauricio Cabrera Ríos, con las cuales estuve compartiendo mi estancia de Investigación.

Agradezco al Dr. Igor Litivinchev por ser parte de mi comité de tesis, guiarme

como ayuda en la elaboración de la misma.

Deseo agradecer a todos los profesores y compañeros del PISIS por compartir durante la maestría ayudaron a seg. En especial a los compañeros de la generación que entramos en el enero del año 2012.

Agradezco profundamente a mi familia, mis padres Bertha Alicia y José Loreto, así como a mi hermano José Alberto, que siempre estuvieron para apoyarme en mi transcurso de mis estudios de maestría tanto en el ámbito personal y profesional, con sus consejos, siempre motivándome para enfrentar las adversidades que se presenten y esforzándose para sacar lo mejor de mi con el fin de mejorarme en cada situación que se me presenta. Gracias a ustedes he podido completar esta meta y podré completar cualquier cosa que me proponga .

Agradezco de todo corazón a mi novia Margarita Cadena por comprenderme y estar siempre a mi lado dándome el apoyo cuando más lo necesitaba, aconsejándome para mejorar y esforzarme en lo que me proponga para hacerlo siempre lo mejor posible. Me has ayudado mucho en la etapa de posgrado y en mi crecimiento personal durante esta etapa. Juntos lograremos todo lo que nos propongamos, somos un equipo.

Resumen

En esta tesis se quieren detectar pérdidas de material en las refinerías de petróleo. El proceso llevado a cabo en las refinerías es complejo ya que involucra una serie de procesos químicos para hacer decenas de productos finales diferentes. Actualmente, en México se tienen instalados medidores de flujo donde el más usado es el medidor de presión diferencial del tipo de placa y orificio. El mayor problema es la medida de incertidumbre asociada a estos instrumentos haciendo que esto produzca falsas alarmas.

Se propone representar la refinería como una red de transporte basado en flujo de redes. Para poder simular una refinería real, se utilizan las siguientes herramientas:

- Generación de base datos de los flujos de masa en los tramos de tubería entre procesos para generar el flujo en ellos.
- Modificación al algoritmo de flujo máximo (Ford-Fulkerson).
- Ecuación característica de medidores de placa y orificio.
- Cálculo y simulación de propagación de errores en la red de transporte.

Al generar la base de datos del comportamiento de la refinería, procedemos a clasificar los flujos de masa en todos los tramos de tubería en la refinería usando redes neuronales artificiales.

Para validar el método propuesto utilizamos cuatro conjuntos de flujos de masa para cada tramo de tubería, generados en la red propuesta con su correspondiente

detección de pérdidas. Cada conjunto consta de diecisiete arcos. El tamaño de cada conjunto va en aumento, esto quiere decir que el primer conjunto tiene menos datos que el ultimo. Así, el primer conjunto tiene 100 datos, el segundo 232 datos, el tercer 332 datos y el cuarto 1000 datos. Cada conjunto se utiliza para entrenar la red neuronal artificial y clasificar posteriormente 1000 valores generados aleatoriamente.

Ing. Luis Alejandro Benavides Vázquez. Candidato para el grado de Maestría en Ciencias

en Ingeniería de Sistemas.

Universidad Autónoma de Nuevo León.
Facultad de Ingeniería Mecánica y Eléctrica.

Título del estudio:

Detección de fugas en los procesos de una refinería

Número de páginas: 129.

Objetivos y método de estudio: Detectar confiablemente pérdidas de material en transporte entre procesos de una refinería. Para este fin proponemos crear una herramienta computacional basada principalmente en teoría de grafos para efectos de modelación y optimización, así como en redes neuronales artificiales para clasificación eficiente de fugas.

Contribuciones y conclusiones: *Contribuciones*: Como resultado de esta tesis se desarrollaron dos herramientas como ayuda en la detección de pérdidas en refinerías de productos refinados. La primer herramienta nos permite hacer una simulación simplificada de una refinería en la cual se generan fugas aleatoriamente en los arcos de la red y nos ayuda además a tener una detección preliminar de las fugas. La segunda herramienta se utiliza para clasificación en donde se obtuvieron

buenos resultados para la detección de pérdidas en el caso de estudio de la refinería en estado estable.

Conclusiones: Comprobamos que es posible detectar pérdidas en el proceso de refinamiento del petróleo usando modelos de flujo en redes que incorporan datos de balance de masa para obtener una clasificación suficientemente precisa mediante redes neuronales. Además, como es de esperar, al utilizar mas datos en el entrenamiento de la red neuronal se obtienen mejores clasificaciones.

Firma de la directora: _____.

Dra. Yasmín A. Ríos Solís

Capítulo 1

Introducción

1.1 Descripción del Problema de Fugas en una Refinería

Durante las últimas décadas, el petróleo ha sido primordial en la economía mundial contribuyendo al desarrollo socioeconómico y tecnológico de distintas culturas y países, ya que tiene una gran aplicación a través de sus derivados en diversas áreas de interés como la automotriz, farmacéutica y en la de los plásticos entre otras (Barberii, 1998). Por este motivo es importante prestarle atención al proceso de producción del petróleo, desde su extracción hasta la obtención de sus derivados, ya que durante el proceso se pueden ocasionar pérdidas de material que afectan al medio ambiente causando daños irreversibles, además de generar grandes pérdidas económicas.

La infraestructura de Pemex Refinación ha permanecido sin cambios durante casi dos décadas, frente a una demanda interna de petrolíferos que aumenta a tasas más elevadas que la economía mundial (Mexicana, 2010). Al cierre del 2008 Pemex Refinación alcanzó $ 547,000,000 a $ 548,000,000 pesos por ventas totales de productos petrolíferos. El Sistema Nacional de Refinación (SNR) de Pemex tenía, en el 2010, una capacidad de procesamiento de crudo de un 1,540,000 barriles por día, con seis refinerías en las siguientes localidades, Cadereyta, Nuevo León; Ciudad Madero, Tamaulipas; Minatitlán, Veracruz; Salamanca, Guanajuato; Salina Cruz,

Oaxaca; y Tula, Hidalgo. Cada una de estas refinerías cuenta con una capacidad de procesamiento que se muestra en la Tabla 1.1.

Localidad	MBD[1]
Cadereyta, Nuevo León	208.35
Ciudad Madero, Tamaulipas	152.09
Minatitlán, Veracruz	161.58
Salamanca, Guanajuato	192.48
Salina Cruz, Oaxaca	279.36
Tula, Hidalgo	267.19
Total	1,261.05

Tabla 1.1: Proceso de petróleo crudo por refinería en el 2008. (Mexicana, 2010).
[1] Millones de Barriles Diarios

De acuerdo con la Tabla 1.1 la producción en el 2008 es de 1,261.05 MBD lo cual generó una ganancia de $547,000,000 a $548,000,000 pesos por ventas totales de productos petrolíferos (Mexicana, 2010). Esto nos indica que cualquier pérdida que pueda aparecer en los procesos de producción del petróleo y sus derivados debe ser detectada rápidamente con el fin de obtener una mayor producción y por ende una mayor ganancia. Por lo que es necesario prestarle atención a cada uno de los procesos involucrados en la refinación del petróleo, pero en especial al proceso de refinación enfocado en esta tesis, ya que a partir de la refinación se consiguen los derivados del petróleo, los cuales tienen un gran impacto en una gran cantidad de industrias que tienen como materia prima alguno de los derivados del petróleo.

El problema de pérdidas no ocurre solamente en las industrias del petróleo sino también en industrias en donde se transporta material por tuberías que pueden tener grietas o simplemente que no tengan un funcionamiento correcto de un equipo y esto genere una pérdida de material. Aquí es donde en todas estas industrias, incluida la del petróleo, requieren tener una herramienta que consista en detectar las pérdidas

de productos con el fin de prevenir que se pierda material y que se afecte al medio ambiente.

Este trabajo se enfoca principalmente en el proceso de refinado el cual es complejo ya que involucra una serie de procesos químicos para hacer decenas de productos finales diferentes. En México se tienen instalados medidores de flujo y el más usado es el medidor de placa y orificio, un medidor de presión diferencial. El mayor problema es la medida de incertidumbre asociada a estos instrumentos (Preparedness y Program, 1999), haciendo que esto produzca falsas alarmas, por lo que hay que tenerlo en cuenta al realizar los cálculos.

Industrias petrolíferas mexicanas, expertas del proceso, proponen que las pérdidas se deban a la diferencia de entradas con salidas, a los inventarios y restando a esto el autoconsumo generado por la refinería (CONACYT-SENER-Hidrocarburos, 2012).

Proponemos representar la refinería como una red de transporte basándonos en flujo de redes. Para poder simular una refinería real, utilizamos las siguientes herramientas:

- Modificación al algoritmo de flujo máximo (Ford-Fulkerson).
- Ecuación característica de los medidores usados en la refinería (tipo placa y orificio).
- Cálculo de propagación de errores en la red de transporte.
- Generación de flujos de masa en los tramos de tubería de la refinería en todas las condiciones posibles, sin pérdidas y con pérdidas desde un tramo hasta todos los tramos de la tubería. Esto se almacena en una base de datos.

Al generar una base de datos del comportamiento de la refinería, se proceden a clasificar los flujos de masa en todos los tramos de tubería en la refinería usando redes neuronales artificiales

Para validar el método propuesto se utilizan cuatro conjuntos de flujos de masa para cada tramo de tubería, generados en la red propuesta con su correspondiente detección de pérdidas. Cada conjunto consta de diecisiete arcos. El tamaño de cada conjunto va en aumento, esto quiere decir que el primer conjunto tiene menos datos que el último. Así, el primer conjunto tiene 100 datos, el segundo 232 datos, el tercer 332 datos y el cuarto 1000 datos. Cada conjunto se utiliza para entrenar la red neuronal artificial para clasificar posteriormente 1000 valores de prueba generados aleatoriamente.

1.2 Hipótesis

Habiendo descrito la problemática que existe en las refinerías de México proponemos la siguiente hipótesis: Se pueden detectar fugas en procesos de refinamiento del petróleo usando modelos de flujo en redes que incorporan datos de balance de masa para obtener una clasificación precisa mediante redes neuronales.

1.3 Motivación y Justificación

Las razones por la cual nos motivamos a realizar este proyecto en la presente tesis es que nos damos cuenta que actualmente en las refinerías de México existen problemas reales de pérdidas de material que deben ser tratados para evitar grandes pérdidas económicas y daños al medio ambiente. Se sabe que hay al menos 5 % de pérdidas de material en el proceso de refinación. Esto lo sabemos porque en diciembre de 2012 se convocó una Demanda Específica de (CONACYT-SENER-Hidrocarburos, 2012) que lleva por nombre: *Implementación de una plataforma de medición y balance másico reconciliado de petrolíferos en la refinería de Tula.* El objetivo principal de dicha demanda es el desarrollo de una plataforma que permita conocer las pérdidas que se están produciendo en la refinería de Tula, y una vez que se tenga la plataforma poder implementarla en otras refinerías de México.

Consideramos que el uso del método propuesto será de gran utilidad para la solución de este problema debido a su práctica implementación y aplicación.

1.4 Objetivo

El objetivo principal de la tesis es el de detectar confiablemente pérdidas de material en transporte entre procesos de una refinería y para este fin se propone crear una herramienta computacional basada principalmente en teoría de grafos para efectos de modelación y optimización, así como en redes neuronales artificiales para clasificación eficiente.

1.5 Estructura de la Tesis

En los siguientes cinco capítulos, presentamos las bases teóricas de las herramientas utilizadas para la elaboración de la solución al problema presentado en la tesis, además de la descripción de la metodología utilizada. En el Capítulo 2 presentamos el marco teórico en donde veremos el funcionamiento de la refinería, conociendo sus procesos principales y los equipos característicos de cada proceso. Además, el concepto de teoría de grafos utilizado con el fin de llevar a cabo la modelación y visualización del problema. Como se mencionó anteriormente, en México se tienen medidores del tipo de placa y orificio por lo cual se incluye el funcionamiento de éstos y algunas sugerencias de la instalación de medidores en las refinerías. También se da una breve introducción a redes neuronales artificiales que nos ayudan a llevar a cabo la metodología y dar solución al problema presentado. Además, presentamos los antecedentes al problema de detección de pérdidas en productos refinados. En el Capítulo 3 describimos la generación de instancias que nos permite obtener una base de datos tomando el comportamiento de la refinería. En el Capítulo 4 presentamos la descripción de la red neuronal artificial, concepto que utilizamos como ayuda para realizar la clasificación de los flujos de masa obtenidos por los medido-

res. En el Capítulo 5 presentamos los resultados obtenidos de la experimentación para ambas herramientas computacionales desarrolladas. Por último, en el Capítulo 6 presentamos las conclusiones finales de la tesis, así como las contribuciones aportadas al problema de detección de pérdidas y el trabajo futuro para el problema. El código utilizado para el generador de instancias se encuentra en el Apéndice B y el código con el que se trataron la base de datos generada mediante redes neuronales artificiales se encuentra en el Apéndice C.

Capítulo 2

Marco Teórico

En este capítulo se habla primeramente de las definiciones correspondientes con teoría de grafos ya que se utilizan en la Sección 2.2 y en el Capítulo 3 para representar los procesos de la refinería sirviendo de ayuda para la visualización y modelación del problema. Después, en la Sección 2.2 se presenta el funcionamiento de una refinería, en donde vemos lo complejo que es el proceso de refinación por involucrar un gran número de subprocesos y equipos diferentes para producir más de una docena de derivados del petróleo. Otro tema que se incluye en la Sección 2.3 es el funcionamiento de los medidores más usados en la industria petrolera mexicana para medición de flujo, los cuales son del tipo placa y orificio. Además, en la Sección 2.4 se da un breve resumen de las redes neuronales artificiales que se utilizan como ayuda para la metodología de solución del problema mostrado en esta tesis en el Capítulo 4. Al terminar de presentar la teoría requerida para entender los conceptos utilizados para dar solución al problema de detección pérdidas en refinerías, en la Sección 2.5 se presentan los antecedentes a dicho problema, ya sea de la industria petrolera o de cualquier otra industria en donde haya transporte de material.

2.1 Teoría de Grafos

En esta sección se habla del concepto de teoría de grafos, usado para representar diferentes problemas que se tienen en la vida cotidiana o en las industrias de las diversas áreas existentes, mediante conceptos que nos permiten un mejor en-

tendimiento del problema y simplificación visual. En el Capítulo 3 representamos la refinería de estudio mediante el uso de las definiciones expuestas en ésta sección. Definiremos los conceptos de teoría de grafos que nos sirven de ayuda para esta tesis. Existen una gran cantidad de definiciones, solo nos enfocaremos a las usadas en esta tesis. Para más información de teoría de grafos ir a los libros: *Graph Theory* (Diestel, 2005), *Graph Theory With Applications* (Bondy y Murty, 1976), *Matemáticas Discretas y sus Aplicaciones* (Rosen, 2004), *Linear Programming and Network Flows* (Bazaraa et al., 2010) o *Introduction to Graph Theory* (West, 2000).

A continuación se muestran las definiciones de teoría de grafos, que utilizamos en la tesis, según el libro de *Matemáticas Discretas y sus Aplicaciones* (Rosen, 2004).

2.1.1 Definiciones

Un grafo simple G, se expresa como $G(N, A)$, consta de N, un conjunto no vacío de vértices o nodos, y de aristas o arcos A, un conjunto de pares no ordenados de elementos distintos de V.

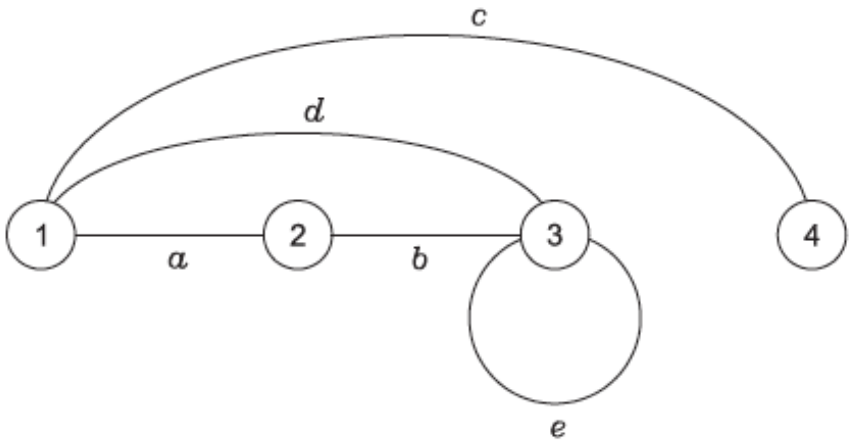


Figura 2.1: Grafo.

La Figura 2.1 corresponde al grafo G en donde $V = 1, 2, 3, 4, A = a, b, c, d, e$ Un camino en un grafo $G(V, A)$ es una sucesión (posiblemente vacía) $a = a_1, ..., a_n$

de aristas tal que existe una sucesión $p = p_0, ..., p_n$ de nodos tal que para cada $a_i, P(a_i) = p_{i-1}, p_i$. La sucesión p se llama *recorrido* del camino n. Se dice que a empieza en p_0 y termina en p_n.

No cualquier sucesión de aristas es un camino, porque puede ser que no exista recorrido. Por ejemplo, en la Figura 2.1 la sucesión c, b no es un camino. La sucesión a, b sí es, y su recorrido es 1, 2, 3.

Los nodos p, q de un grafo son conexos, o p es conexo con q, si hay un camino que empieza en p y termina en q. Un grafo es conexo si cualquier par de sus nodos son conexos.

Se quiere que el grafo sea conexo para tener sentido físico al momento de representar la refinería, como se verá en el Capitulo 3. Al no ser conexo representa que algún proceso se encuentra fuera de las instalaciones de la refinería y no tiene conexión con el proceso.

2.1.2 Redes de Transporte

Dado un grafo, si ponemos una flecha en cada arista tenemos lo que se llama un *grafo dirigido*. En el problema que vamos a estudiar, no hace falta considerar la posibilidad de tener dos aristas de un nodo p a un nodo q así que podemos simplificar la notación designando cada arista con nodo origen p y nodo destino q por (p, q). También excluimos la posibilidad de una arista (p, p), como en la Figura 2.1. Entonces podemos nombrar un camino con solo nombrar su recorrido.

Una red de transporte es un grafo dirigido conexo con una función w de las aristas a los números reales no negativos.

La red representa capacidad de flujo de algo - mensajes, petróleo, tráfico vehícular - y se quiere calcular el flujo máximo de que es capaz, dado que el flujo en cada arista (p, q) no puede exceder a su capacidad $w(p, q)$, y que todo el flujo entrando en un nodo tiene que ser igual en cantidad a todo el flujo que sale del nodo, excep-

tuando la fuente s y el sumidero t. Se considera que la fuente produce el flujo que se transporta y el sumidero lo recibe.

Este problema puede ser resuelto mediante diferentes algoritmos, pero nos centraremos en el más utilizado, el Algoritmo de Ford-Fulkerson también llamado Algoritmo de Flujo Máximo, presentado en la Sección 2.1.3 del Capítulo 2. En el Capítulo 3 se usa para resolver el problema de los diferentes caminos que llevan del proceso inicial del transporte hasta el almacenamiento de los productos derivados.

Si se quiere saber más acerca de algoritmos de flujo en redes se recomiendan las siguientes lecturas: *Linear Programming and Network Flows* (Bazaraa et al., 2010), *Network Flows: Theory, Algorithms, and Applications* (Ahuja et al., 1993), *Matemáticas Discretas y sus Aplicaciones* (Rosen, 2004).

2.1.3 Algoritmo Ford-Fulkerson

El algoritmo de Ford-Fulkerson propone buscar caminos en los que se pueda aumentar el flujo, hasta que se alcance el flujo máximo. La idea es encontrar una **ruta de penetración** con un flujo positivo neto que una los nodos origen y destino. Consideraremos las capacidades iniciales del arco que une el nodo i y el nodo j como C_{ij} y C_{ji}. Estas capacidades iniciales irán variando a medida que avanza el algoritmo, denominaremos **capacidades residuales** a las capacidades restantes del arco una vez pasa algún flujo por él, las representaremos como c_{ij} y c_{ji}. Para un nodo j que recibe el flujo del nodo i, definimos una clasificación $[a_j, i]$ donde a_j es el flujo del nodo i al nodo j. Los pasos del algoritmo se definen como sigue:

1. Inicializar las capacidades residuales a las capacidades iniciales, hacemos $(c_{ij}, c_{ji}) = (C_{ij}, C_{ji})$ para todo arco de la red. Suponiendo el nodo 1 como el nodo origen, hacemos $a_1 = \infty$ y clasificamos el nodo origen con $[\infty, -]$. Tomamos $i = 1$ y vamos al paso 2.

2. Determinar S_i como un conjunto que contendrá los nodos a los que podemos

acceder directamente desde i por medio de un arco con capacidad positiva, y que no formen parte del camino en curso. Si contiene algún nodo vamos al paso 3, en el caso de que el conjunto sea vacío saltamos al paso 4.

3. Obtener $k \in S_i$ como el nodo destino del arco de mayor capacidad que salga de i hacia un nodo perteneciente a S_i. Es decir, $c_{ik} = max\{c_{ij}\}$ con $j \in S_i$. Hacer $a_k = c_{ik}$ y clasificar el nodo k con $[a_k, i]$. Si k es igual al nodo destino o sumidero, entonces hemos encontrado una ruta de penetración, vamos al paso 5. En caso contrario continuamos con el camino, hacemos $i = k$ y volvemos al paso 2.

4. Si $i = 1$, estamos en el nodo origen, y como S_i es vacío, entonces no podemos acceder a ningún nodo, ni encontrar algún nuevo camino, hemos terminado, vamos al paso 6. En caso contrario, $i \neq 1$, le damos al valor i el del nodo que se ha clasificado inmediatamente antes, eliminamos i del conjunto S_i actual y volvemos al paso 2 (Retroceso).

5. Llegados a este paso tenemos un nuevo camino: $N_p = 1, k_1, k_2, ..., n$, esta será la p-ésima ruta de penetración desde el nodo origen al nodo destino. El flujo máximo a lo largo de esta ruta será la capacidad mínima de las capacidades residuales de los arcos que forman el camino, es decir: $f_p = min\{a_1, a_{k_1}, a_{k_2}, ..., a_n\}$. La capacidad residual de cada arco a lo largo de la ruta de penetración se disminuye por f_p en dirección del flujo y se incrementa por f_p en dirección inversa, es decir, para los nodos i y j en la ruta, el flujo residual se cambia de la (c_{ij}, c_{ji}) actual a $(c_{ij} - f_p, c_{ji} + f_p)$ si el flujo es de i a j, o $(c_{ij} + f_p, c_{ji} - f_p)$ si el flujo es de j a i. Inicializar $i = 1$ y volvemos al paso 2 para intentar una nueva ruta de penetración.

6. Una vez aquí, hemos determinado m rutas de penetración. El flujo máximo en la red será la suma de los flujos máximos en cada ruta obtenida, es decir: $F = f_1 + f_2 + ... + f_m$. Teniendo en cuenta que las capacidades residuales inicial y final del arco (i, j) las dan (C_{ij}, C_{ji}) y (c_{ij}, c_{ji}) respectivamente, el flujo máximo

para cada arco se calcula como sigue: sea $(\alpha, \beta) = (C_{ij} - c_{ij}, C_{ji} - c_{ji})$, si $\alpha > 0$, el flujo óptimo de i a j es α, de lo contrario, si $\beta > 0$, el flujo óptimo de j a i es β. Es imposible lograr que tanto α como β sean positivas.

2.2 Descripción de una Refinería

Esta sección habla del comportamiento de la refinería y de las características que tiene el proceso de refinación del petróleo, siendo la base de los supuestos realizados en la generación de flujos de masa y de la detección correspondiente de los flujos para formar una base de datos en el Capítulo 3 que nos ayudan a abordar el problema. La industria del petróleo es la más grande y la que más se puede extender de las industrias químicas de proceso (Torres-Robles y Castro-Arellano, 2002). El impacto que tiene en la economía y en la vida, no solamente nacional sino mundial, es tremendo (Torres-Robles y Castro-Arellano, 2002). Asimismo, de que es la más compleja, física y químicamente hablando, de todas las industrias químicas de proceso (Barberii, 1998).

La operación de una refinería de petróleo es muy compleja y para reducir esta complejidad al entendimiento de un grupo de procesos se requiere del conocimiento fundamental de los mismos. En el libro de *Análisis y Simulación de Procesos de Refinación del Petróleo* (Torres-Robles y Castro-Arellano, 2002) se describe en términos generales una refinería típica de petróleo, se discute también en forma genérica las unidades de procesamiento que la conforman y a su vez se describe la relación funcional que tienen dichas unidades de procesamiento o procesos con equipos clave como torres de destilación, bombas, compresoras, turbinas, cambiadores de calor, expansores, etc. En este libro nos basamos para realizar esta sección.

Una refinería es una planta de manufactura de productos químicos y combustibles. Como se puede observar en la Figura 2.2, la materia prima es petróleo crudo y los productos finales son gasolina, querosina, nafta, combustóleos, lubricantes, asfalto, azufre, gas y otros productos derivados del petróleo.

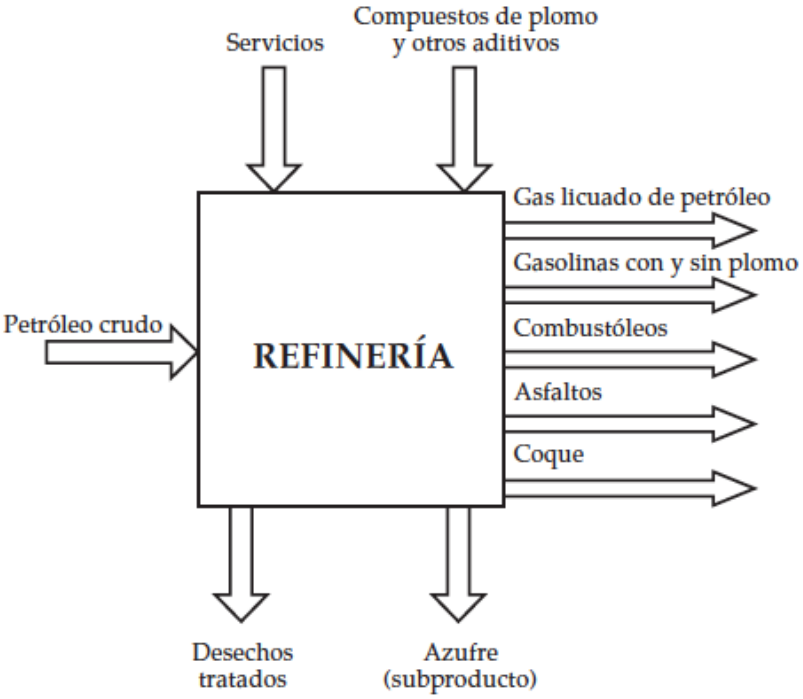


Figura 2.2: Entradas y salidas en una refinería (Torres-Robles y Castro-Arellano, 2002).

Utilizando los conceptos de teoría de grafos se puede representar el proceso como un nodo con arcos de entrada y salida (Figura 2.3).

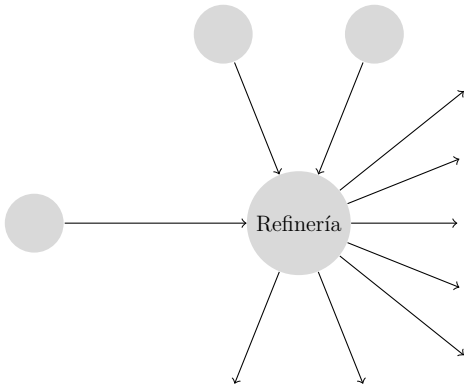


Figura 2.3: Representación gráfica de una refinería con cuatro nodos (proceso) y siete arcos de salida.

La refinería convierte el petróleo crudo y produce una variedad de derivados a través de una serie compleja de reacciones químicas y de cambios físicos que se pueden englobar básicamente en las seis siguientes operaciones principales:

- *Fraccionar*: porque separa una mezcla de hidrocarburos líquidos en diversos grupos específicos que incluyen a la gasolina, el diésel, los combustibles y otras sustancias más ligeras.
- *Desintegrar*: porque rompe los hidrocarburos grandes, convirtiéndolos en compuestos más pequeños y de mayor utilidad. La desintegración puede llevarse a cabo térmica o catalíticamente.
- *Rearreglar*: porque con altas temperaturas y con catalizadores rearregla la estructura química de los hidrocarburos del petróleo. Algunos hidrocarburos de cadena lineal son transformados en hidrocarburos cíclicos o de cadena circular; del mismo modo los hidrocarburos cíclicos son transformados.
- *Combinar*: porque hace reaccionar dos o más hidrocarburos o no hidrocarburos, tales como el azufre o el hidrógeno, para obtener otros productos que son considerados como mejorados.
- *Tratar*: porque convierte materiales contaminantes a una forma tal que pueden ser desechados al medio ambiente sin causar problemas ecológicos.
- *Mezclar*: porque combina diferentes líquidos para producir los materiales finales con ciertas propiedades deseadas.

2.2.1 Subdivisión de una Refinería

Una refinería típica podría ser subdividida en doce procesos o unidades (nodos), aunque en ocasiones podrá contar con más, dependiendo de si se integran los procesos que elaboran compuestos oxigenados. Solo se mencionarán brevemente la relación operacional que tienen estos procesos, así como la función principal que cumplen,

los flujos que manejan y los productos elaborados en cada uno de ellos. Esta relación se puede observar esquemáticamente (Figura 2.2). Esto puede ser representado por un grafo comprendido con un conjunto de doce nodos, cada nodo representa a un proceso de la refinería.

Destilación primaria de crudo: este proceso inicia la refinación del petróleo y su función es separar los diferentes componentes del crudo en una torre de destilación. Los productos del proceso son gas combustible, gasolina de destilación directa, naftas ligera y pesada, combustóleos y crudo reducido (Figura 2.4). El proceso lo podemos representar como un nodo con dos arcos de entrada y cuatro arcos de salida (Figura 2.5).

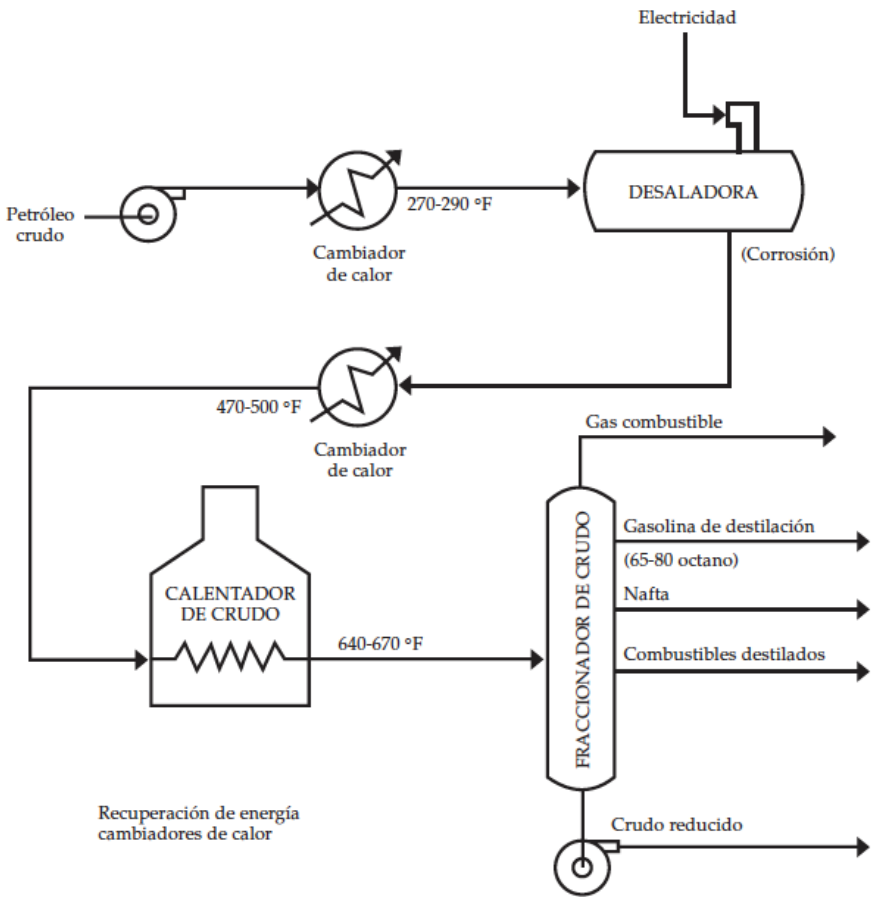


Figura 2.4: Destilación primaria (Torres-Robles y Castro-Arellano, 2002).

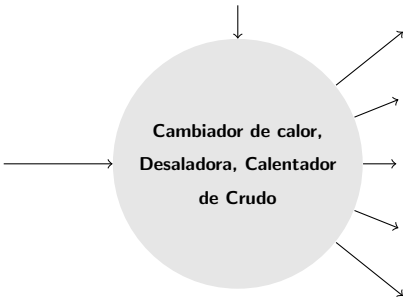


Figura 2.5: Representación gráfica del proceso de destilación primaria.

Destilación al vacío: en este proceso se alimenta el crudo reducido de la destilación primaria y su función es la de separar aún más esta fracción realizando una destilación al vacío. Los productos obtenidos son los siguientes: gasóleos ligero y pesado, aceites lubricantes, asfalto o combustóleo pesado y la alimentación del coquizador (Figura 2.6). El proceso lo podemos representar como un nodo con un arco de entrada y cinco arcos de salida (Figura 2.7).

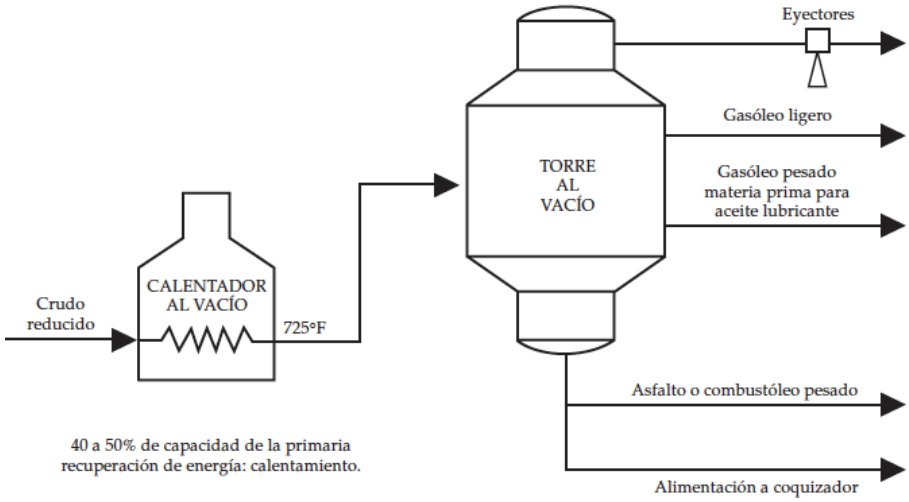


Figura 2.6: Destilación al vacío (Torres-Robles y Castro-Arellano, 2002).

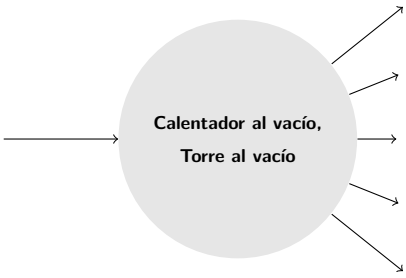


Figura 2.7: Representación gráfica del proceso de destilación al vacío.

Hidrodesulfuración: en esta unidad se purifica la corriente alimentada eliminándole básicamente los compuestos de azufre. También se eliminan nitrógeno, oxígeno y metales pesados. Todo esto con el objeto de proteger los catalizadores empleados en otros procesos de la refinería. Los flujos de entrada que se manejan en este proceso son hidrocarburos seleccionados de la destilación primaria con hidrógeno convirtiendo los compuestos de azufre en sulfuro de hidrógeno el cual se elimina en forma gaseosa. Los productos del proceso son: gasolina desulfurizada, naftas ligera y pesada desulfurizada, o combustóleos desulfurizados o combustóleos catalíticos desulfurizados (Figura 2.8). El proceso lo podemos representar como un nodo con un arco de entrada y siete arcos de salida (Figura 2.9).

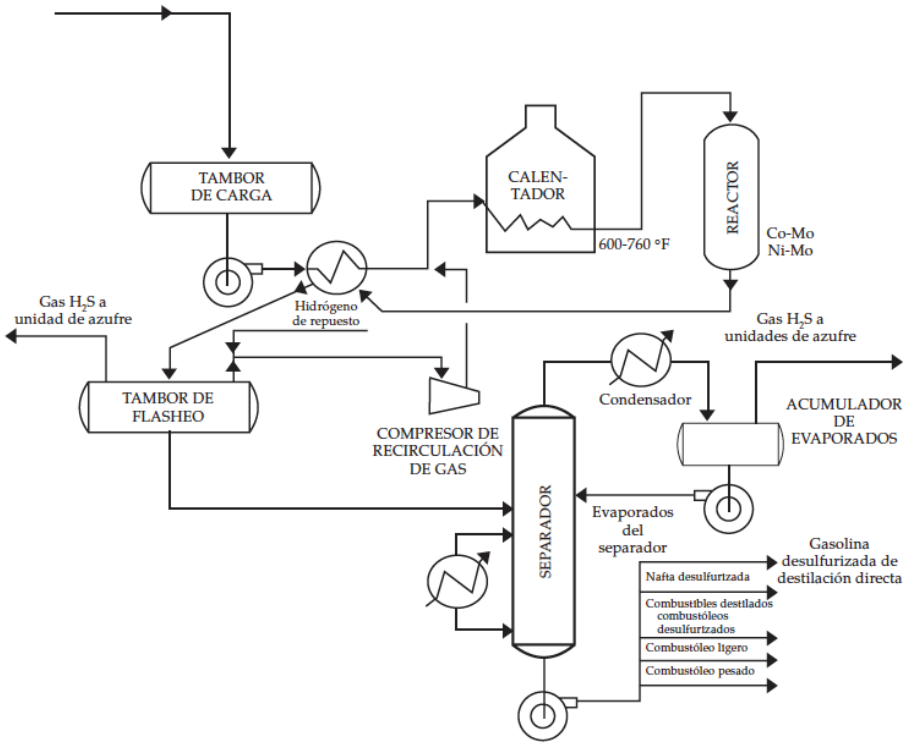


Figura 2.8: Hidrodesulfuración (Torres-Robles y Castro-Arellano, 2002).

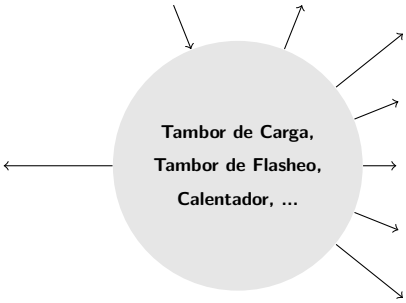


Figura 2.9: Representación gráfica del proceso de hidrodesulfuración.

Reformación: la nafta desulfurizada se bombea a este proceso, el cual cumple la función de rearreglar los hidrocarburos por medio de desintegración en catalizadores de platino-aluminio y bimetálico para producir gasolina de alto octano. Los productos de la unidad son: gasolina reformada de alto octano, hidrógeno, gas combustible y residuos ligeros como los propanos C_3s y butanos C_4s (Figura 2.10). El proceso lo

podemos representar como un nodo con dos arcos de entrada y cinco arcos de salida salida (Figura 2.11).

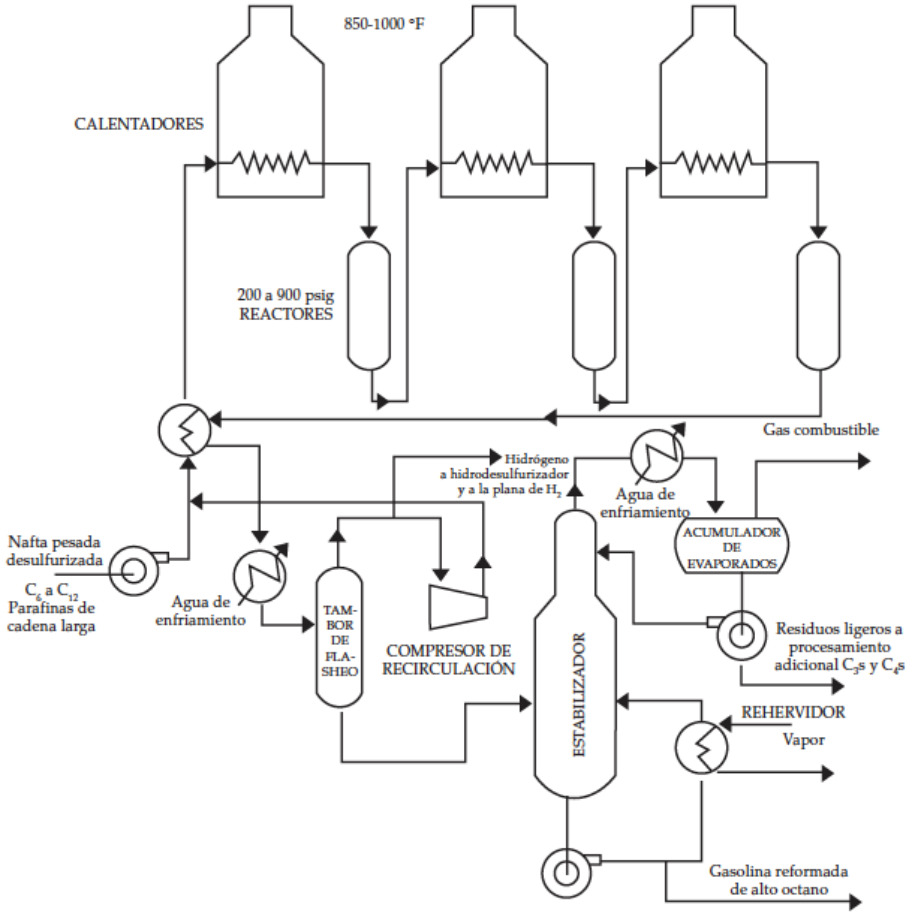


Figura 2.10: Reformación (Torres-Robles y Castro-Arellano, 2002).

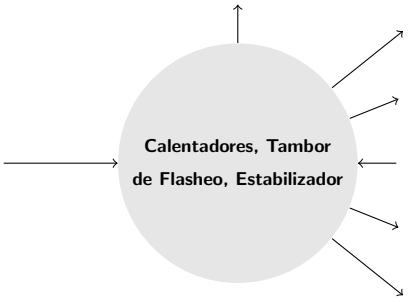


Figura 2.11: Representación gráfica del proceso de reformación.

Isomerización: en este proceso se emplea como materia prima la gasolina producto de la destilación primaria y desulfurizada por la hidrodesulfurización. En este proceso también son rearreglados o reacomodados los hidrocarburos de la gasolina, en presencia de un catalizador de platino o de cloruro de aluminio. El producto es la gasolina de alto octano y gas combustible (Figura 2.12). El proceso lo podemos representar como un nodo con dos arcos de entrada y dos arcos de salida (Figura 2.13).

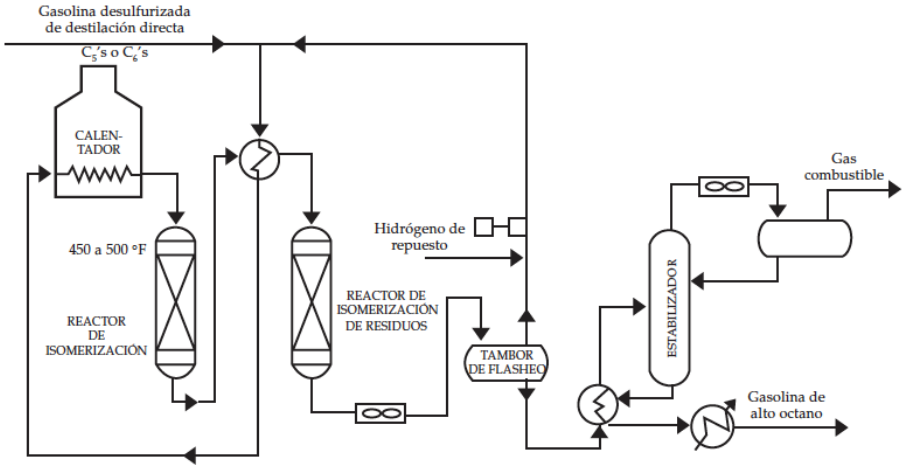


Figura 2.12: Isomerización (Torres-Robles y Castro-Arellano, 2002).

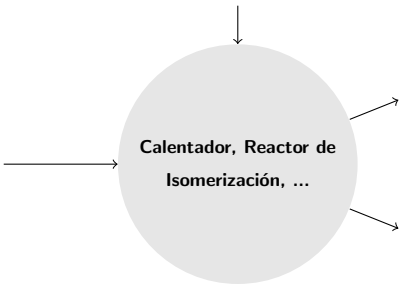


Figura 2.13: Representación gráfica del proceso de isomerización.

Desintegración catalítica: el gasóleo ligero producido en la destilación al vacío sirve esencialmente de carga en este proceso, el cual cumple la función de romper los hidrocarburos del gasóleo con ayuda de un catalizador que normalmente es de com-

puestos de sílice-aluminio. Durante el proceso se forma coque (depósitos de carbón), que se deposita en el catalizador reduciendo con esto su actividad catalítica. El catalizador se regenera quemando el coque con aire. Los productos en este proceso son gasolina catalítica, destilados ligeros y gasolina que se emplean como combustóleos destilados (Figura 2.14). El proceso lo podemos representar como un nodo con dos arcos de entrada y cinco arcos de salida (Figura 2.15).

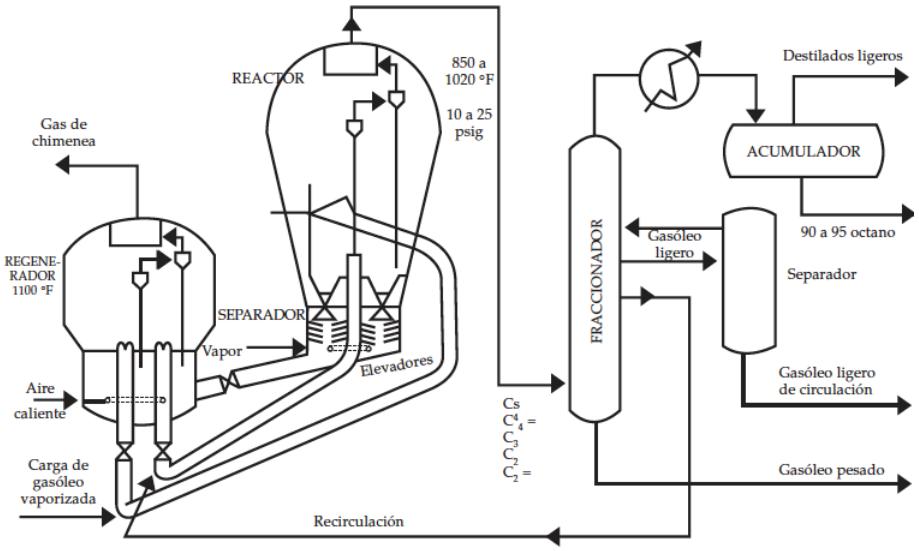


Figura 2.14: Desintegración (Torres-Robles y Castro-Arellano, 2002).

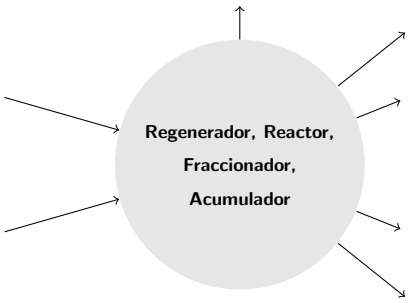


Figura 2.15: Representación gráfica del proceso de desintegración catalítica.

Alquilación: los compuestos de cuatro átomos de carbono, butilenos y butanos, y algunas veces los de tres átomos de carbono, propilenos, que provienen de otros

procesos en la refinería, se hacen reaccionar en esta unidad de alquilación para formar el alquilado ligero. En esta unidad se utiliza como catalizador el ácido fluorhídrico o ácido sulfúrico. Los productos del proceso son: alquilado ligero de alto octano y gas licuado del petróleo o LP (Figura 2.16). El proceso lo podemos representar como un nodo con dos arcos de entrada y dos arcos de salida (Figura 2.17).

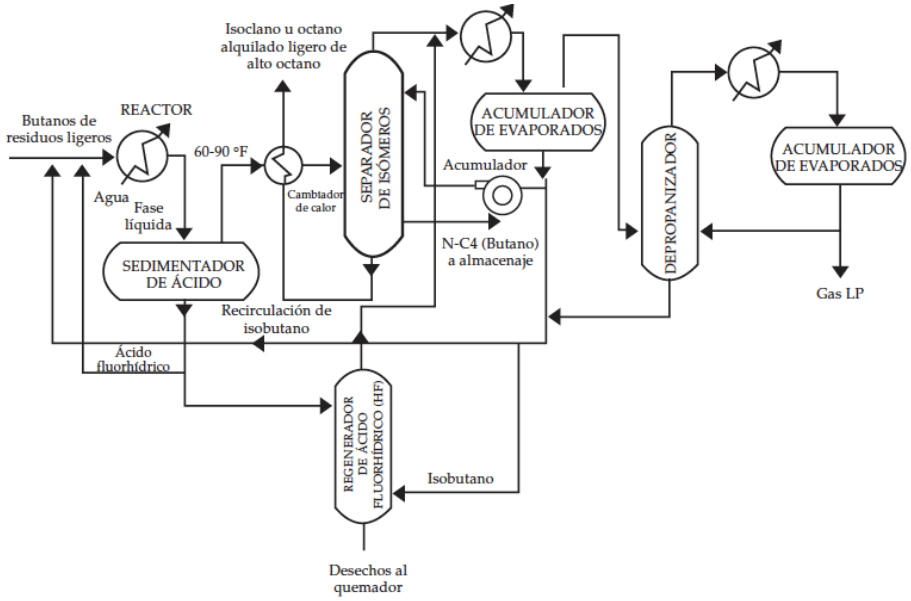


Figura 2.16: Alquilación (Torres-Robles y Castro-Arellano, 2002).

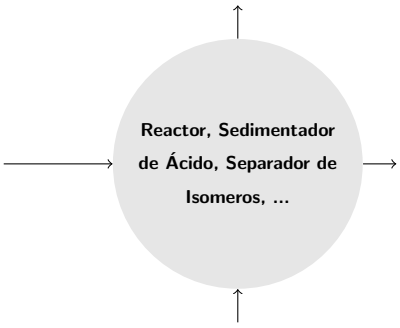


Figura 2.17: Representación gráfica del proceso de alquilación.

Polimerización: en este proceso son aprovechados los polipropilenos que se producen en la desintegración catalítica haciéndolos reaccionar entre sí y en presencia

de un catalizador con base en el ácido fosfórico o de sílice. En este proceso se producen la gasolina de polimerización de alto octano y gas licuado del petróleo o LP (Figura 2.18). El proceso lo podemos representar como un nodo con tres arcos de entrada y cuatro arcos de salida (Figura 2.19).

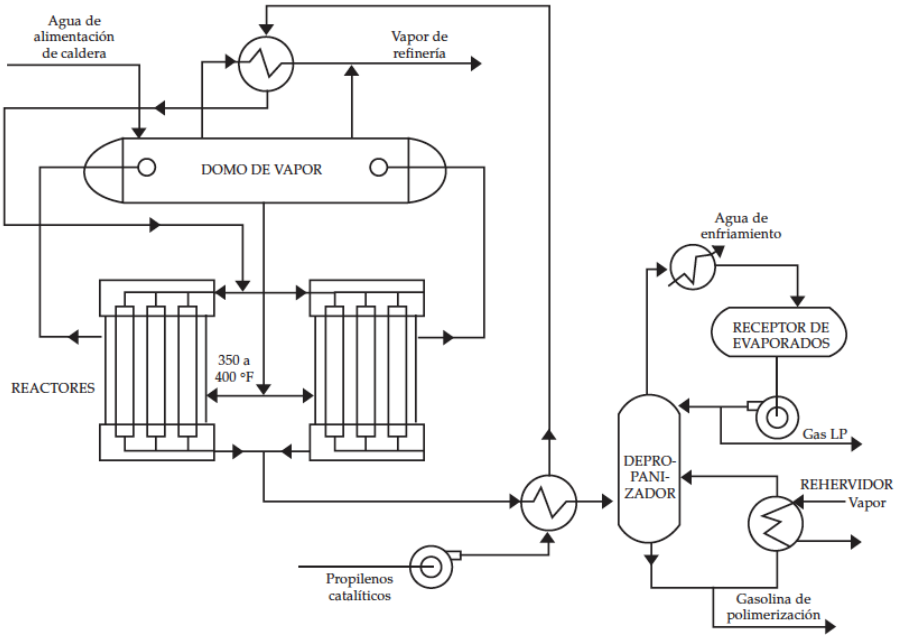


Figura 2.18: Polimerización (Torres-Robles y Castro-Arellano, 2002).

Figura 2.19: Representación gráfica del proceso de polimerización.

Coquización: los residuos de la destilación al vacío son desintegrados térmicamente para convertirlos en combustibles ligeros y en coque. Los productos en este

proceso son: gas combustible, nafta, gasóleos ligeros y pesados y coque (Figura 2.20). El proceso lo podemos representar como un nodo con dos arcos de entrada y seis arcos de salida (Figura 2.21).

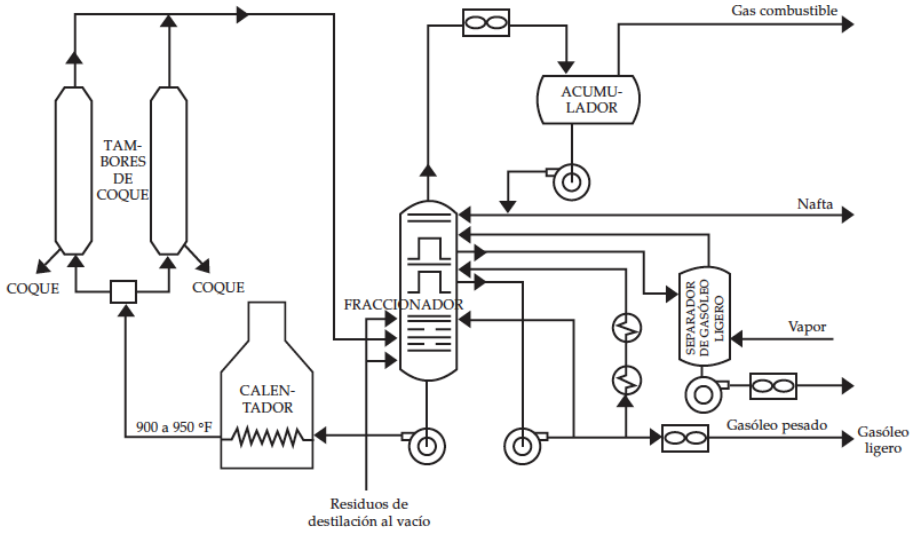


Figura 2.20: Coquización (Torres-Robles y Castro-Arellano, 2002).

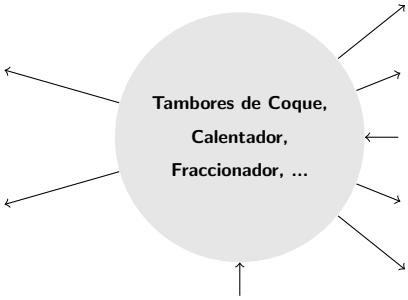


Figura 2.21: Representación gráfica del proceso de coquización.

Recuperación de azufre: en varios procesos de una refinería se produce ácido sulfhídrico (H_2S); en la mayoría de las hidrodesulfuradoras éste es recolectado en forma gaseosa o disuelto en soluciones de amina y es convertido en materiales más comerciales que son el azufre y el ácido sulfúrico. El producto de la unidad es azufre (Figura 2.22). El proceso lo podemos representar como un nodo con tres arcos de

entrada y tres arcos de salida (Figura 2.23).

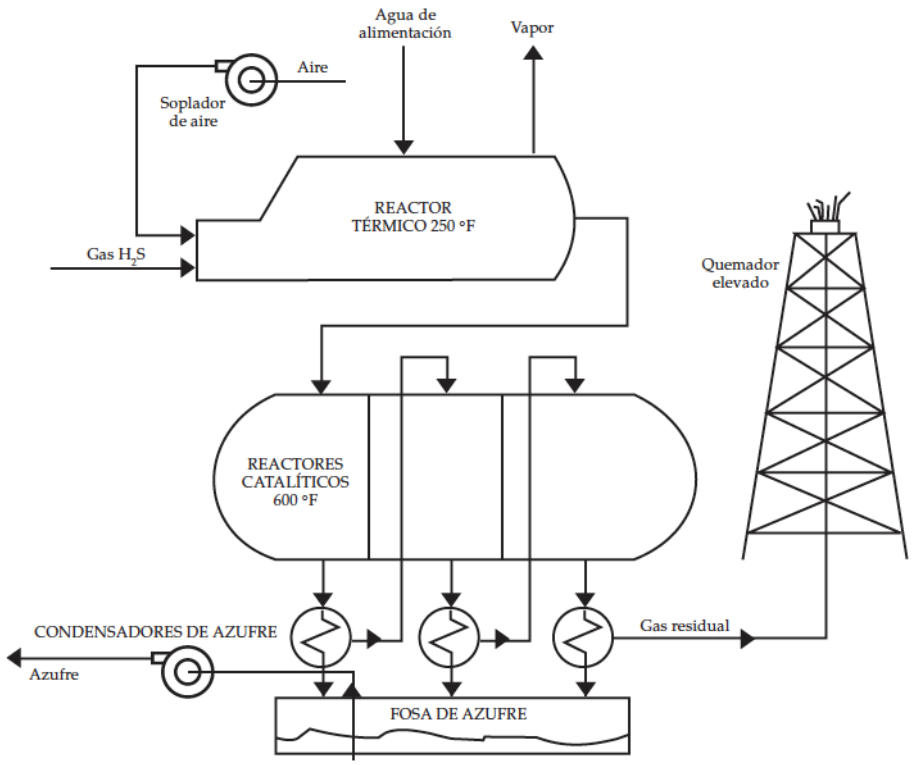


Figura 2.22: Recuperación de azufre (Torres-Robles y Castro-Arellano, 2002).

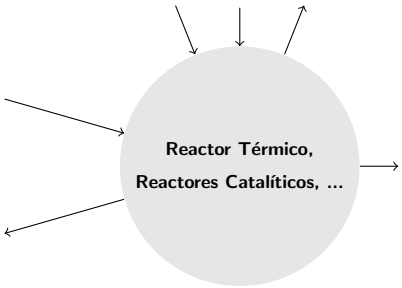


Figura 2.23: Representación gráfica del proceso de recuperación de azufre.

Mezclado de gasolina: en esta unidad se reciben todos los componentes para el mezclado de gasolinas, estos provienen de diferentes unidades. Una vez formada una mezcla se le agregan aditivos que sirven como antidetonantes y que dan los grados de octanaje necesarios en las gasolinas con antidetonante de alto y bajo octano.

Cuando no se les agrega ningún compuesto oxigenado se obtienen las gasolinas de alto y bajo octano y gasolina de aviación (Figuras 2.24 y 2.25). Ambos procesos, por separado, los podemos representar como un nodo con ocho arcos de entrada y seis arcos de salida (Figura 2.26).

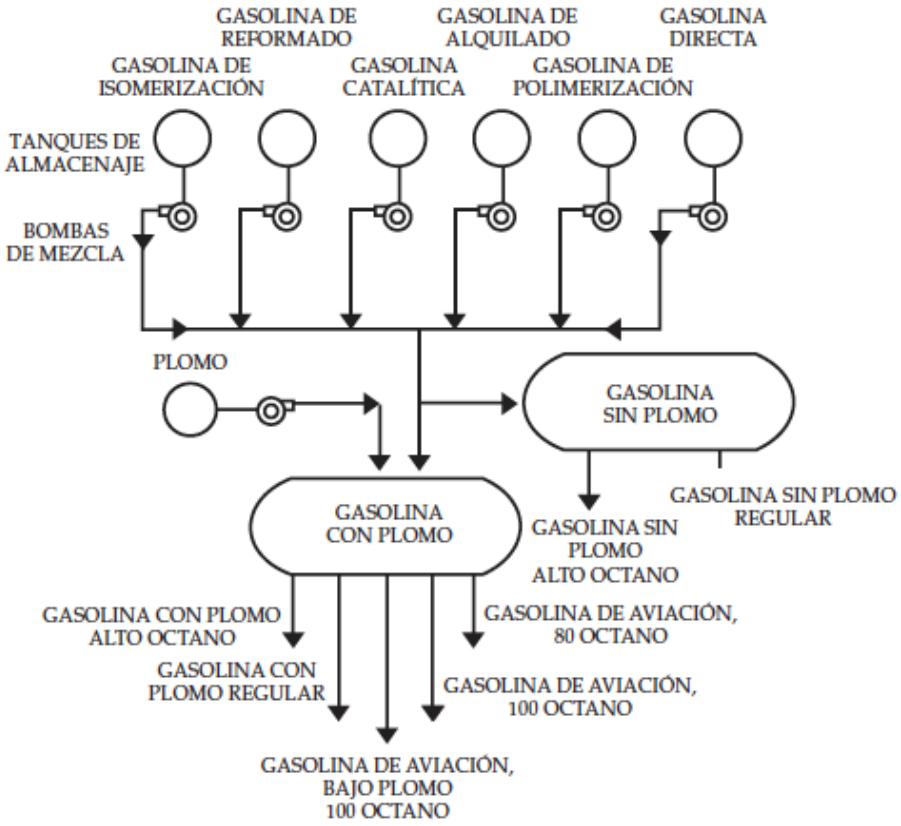


Figura 2.24: Mezclado por carga (Torres-Robles y Castro-Arellano, 2002).

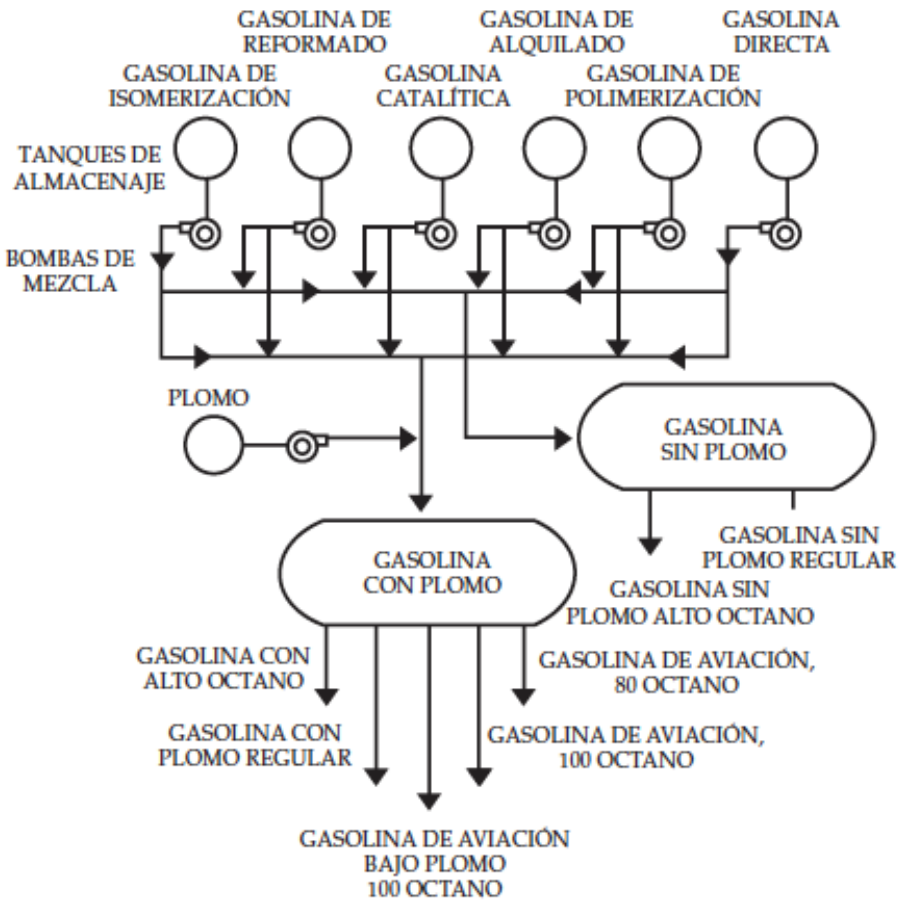


Figura 2.25: Mezclado continuo (Torres-Robles y Castro-Arellano, 2002).

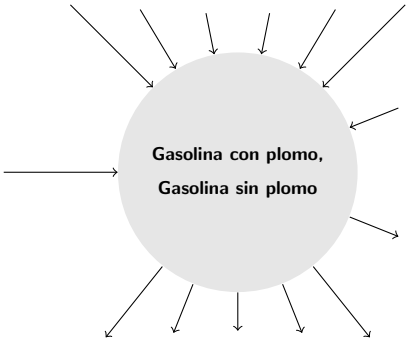


Figura 2.26: Representación gráfica de los procesos de mezclado por carga y continuo.

Unidad de servicios auxiliares: en esta unidad se da un soporte técnico a los

otros once procesos suministrándoles vapor de agua de alta, media y baja presión, electricidad, aire comprimido y agua de enfriamiento. En algunas refinerías se utiliza parte del vapor para producir electricidad y en otras la electricidad se compra y se utiliza totalmente el vapor generado en una caldera para los procesos. El calor necesario para la producción del vapor proviene del quemado de combustibles y derivados del petróleo de bajo valor comercial provenientes de los diferentes procesos (Figura 2.27). El proceso lo podemos representar como un nodo con dos arcos de entrada y cuatro arcos de salida (Figura 2.28).

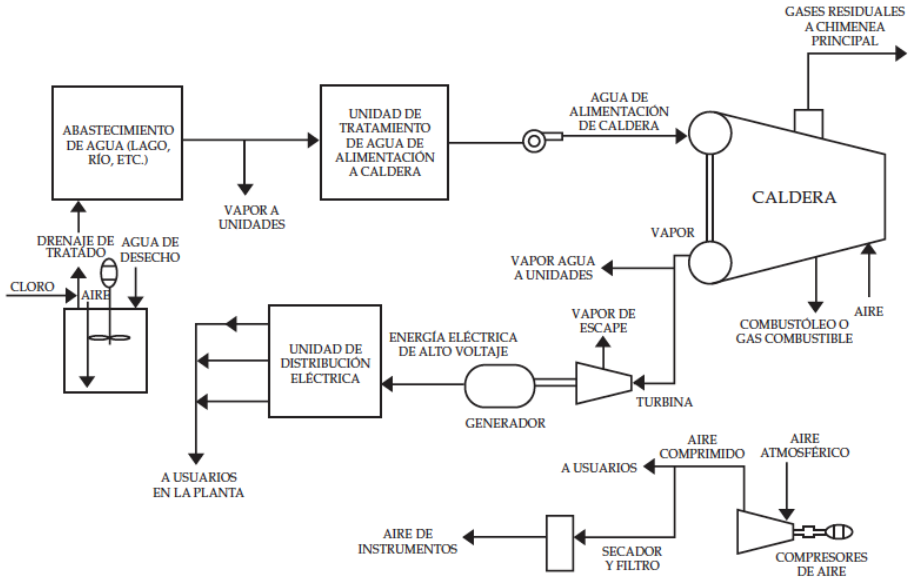


Figura 2.27: Unidad de servicios auxiliares (Torres-Robles y Castro-Arellano, 2002).

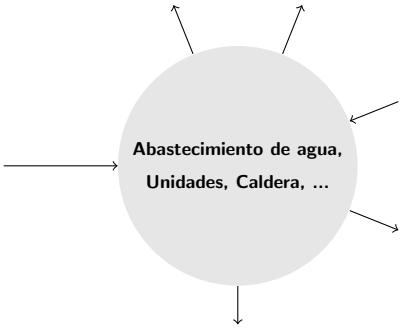


Figura 2.28: Representación gráfica de la unidad de servicios auxiliares.

2.2.2 Equipo Principal en los Procesos de una Refinería

La información presentada a continuación se encuentra en el libro de *Análisis y Simulación de Procesos de Refinación del Petróleo* (Torres-Robles y Castro-Arellano, 2002).

A pesar de las diferencias en función y en los flujos y productos que maneja cada uno de los doce procesos esenciales de una refinería, el equipo de proceso con que cuenta cada unidad es muy similar. De ahí que se pueda hacer un listado del equipo principal que se tiene en cada proceso y así resumir operaciones modulares que sirvan para cualquiera de ellos. Al conocer qué equipos tiene cada proceso y haciendo uso de la teoría de grafos podemos englobar un proceso que tenga más de un equipo como si fuera un solo nodo en el cual entra el material para descomponerse o simplemente pasar por el proceso para mejorar sus propiedades.

- *Destilación primaria*: bombas, cambiadores de calor, desaladora, calentador y torre de fraccionamiento (Figura 2.4).
- *Destilación del vacío*: calentador, torre de destilación al vacío, eyector (Figura 2.6).
- *Hidrodesulfuración*: tanque de carga, bomba, cambiador de calor, calentador,

reactor, tanque de vaporización instantánea (flasheo), compresor, separador, rehervidor, condensador y acumulador (Figura 2.8).

- *Reformación*: bombas, cambiadores de calor, calentador, reactor, enfriador, tanque de flasheo, compresor, estabilizador, rehervidor, condesador y acumulador (Figura 2.10).

- *Isomerización*: calentador, reactor, cambiador de calor, enfriador por aire, tanque de flasheo, compresor y estabilizador (Figura 2.12).

- *Desintegración catalítica*: reactor, sedimentador, regenerador, separador, rehervidor y torre de destilación depropanizadora (Figura 2.14).

- *Alquilación*: reactor-sentador, separador de isómeros, torre de destilación (depropanizadora), sedimentador, acumulador y regenerador de ácido (Figura 2.16).

- *Polimerización*: bomba, cambiador de calor, domo de vapor, reactor, torre de destilación depropanizadora, condensador, receptor y rehervidor (Figura 2.18).

- *Coquizador*: tanques de coque, calentador, torre de fraccionamiento, enfriadores por aire, acumulador, bombas, cambiadores de calor, separador (Figura 2.20).

- *Recuperación de azufre*: soplador, reactores, condensadores, quemador, fosa y bomba (Figura 2.22).

- *Mezclado de gasolina*: tanques de almacenamiento y bombas (Figuras 2.24 y 2.25).

- *Servicios auxiliares*: calderas, bombas, compresor, generador eléctrico y sistemas de tratamiento de agua (Figura 2.27).

De esta manera quedan resumidas las operaciones en donde cada módulo tiene cierta función que varía de proceso en proceso. La diferencia entre aplicar un cierto

módulo de operación de un proceso a otro, serán los flujos de entrada y salida y sus características de composición.

Los módulos comunes son: bomba, cambiador de calor, compresor, expansor, torre de destilación, turbina, eyector, tanque, recipiente, reactor y flash. Los reactores pueden ser de diferente tipo según su geometría, su contenido y sus alimentaciones.

Si se quiere más información acerca del proceso de refinación del petróleo se recomiendan las siguientes lecturas: *El Pozo Ilustrado* (Barberii, 1998), *Handbook of Petroleum Refining Processes* (Meyers, 2004), *Handbook of Petroleum Product Analysis* (Speight, 2002). Para el funcionamiento específico de cada equipo o proceso se recomienda consultar el libro *Manual del Ingeniero Químico* (Perry y Green, 2010) en donde se explican las ecuaciones a detalle para calcular las entradas y salidas al proceso.

2.3 Medidores de Placa y Orificio

Actualmente los medidores usados en las industrias de refinación en México son medidores de flujo de presión del tipo placa y orificio (CONACYT-SENER-Hidrocarburos, 2012) por lo que en esta sección se habla de la base teórica para conocer su funcionamiento y hacer uso de la ecuación característica que nos permitirá conocer el flujo volumétrico circundante en las conexiones entre procesos siendo base de los cálculos realizados en la generación de instancias que presentamos el Capítulo 3. Conocer la información de los medidores de placa y orificio repercute en gran medida a los supuestos realizados en la generación de flujos de masa (Capítulo 3). El desempeño de los medidores depende de varios factores para obtener medidas con menor error.

La descripción de los medidores de tipo placa y orificio se tomó del libro *Flujo de Fluidos en la Fase Líquida* (Gonzáles, 2000). Para saber el funcionamiento de los medidores se recomiendan las siguientes lecturas: *La Metrología de Flujo de*

Líquidos en México (Loza-Guerrero, 2012), *Simulación del Flujo de Aire en una tubería* (de Oviedo, 0012), *Teoría de la medición de caudales y volúmenes de agua e instrumental necesario disponible en el mercado* (García-Guitiérrez, 1998), *Programa para el cálculo de tuberías y bombas centrífugas en procesos de refinación* (Gallegos-Alvarez, 2011), *Programa de computo para dimensionalizar medidores de flujo por presión diferencial en líquidos* (Campos-Lopez, 2008); para más información de la incertidumbre de los medidores se recomiendan las siguientes lecturas: *A Beginner´s Guide to Uncertainity of Measurement* (Bell, 1999), *Guía para estimar la incertidumbre de la medición* (Schmid y Lazos-Martinez, 2000), *Guide to the Evaluation of Measurement Uncertainity for Quantitative Test Results* (Eurolab, 2006), *Incertidumbre de la Medición: Teoría y Práctica* (Sáez-Ruiz y Font-Avila, 2001), Propagación de Incertidumbre (Weisser, 2010); y para mas información del comportamiento de los fluidos se recomiendan las siguientes lecturas: *Procesos de Transporte y Operaciones Unitarias* (Geankopolis, 1998), *Fenómenos de Transporte* (Bird et al., 2006), *Operaciones Unitarias en Ingeniería Química* (McCabe et al., 1991).

Los medidores de placa y orificio están constituidos por una placa delgada perforada la cual se instala entre las bridas en la tubería. Cada una de las cuales está unida a la parte correspondiente de la tubería. Entre las placas y la brida, se usan empaques para sellar los escapes de fluido. Se hacen generalmente de acero inoxidable, material que resiste satisfactoriamente la acción de los fluidos bajo medición, salvo algunos fluidos corrosivos que requieren una aleación especial de acero como monel o níquel, etc. Estos medidores constan de dos componentes, el elemento primario es el dispositivo que se coloca en la tubería para obstruir el flujo y generar una caída de presión y un elemento secundario que mide la caída de presión.

Las tomas para medir la caída de presión se colocan antes y después de la placa en cualquiera de las siguientes posiciones (Figura 2.29).

- **Tomas en las bridas**: cuando se instalan las conexiones a una pulgada de cada cara de la placa, es el tipo de conexión más utilizado y no hay que perforar la tubería.

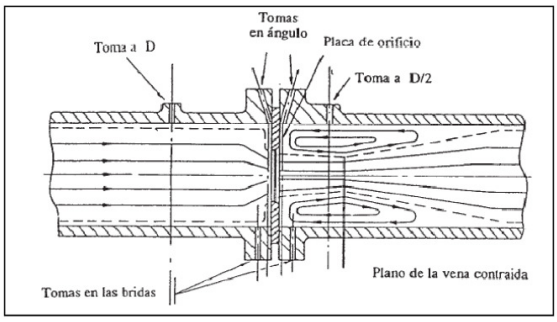


Figura 2.29: Diferentes tomas de presión de una placa y orificio (Gonzáles, 2000).

- **Tomas en la vena contraída**: la toma de alta presión se coloca en un punto que dista un diámetro nominal de la tubería, mientras que la toma de baja presión depende de la relación entre el diámetro del orificio y el de la tubería.
- **Tomas en la tubería**: están localizadas a una distancia de $2\frac{1}{2}$ veces el diámetro nominal y 8 veces el diámetro nominal de la placa. Miden la pérdida de presión permanente a través de un orificio, requiere mayor cantidad de tramos rectos de tubería, hay que perforar la tubería, se usa en la medición de gases y es la que permite mayor estabilidad en la presión diferencial.

El orificio de la placa puede ser de tres tipos (Figura 2.30):

- **Tipo concéntrica**: Para aplicaciones de líquidos limpios, de baja viscosidad para la mayoría de los gases y vapor a baja velocidad.
- **Tipo excéntrico y segmental**: se utilizan principalmente en aplicaciones de fluidos que contienen materiales en suspensión o condensado de vapor.

Hay orificios de dos formas: de bordes cuadrados y de bordes biselados, siendo ésta última la más común por ser la más barata y más fácil de cambiar. Su popularidad se debe a su simplicidad y al bajo mantenimiento pero desafortunadamente no es el método de medición más preciso estando la incertidumbre en la media entre el 0.5 % y el 3 % dependiendo del fluido, la configuración de la tubería aguas arriba

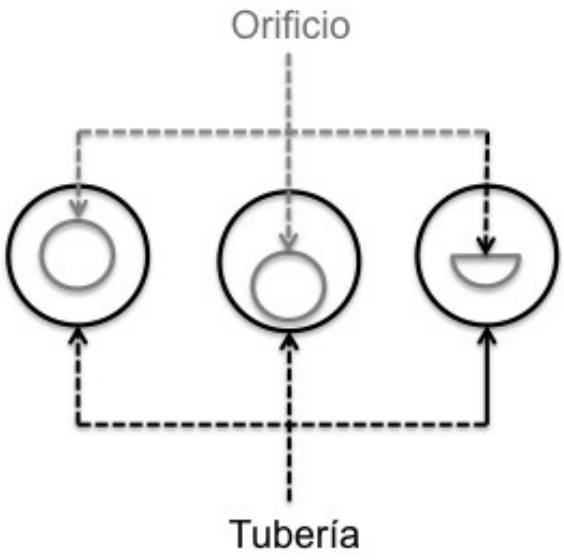


Figura 2.30: Tipos de orificios (Gonzáles, 2000).

y si las correcciones por diversos efectos se hicieron o no en la fórmula del flujo volumétrico.

2.3.1 Deducción de la Ecuación del Medidor de Placa y Orificio

Presentamos la deducción de la ecuación utilizada en el Capítulo 3 para realizar los cálculos de los flujos volumétricos y después pasarlos a flujos de masa, siendo la base del generador de instancias para conocer los flujos de operación en la refinería propuesta.

En la placa de orificio, para calcular el flujo volumétrico real se deben considerar el coeficiente de contracción y velocidad (Campos-Lopez, 2008). El coeficiente de velocidad (C_v) es la relación entre la velocidad media real en la sección recta del chorro y la velocidad media lineal que se tendría sin efectos de rozamiento.

$$\begin{aligned} C_v &= \frac{v_{real}}{v_{ideal}}, \\ v_{real} &= C_v * v_{ideal}. \end{aligned} \tag{2.1}$$

El coeficiente de contracción (C_c) es la relación entre el área de la sección recta contraída de un chorro (A_1) y el área del orifico por el cual pasa el fluido (A_0).

$$\begin{aligned} C_c &= \frac{A_1}{A_0}, \\ A_1 &= C_c A_0. \end{aligned} \tag{2.2}$$

Si se aplican la ecuación de continuidad (ecuación (2.4)) y la ecuación energía (ecuación (2.4)) para el flujo incompresible estacionario entre la sección recta contraída de un flujo y del orifico del medidor por el cual pasa el fluido.

$$v_1 A_1 = v_0 A_0, \tag{2.3}$$

$$P_1 + \rho g z_1 + \frac{1}{2}\rho {v_1}^2 = P_0 + \rho g z_0 + \frac{1}{2}\rho {v_0}^2. \tag{2.4}$$

Donde P_1 es la presión en la sección recta contraída, v_1 es la velocidad en la sección recta contraída, z_1 es la elevación en la sección recta contraída; P_0 es la presión en el orificio, v_0 es la velocidad en el orificio, z_0 es la elevación en el orificio. Además, g es la gravedad y ρ representa la densidad del fluido. Si se combinan las ecuaciones (2.4) y (2.4) para despeja la velocidad en el orificio v_0 (v_{ideal}):

$$v_{ideal} = \sqrt{\frac{2\delta P_{temporal}}{\rho * 1 - {\frac{A_0}{A_1}}^2}}. \tag{2.5}$$

El caudal real esta dado por:

$$Q_{real} = v_{real} A_1.$$

Sustituyendo v_{real} con la ecuación (2.1) y A_1 con la ecuación (2.2):

$$Q_{real} = v_{ideal} C_v C_c A_0. \tag{2.6}$$

Sustituyendo la ecuación (2.6) con la ecuación (2.5):

$$\begin{aligned} Q_{real} &= \frac{C_v C_c}{\sqrt{1 - {\frac{A_0}{A_1}}^2}} A_0 \sqrt{\frac{2(P_1 - P_0)}{\rho}}, \\ C_d &= C_v C_c, \\ Q &= \frac{C_d}{\sqrt{1 - {\frac{A_0}{A_1}}^2}} A_0 \sqrt{\frac{2\delta P_{temporal}}{\rho}}. \end{aligned} \tag{2.7}$$

Para sección transversal circular, conviene introducir la relación de diámetros, mas usado en la práctica: $\beta = \frac{d}{D} = \frac{A_0}{A_1}$, y utilizar la ecuación (2.7) como:

$$Q = \frac{C_d}{\sqrt{1-\beta^4}} A_0 \sqrt{\frac{2\delta P_{temporal}}{\rho}}. \tag{2.8}$$

Si se desea conocer más a detalle cómo se obtuvo esta ecuación, consultar la tesis *Programa de Cómputo para Dimensionalizar Medidores de Flujo por Presión Diferencial en Líquidos* (Campos-Lopez, 2008) en donde se explica qué tipos de medidores son los más utilizados, así como de los medidores de placa y orificio, además de exponer detalladamente cómo se obtiene la ecuación (2.8).

2.4 Redes Neuronales Artificiales

En esta sección hacemos un breve resumen de las redes neuronales artificiales que nos sirve de ayuda en el Capítulo 4 donde las utilizamos como herramienta para llevar a cabo la clasificación de los flujos obtenidos en el generador de instancias. La información para realizar las siguientes secciones de las redes neuronales se basaron en el documento siguiente: *Introducción a las redes neuronales aplicadas* (Marín, 2010). Si se desea adentrar más en las redes neuronales artificiales, saber cómo surgieron o las aplicaciones que se hayan realizado con esta herramienta se recomiendan las lecturas de: *Learning Algorithms for Neural Networks* (Atiya, 1991), *Simulación de Redes Neuronales Artificiales* (Alegre-Lopez, 2003), *Neural Network for Pattern Recognition* (Bishop, 1995).

Las redes neuronales aplicadas están, en general, inspiradas en las redes neuronales biológicas, aunque poseen otras funcionalidades y estructuras de conexión distintas a las vistas desde la perspectiva biológica. Las características principales de las redes son las siguientes:

1. *Auto-Organización y Adaptabilidad*: utilizan algoritmos de aprendizaje adaptativo y auto-organización, por lo que ofrecen mejores posibilidades de procesado

robusto y adaptativo, carácteristica que buscamos para resolver el problema ya que se podrá configurar a diferentes tamaños de refinerías.

2. *Procesado no Lineal*: aumenta la capacidad de la red para aproximar funciones, clasificar patrones y aumenta su inmunidad frente al ruido. Esto nos sirve ya que los sistemas en los que se implementará tienen gran cantidad de ruido debido a las caracteristicas de medición de los fluidos.

3. *Procesado Paralelo*: normalmente se usa un gran número de nodos de procesado, con alto nivel de interconectividad. Como se mencionó en el Capítulo 2, existe una gran cantidad de procesos, que representamos mediante nodos, esto nos ayudará a manejar esa gran cantidad de nodos presentes en la refinería.

Estas tres características cumplen con la estructura de nuestro problema ya que si queremos aumentar el tamaño de la red neuronal, necesitamos que el método de elección sea capaz de ser susceptible a aumentos de nodos en la refinería, para probarlo con diferentes tamaños de refinerías.

El elemento básico de computación (modelo de neurona) se le llama habitualmente nodo o unidad. Recibe una entrada y_j desde otras unidades o de una fuente externa de datos. Cada entrada tiene un peso asociado w_{ij}, que se va modificando en el llamado proceso de aprendizaje. Cada unidad aplica una función f dada de la suma de las entradas ponderadas mediante los pesos. El resultado y_i puede servir como salida de otras unidades (ecuación (2.9)).

$$y_i = \sum jf(w_{ij}y_j). \tag{2.9}$$

Las características de las redes neuronales juegan un importante papel, por ejemplo, en el procesado de señales e imágenes. Se usan arquitecturas que comprenden elementos de procesado adaptativo paralelo, combinados con estructuras de interconexiones jerárquicas. En el Capítulo 4 se utilizan los flujos generados mediante el algoritmo del Capítulo 3. Los flujos son separados en las siguientes dos fases

para el modelo con redes neuronales artificiales:

- **Fase de entrenamiento:** se usa un conjunto de datos o patrones de entrenamiento para determinar los pesos que definen el modelo de red neuronal. Se calculan de manera iterativa, de acuerdo con los valores de los valores de entrenamiento, con el objeto de minimizar el error cometido entre la salida obtenida por la red neuronal y la salida deseada.
- **Fase de prueba:** en la fase anterior, el modelo puede que se ajuste demasiado a las particularidades presentes en los patrones de entrenamiento, perdiendo su habilidad de generalizar su aprendizaje a casos nuevos (sobreajuste). Para evitar el problema del sobreajuste, es aconsejable utilizar un segundo grupo de datos diferentes a los de entrenamiento, el grupo de validación, que permita controlar el proceso de aprendizaje.

Normalmente, los pesos óptimos se obtienen minimizando alguna función de energía. Por ejemplo, un criterio muy utilizado en el llamado entrenamiento supervisado, es minimizar el error cuadrático medio entre el valor de salida y el valor real esperado. La función de energía en la metodología que proponemos en ésta tesis es minimizar la suma de los errores cuadráticos entre el valor de salida y el valor real esperado.

Estructuras de conexión de atrás hacia delante

Una red neuronal se determina por las neuronas y la matriz de pesos. Hay tres tipos de capas de neuronas: la capa de entrada, la capa oculta y la capa de salida.

Entre dos capas de neuronas existe una red de pesos de conexión, que puede ser de los siguientes tipos:

1. Conexiones hacia delante: los valores de las neuronas de una capa inferior son propagados hacia las neuronas de la capa superior por medio de las redes de conexiones hacia adelante.

2. Conexiones hacia atrás: estas conexiones llevan los valores de las neuronas de una capa superior a otras de la capa inferior.

3. Conexiones laterales: un ejemplo típico de este tipo es el circuito, el ganador toma todo (*winner-takes-all* en inglés), que cumple un papel importante en la elección del ganador: a la neurona de salida que da el valor más alto se le asigna el valor total (por ejemplo, 1), mientras que a todas las demás se le da un valor de 0.

4. Conexiones con retardo: los elementos de retardo se incorporan en las conexiones para implementar modelos dinámicos y temporales, es decir, modelos que precisan de memoria.

En la red neuronal que proponemos en ésta tesis se utiliza la conexión hacia adelante.

Tamaño de las Redes Neuronales

En una red multicapa de propagación hacia delante, puede haber una o más capas ocultas entre las capas de entrada y salida. El tamaño de las redes depende del número de capas y del número de neuronas ocultas por capa. El número de unidades ocultas está directamente relacionado con las capacidades de la red. Para que el comportamiento de la red sea correcto, se tiene que determinar apropiadamente el número de neuronas de la capa oculta. Para la determinación del número de neuronas en la capa oculta se realizó en esta tesis una experimentación en donde se varía el numero de neuronas en la capa oculta y comparando su rendimiento, la experimentación se explica en el Capítulo 4.

El Perceptrón Multicapa

Es una red de retropropagación conteniendo al menos una capa oculta con suficientes unidades no lineales puede aproximar cualquier tipo de función o relación continua entre un grupo de variables de entrada y salida. Esta propiedad convierte a las redes perceptrón multicapa en herramientas de propósito general, flexibles y

no lineales.

Rumelhart et al. (1986) formalizaron un método para que una red del tipo perceptrón multicapa aprendiera la asociación que existe entre un conjunto de patrones de entrada y sus salidas correspondientes: método retropropagación del error (propagación del error hacia atrás). Esta red tiene la capacidad de generalización: facilidad de dar salidas satisfactorias a entradas que el sistema no ha visto nunca en su fase de entrenamiento.

Debido a esta propiedad utilizamos el perceprón multicapa en nuestra propuesta al resolver el problema presentado en esta tesis, el cual se detalla en la metodología del Capítulo 4.

Arquitectura

La arquitectura presentada a continuación es parte de la red neuronal artificial usada en la metodología propuesta en el Capítulo 4.

Un perceptrón multicapa está compuesto por una capa de entrada, una capa de salida y una o más capas ocultas, aunque se ha demostrado que para la mayoría de problemas bastará con una sola capa oculta. En la Figura 2.31 siguiente se puede observar un perceptrón típico formado por una capa de entrada, una capa oculta y una de salida.

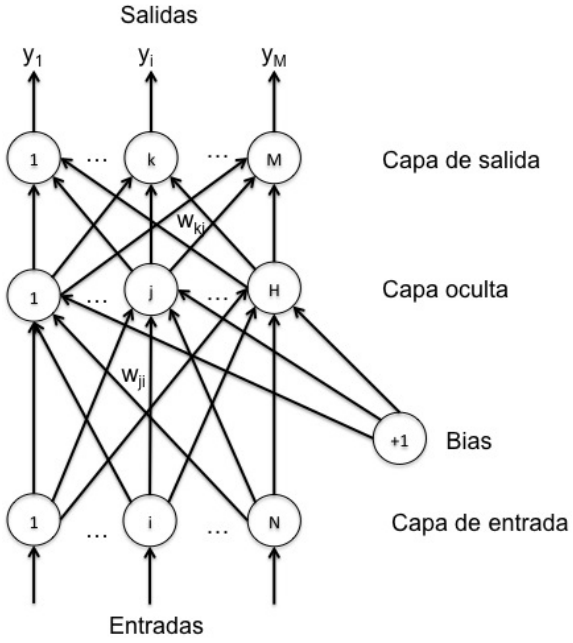


Figura 2.31: Estructura del perceptrón.

Las conexiones entre neuronas son siempre hacia delante: las conexiones van desde las neuronas de una determinada capa hacia las neuronas de la siguiente capa; no hay conexiones laterales ni conexiones hacia atrás. Por tanto, la información siempre se transmite desde la capa de entrada hacia la capa de salida. Como se mencionó anteriormente la función para solo una neurona (ecuación (2.9)), ahora se complementa para el perceptrón multicapa, en donde se considera w_{ji} como el peso de conexión entre la neurona de entrada i y la neurona oculta j, y v_{kj} como el peso de conexión entre la neurona oculta j y la neurona de salida k.

Fases en la aplicación de un perceptrón multicapa

Una red del tipo perceptrón multicapa intenta resolver dos tipos de problemas:

- **Problemas de predicción**: que consisten en la estimación de una variable continua de salida, a partir de la presentación de un conjunto de variables predictoras de entrada (discretas y/o continuas).
- **Problemas de clasificación**: que consisten en la asignación de la categoría

de pertenencia de un determinado patrón a partir de un conjunto de variables predictoras de entrada (discretas y/o continuas).

La red neuronal artificial que proponemos en el Capítulo 4 se enfoca en el problema de clasificación de flujos en una refinería en el cual se tendrá un flujo de masa que se desea saber si está en un estado de pérdida o no.

Algoritmo de retropropagación

Se considera una etapa de funcionamiento donde se presenta, ante la red entrenada, un patrón de entrada que se transmite a través de las sucesivas capas de neuronas hasta obtener una salida y, después, una etapa de entrenamiento o aprendizaje donde se modifican los pesos de la red de manera que coincida la salida deseada por el usuario con la salida obtenida por la red.

Selección de las variables relevantes y preprocesamiento de los datos

Para obtener una aproximación funcional óptima, se deben elegir cuidadosamente las variables a emplear: se trata de incluir en el modelo las variables predictoras que realmente predigan la variable dependiente o de salida, pero que a su vez no tengan relaciones entre sí ya que esto puede provocar un sobreajuste innecesario en el modelo. El rango de posibles valores debe ser aproximadamente el mismo y acotado dentro del intervalo de trabajo de la función de activación empleada en las capas ocultas y de salida de la red neuronal. Así, las variables de entrada y salida suelen acotarse en valores comprendidos entre 0 y 1 o entre -1 y 1.

En nuestra propuesta se usa una codificación entre -1 y 1 para la variable de entrada, y en la variable de salida como ya se encuentra en el rango de 0 y 1, decidimos utilizar el mismo rango. Además la función de activación que utilizamos es la sigmoidal que recibe valores entre -1 y 1 y da como salida valores entre 0 y 1.

2.5 Antecedentes de Pérdidas de Material en Refinerías

El problema de detección de pérdidas de material tiene como objetivo conocer el lugar en que se encuentra la pérdida en la red de tuberías descrita en un proceso. La metodología que proponemos es usada para la detección de pérdidas de petróleo, tratando múltiples fugas, además realizamos una modelación matemática del proceso, utilizamos teoría de grafos para la representación de la refinería, redes neuronales para clasificar los datos de operación de la red y realizamos una simulación de la refinería. A continuación se presentan algunos problemas resueltos en literatura, de los que se da una descripción de la metodología propuesta y se realiza una comparación con la nuestra.

Zhang y Xu (1999) utilizan la aplicación de ATMOS Pipe (sistema de detección de fugas, sin falsas alarmas) en una tubería de North Western Ethylene, para la compañía de Shell UK Limited. Para realizar esto plantean una red con 25 válvulas en la tubería la cual cuenta con medidores de flujo, presión, temperatura y densidad en la entrada y salida. La temperatura y presión son medidas antes y después de cada bloque. Estos datos se utilizan para afinar los datos y evitar falsas alarmas, cuando se opera a condiciones normales. Las funciones del sistema son: 1) colectar los datos de flujo, presión y temperatura en intervalos de 30 segundos, 2) validar los datos, 3) detectar las fugas en diferentes condiciones operacionales, estimar el tamaño y localización de las fugas, 4) almacenar datos y eventos históricos. Para optimizar el desempeño del sistema utilizaron 4 características: confiabilidad de los datos, robustez, sensibilidad y precisión. Mientras en nuestro estudio se detectan fugas múltiples de petróleo mediante una simulación, Zhang y Xu (1999) detectan fugas múltiples mediante una simulación, pero en sistemas de etileno y por medio de un software.

Verde (2001) diseñan un sistema de detección múltiple de fugas en tuberías. Usando sensores de flujo y presión en los extremos de la tubería. Proponen un modelo del fluido en la tubería con múltiples fugas en donde se consideran condiciones de frontera en las ecuaciones diferenciales parciales. Discretizan el dominio para obtener un estado no lineal. Observan que el análisis de detección de fugas con el modelo general muestra que el generador residual puede distinguir entre dos posiciones de fuga. Para enfrentarse a este problema sugieren integrar generadores de residuales al procedimiento. La evaluación de residuales y el diagnostico del proceso esta basado en la relación estática entre fugas y residuales, tomando en cuenta solo valores positivos de fugas. El objetivo es combinar dos tareas para detectar y localizar, mejorar el aislamiento y robustez de todo el sistema. La comparación de nuestro trabajo con el de (Verde, 2001) indica que en su trabajo no se hace uso de las redes neuronales artificiales para la detección de múltiples fugas de petroleo, sino que utilizan sensores y redundancia analítica, pero sí realizan una modelación matemática como ayuda para la detección.

Ahmad y Hamid (2003) proponen una red neuronal con prealimentación de un clasificador de fallas y que además sea capaz de detectar fugas por debajo del 1 % o menos del flujo nominal con la finalidad de detectar fugas en un proceso de fraccionamiento de petróleo. El proceso consiste en 5 columnas de destilación en serie. Los datos de entrada de la red neuronal son la presión del flujo y la temperatura. El método se da de forma jerárquica. Primero se estima el comportamiento normal de la red y después se detecta la fuga y el lugar en donde ocurre. En comparación con lo que proponemos en la tesis, (Ahmad y Hamid, 2003) también utilizan la metodología para detección de pérdidas así como el uso de redes neuronales, aunque solo tratan una fuga en la tubería.

da Silva et al. (2004) presentan la correlación que existe en la desviación de flujos de entrada y salida con las condiciones de operación. Mediante la ayuda de sistemas ruidosos y reglas, proponen identificar errores en el proceso. Ellos lo enfocan en monitoreo de gas licuado. Para esto disponen un sistema de inferencia ruidoso

usando el sistema de base regla compuesto de 3 módulos: 1) diseño de reglas ruidosas, 2) reconocimiento de estado y 3) evaluación de la desviación. En el módulo 1, se definen las variables y las funciones ruidosas de ingreso por medio de herramientas estadísticas. En los módulos 2 y 3 las reglas basadas en sistemas ruidosos clasifican el flujo e identifican los problemas de condición operacionales. En comparación de nuestro estudio que se centra en la detección de múltiples fugas de petróleo en un sistema, (da Silva et al., 2004) detectan pérdidas en tuberías de gas licuado con un sistema, mas tan sólo en una fuga.

Jian y Huaguang (2004) obtienen un método de detección de fugas y su localización mediante el uso de doble sensores de gradiente de presión en un tramo de tubería, involucrando ecuaciones diferenciales de presión que dependen de características de la tubería como la presión, el flujo másico, la densidad del fluido, la coordenada en x, coordenada en t, gravedad, área de sección transversal, diámetro de la tubería y coeficiente del factor de fricción. Mencionan que las variables que en el proceso no cambian con el tiempo son la presión y el flujo al inicio y al final de la tubería. Si la presión inicial, la presión final o la diferencia de entrada y salida sobrepasan de un rango establecido se observa el comportamiento de la red y se determina la existencia de fugas. Para localizar las fugas se agrega la variable de distancia a las ecuaciones diferenciales, además para fines de experimentación se agregan antes de cada tubería dos sensores de presión en el inicio y al final de la tubería. Donde los cambios de presión en la tubería ocurren antes y después de que aparezca la fuga. En comparación con la metodología que proponemos en la tesis, (Jian y Huaguang, 2004) realizan la modelación matemática y con la ayuda de sensores de gradiente instalados en cada extremo de la tubería para detectar una fuga en una sección de tubería.

Jian et al. (2006) muestran la metodología para la detección de fugas mediante el uso de redes neuronales competitivas, involucrando cinco parámetros (presión de entrada, flujo de entrada, presión de salida, flujo de salida y un delta de flujo). Además, exponen las ecuaciones que utilizará la red neuronal y el esquema para llevar

a cabo la detección. La estructura de la red consiste en diferentes componentes como son la entrada conectada a nodos de salida que involucran el peso del nodo de entrada al nodo de salida. Se determina la similitud o diferencia de los datos mediante la distancia euclidiana. Para la detección involucran señales de los sensores para generar vectores y procesar esta información para poder identificar si la red se encuentra en alguna de las clasificaciones propuestas: incremento de presión, decremento de presión, fuga o corrida normal. Las similitudes que presenta este trabajo con el nuestro es que ambos realizamos una modelación matemática y que utilizamos redes neuronales para detectar múltiples fugas de petróleo. La única diferencia sería que el trabajo de (Jian et al., 2006) no realiza una simulación.

Fuentes-Mariles et al. (2007) exponen un método con base en algoritmos genéticos para estimar y localizar fugas en una red de tuberías de agua potable, con datos obtenidos de la medición de presión de las tuberías, la estimación de las demandas y el conocimiento característico de la red (longitud tubería, diámetro, factor de fricción, etc.). En el método que proponen utilizan restricciones del principio de continuidad en los nodos, dejando así tres ecuaciones diferenciales en donde involucran los flujos de entrada y salida de cada nodo, además de incorporar las características principales de la tubería siendo como base del algoritmo genético planteado para encontrar los flujos y la localización de las fugas. La única similitud que este estudio guarda con el nuestro es que ambos realizamos una modelación matemática.

Begovich y Pizzano-Moreno (2008) trabajan con un algoritmo de detección de fugas para un prototipo de tubería de agua. Se obtienen datos reales de flujo y presión de cabezal de los sensores instalados en la alimentación y la salida. Se estima el tamaño de la fuga y su posición. Diseñan un prototipo de tubería para el problema. El algoritmo que usan para detectar las fugas es una modificación del de Billman con un estimador de factor de fricción adaptativo, que a su vez usa el Método de Características en lugar del esquema de diferencias finitas. Una vez que aparece una fuga, la estimación en línea del factor de fricción se debe detener, entonces proponen una manera de paro para la estimación del factor de fricción. Finalmente,

se agrega filtrado a las señales de entrada del algoritmo para aislar los datos ruidosos del prototipo. Al comparar el trabajo de (Begovich y Pizzano-Moreno, 2008) con el nuestro se aprecian algunas diferencias, estas son: detectan fugas de agua en una sección de tubería sin realizar una modelacion matemática y a cambio de de utilizar redes neuronales artificiales, modifican un algoritmo para detectar las fugas.

Mashford et al. (2009) estudian un método de minería de datos obtenido por la medición de sensores de presión en una red de tuberías con el fin de obtener información acerca de su localización y tamaño. La ingeniería inversa utilizada es llevada a cabo mediante reconocimiento de patrones, entrenados y validados mediante modelación hidráulica. El método usado para detectar el tamaño de fuga y predicción de su localización se monitorea la presión de los nodos en la red de tuberías y alimentando estos valores al reconocimiento de patrones se entrena para predecir el tamaño y localización de la fuga. Comprueban la efectividad de regresión en donde simulan una fuga en un nodo con 300 casos y miden su media de errores cuadrados. De la misma forma que nuestro trabajo, (Mashford et al., 2009) detectan múltiples fugas pero en el trasporte de agua, realizando una modelación matemática, sin embargo, utilizan otra herramienta de reconocimiento de patrones para la detección y localización de las fugas.

de Sousa et al. (2012) analizan el fluido de aceites pesados en ductos verticales con fugas mediante la modelación matemática y del análisis numérico. El modelo matemático considera el efecto de arrastre y fuerzas gravitacionales entre las fases y el flujo turbulento, dos dimensiones, flujo isotérmico y continuo, modelo no homogéneo para la mezcla de fluido y modelo homogéneo cuando existe turbulencia. El modelo homogéneo considera un solo campo para las dos fases mientras que el modelo no homogéneo considera un campo especifico para cada fase. Estos modelos involucran volumen, densidad, velocidad, presión, viscosidad, momento, fuerza de arrastre. Las similitudes que este estudio guarda con el nuestro son que ambos detectamos fugas de petróleo y realizamos una modelación matemática, pero (de Sousa et al., 2012) detectan pérdidas en un tramo de tubería mediante un análisis numérico en un

software.

Como puede observarse en la literatura reportada, los artículos donde se estudia la detección de pérdidas de material en la red de una refinería son prácticamente inexistentes o escasas. Asimismo, es un problema que ocurre durante el traslado de productos en refinerías y en otros tipos de industrias que afectan la economía del sector y además al medio ambiente, por lo que su solución se vuelve perentoria. Lo que se propone hacer ayudaría a conocer de manera rápida y eficaz el lugar en que se encuentra la fuga para actuar inmediatamente. En casos anteriores no se ha intentado resolver este tipo de problemas en toda una red de refinería, sino solo se ha implementado específicamente para tramos de tuberías o en tanques. Con la ayuda del algoritmo que se propone, se conocería en qué lugar hay una pérdida, lo que haría más eficaz la reducción y eliminación de pérdidas que lleguen a contaminar el medio ambiente. Además, es un problema real que se tiene dentro de una planta de refinación en México.

Capítulo 3

Generador de instancias

En este capítulo hablamos de la configuración de la red propuesta en base al comportamiento de las refinerías en donde el producto de entrada a la refinería se separa en diferentes compuestos como se muestra en el Capítulo 2. Además, se incluyen los supuestos que se realizan para llevar a cabo la simulación de la refinería basada en el algoritmo de Ford Fulkerson (Sección 2.1.3). La simulación nos permite la generación de instancias para adquirir una base de datos de las operaciones de los productos de refinación del petroleo. La base de datos incluye el flujo de masa en cada tramo de la tubería y su detección mediante el algoritmo de Ford Fulkerson Modificado. La salida de la simulación de la refinería sirve como entrada a la metodología propuesta en el Capítulo 4 en donde se realiza una detección de pérdidas de productos refinados mediante el uso de redes neuronales artificiales.

3.1 Descripción de la Refinería Propuesta

La configuración de la red de la refinería en este trabajo cuenta con la idea general de la operación de las refinerías. Proponemos una refinería con trece procesos y diecisiete conexiones, esto con el fin de ver el comportamiento de la refinería y en un futuro poder llevarlo a cabo en una refinería real. Con la ayuda de la teoría de grafos incluida en el Capítulo 2, procederemos a definir la configuración de la refinería propuesta.

La refinería se visualiza como una grafo dirigido conexo (red de transporte),

$G(N, A)$, que contiene un conjunto de trece nodos (N) y un conjunto de diecisiete arcos (A) con capacidades en cada arco (c_{ij}). En donde los procesos de la refinería son los nodos, las conexiones entre procesos son los arcos que unen a los nodos y las capacidades son las condiciones normales en las que opera cada conexión. El grafo $G(N, A)$ se puede observar en la Figura 3.1. Podemos observar que se tiene un nodo inicial, etiquetado con la letra s, en donde empieza el proceso, que envía un flujo que pasa por todo el grafo para llegar al nodo etiquetado como t donde recibe el flujo enviado. Esto corresponde a la definición de las redes de transporte en el Capítulo 2, por lo cual podemos encontrar el flujo que se puede enviar desde el nodo inicial s hasta el nodo final t, en condiciones normales de operación de la refinería. Para esto realizamos ciertas modificaciones del algoritmo de Ford-Fulkerson que se encuentra detallado en la Sección 2.1.3 y además presentamos el algoritmo modificado de Ford-Fulkerson usado en la tesis.

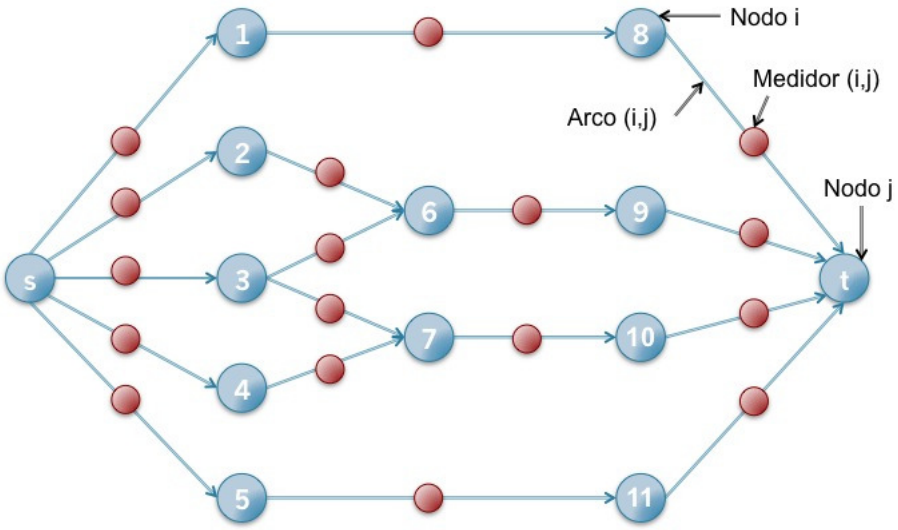


Figura 3.1: Diagrama de la red (caso de estudio).

Entre cada proceso se encuentra un pequeño círculo rojo, representando al medidor de tipo placa y orificio (para ver su funcionamiento revisar el Capítulo 2) proporcionando una medición de la presión diferencial del flujo que lo atraviesa. Con la ayuda de la ecuación típica de los medidores de placa y orifico (ecuación (2.8)), vista en el Capítulo 2, se transforma en flujo volumétrico (V, unidades m^3/s) y

haciendo uso de la ecuación de densidad ($\rho = m/V$, unidades kg/m^3) se obtiene el flujo másico (m, unidades kg/s).

Antes de empezar con el generador de instancias realizamos una serie de supuestos que nos ayudan a delimitar las características del grafo del caso de estudio. Con base a estos supuestos proponemos el algoritmo para realizar el generador de instancias para obtener los flujos de masa de la red y generar pérdidas en ella. En las siguientes secciones del Capítulo 3 se presentan lo supuestos realizados, el algoritmo propuesto para la simulación y los parámetros que se utilizan dentro del algoritmo.

3.2 Supuestos para del Algoritmo de Detección General

- Cada conexión existente en la refinería (tubería) entre dos procesos, se representa con un arco.
- Cada proceso de la refinería, se representa con un nodo.
- En cada arco de la red se tiene un medidor de placa y orificio, el cual mide la diferencia de presión con la cual se calcula el flujo de masa del arco.
- Cada arco cuenta con una capacidad mínima de flujo y una capacidad máxima.
- La densidad del petróleo es constante durante toda la red.
- Sólo se transporta petróleo en la red. No ocurre una separación de petróleo en diversos subproductos.
- El coeficiente de descarga, involucrado en la ecuación del medidor (ecuación (2.8)) es el mismo para cada medidor.

3.3 Algoritmo de Detección General

Con base a los supuestos realizados procedemos a explicar el algoritmo propuesto basado en modelos matemáticos y algoritmos de flujo en redes (Ford-Fulkerson) para realizar la generación de pérdidas en los arcos, así como de su correspondiente detección. La base de datos generada sirve como datos de entrada a la red neuronal artificial la cual utilizamos para la clasificación de los flujos de las tuberías en la refinería, expuesta en el Capítulo 4, con el fin de determinar si existe alguna pérdida a lo largo de la refinería. A continuación se presenta el algoritmo:

- Los datos de la red son leídos de un archivo de Excel, el cual se encuentra con fugas en alguno de sus arcos. Entre los datos se encuentran: condiciones de operación, conexiones, capacidades, diferencia de presión, densidad, coeficiente del medidor, etc.
- Calcular los flujos de masa para cada arco mediante la ecuación (2.8) y la información obtenida anteriormente.
- Generar fugas aleatoriamente en cada arco tomando como base los las condiciones de operación de la red (para detalles del algoritmo y pseudocódigo ir a la Sección 3.3.1).
- Realizar el Algoritmo de Ford Fulkerson Modificado (AFFM) (Sección 3.3.2), con el fin de encontrar todos los caminos que hay en la red, es decir, conocer la línea que lleva cada producto del proceso de refinación.
- Mientras se realiza el AFFM, en cada arco calculamos el error de medición, expandiéndose a lo largo de la red, con el fin de obtener un rango de aceptación en donde el flujo real se encuentra.
- Se comprueba para cada arco que el flujo calculado esté dentro del rango de aceptación, si se encuentra por debajo de este rango existe una fuga (1), si esta en el dentro rango o por encima, no existe fuga (0).

- Una vez obtenidos los flujos y detección de cada arco, estos son guardados en un documento, este proceso se repite un determinado número de veces, con el fin de generar una base de datos del comportamiento de la refinería y poder utilizarlo posteriormente.

- La base de datos obtenida la introducimos a un software estadístico llamado Matlab, el cual nos permite hacer un entrenamiento de la red neuronal artificial y encontrar la ecuación que nos permite hacer una clasificación suficientemente buena (Capítulo 4).

3.3.1 Algoritmo Generador de Fugas en la Red

Este algoritmo lo utilizamos para hacer una generación de fugas alrededor de la red de la refinería propuesta.

Damos como entrada una probabilidad de existencia de fugas en toda la red (P_{leak}), el vector de arcos de la red (A), el vector de nodos de la red (N), las matriz de capacidades de los arcos (c_{ij}) y la probabilidad en cada arco de que exista una fuga (P_{ij}^{leak}). Seguimos el siguiente procedimiento:

1. Como las capacidades son las condiciones de operación asignamos a una matriz que le llamaremos $Leak$.

2. Se obtiene un numero al azar (P_f) que comparamos con la probabilidad de entrada (P_{leak}).

3. Si P_f es mayor a P_{leak} existen fugas en la red y se continua con los siguientes pasos, por el contrario, si P_f es menor a P_{leak} no hay fuga en la red y se detiene el algoritmo.

4. Se obtiene al azar el número de arcos en donde hay fugas n_{leak}.

5. Como hay fuga en la red, se obtiene al azar un arco de la red, $[m, n]$, perteneciente al conjunto de arcos A.

6. Se obtiene un numero al azar (P^1_{leak}) que comparamos con la probabilidad de que exista fuga en ese arco elegido $[m, n]$ (P^{mn}_{leak}).

7. Se multiplica el valor en la matriz del arco correspondiente $Leak_{mn}$ por un valor aleatorio entre el 0.8 y 1.2 ($\pm 20\,\%$ de su operación).

8. Se elimina el arco elegido del conjunto de arcos y regresamos al punto 5 hasta que se llegue al numero de arcos obtenido donde hay fugas n_{leak}.

A continuación se muestra el pseudoalgoritmo utilizado en esta tesis que después implementamos en el lenguaje de programación Python.

Generador de fugas$(P_{leak}, A, N, c_{ij}, P^{ij}_{leak})$
$Leak_{ij} \leftarrow c_{ij}$
$P_f \leftarrow$ valor aleatorio entre (0,100)
if $P_f \geq P_{leak}$ **then**
  $n_{leak} \leftarrow$ valor aleatorio entre $(0, |N|)$
  $k \leftarrow 0$
  repeat
    $[m, n] \leftarrow$elegir valor aleatorio (i,j) $\in A$
    $P^1_{leak} \leftarrow$ valor aleatorio entre $(0, 100)$
    if $P^1_{leak} \geq P^{mn}_{leak}$ **then**
      $Leak_{mn} \leftarrow Leak_{mn}$*valor aleatorio entre 0.8 and 1.2
    end if
    $A \leftarrow A/\{Leak_{ij}\}$
    $k \leftarrow k + 1$
  until $k = n_{leak}$
end if
return $Leak_{ij}$

3.3.2 Algoritmo de Ford-Fulkerson Modificado

Tomando como base el algoritmo de Ford Fulkerson mostrado anteriormente se propone modificarlo para calcular la propagación del error en la red, encontrar los diferentes recorridos que hay en la red y generar la base de datos de los flujos de masa.

Al algoritmo se da como entrada el grafo $G(N, A)$ con su conjunto de nodos N y su conjunto de arcos A, el nodo de inicio s, el nodo final t, el diámetro del orificio presente en cada arco de la red D_0, el diámetro de la tubería en cada arco de la red D_1, la diferencia de presión obtenida del medidor en cada arco δP, el coeficiente de compresibilidad de cada medidor C_0 y los errores de cada medidor E puestos en 5 %. Seguimos el siguiente procedimiento:

1. Se calcula el flujo de operación una vez que se han realizado fugas en la red mediante la ecuación (2.8).
2. Se realizar el algoritmo de Ford Fulkerson.
3. Se inicializan los flujos máximos en cada arco f_{ij} .
4. Se inicializan los errores en cada arco ξ_{ij}, también los flujos máximos por error de medición $f_{max_{ij}}$ y los flujos mínimos por error de medición $f_{min_{ij}}$.
5. Se determina si existe un camino en el grafo.
6. Para cada arco en el camino se calcula el flujo máximo que se puede pasar por el camino f_{ij} y se determina el error de medición acumulado de cada arco en el camino ξ_{ij}. Se encuentra el flujo máximo $f_{max_{ij}}$ ocasionado por error de medición acumulado ξ_{ij} en cada arco y el flujo mínimo $f_{min_{ij}}$ ocasionado por error de medición acumulado ξ_{ij}.
7. Comparar el flujo de operación Q_{ij} calculado en el paso 1 con el flujo máximo $f_{max_{ij}}$ y el flujo mínimo $f_{min_{ij}}$. Si el flujo de operación Q_{ij} es mayor al flujo

máximo $f_{max_{ij}}$ entonces se tiene una ganancia en el arco. Si es menor al flujo mínimo $f_{max_{ij}}$ se tiene una fuga en el arco.

8. Repetir hasta que no haya caminos posibles en el grafo.

A continuación se muestra el pseudoalgoritmo propuesto en esta tesis que después implementamos en el lenguaje de programación Python.

procedure$(G(N,A), s, t, D_0, D_1, \Delta P, C_0, Em)$

$v_0 = \frac{C_{0_{ij}}}{\sqrt{1-\left(\frac{D_{0_{ij}}}{D_{1_{ij}}}\right)^4}} \sqrt{2\frac{\Delta P}{\rho_{ij}}} \ \forall (i,j) \in A$

$v_{ij} = v_{0_{ij}} \frac{\pi D_{0_{ij}}^2}{4} \ \forall (i,j) \in A$

$Q_{ij} = v_{ij}\rho_{ij} \ \forall (i,j) \in A$

Ford-Fulkerson(G, s, t)

$f_{ij} \leftarrow 0 \ \forall (i,j) \in A$

$\xi_{ij} \leftarrow 0 \ \forall (i,j) \in A$

$f_{max_{ij}} \leftarrow 0 \ \forall (i,j) \in A$

$f_{min_{ij}} \leftarrow 0 \ \forall (i,j) \in A$

while existe un camino p de s a t en G_f **do**

$c_{f_{ij}} \geq 0$

$c_f(p) \leftarrow min\{c_{f_{ij}} : (i,j) \ \epsilon \ p\}$

$(k,l) = \emptyset$

for (i,j) en p **do**

$f_{ij} \leftarrow f_{ij} + c_f(p)$

$f_{ji} \leftarrow f_{ji} - c_f(p)$

$\xi_{ij} \leftarrow \sqrt{E_{ij}^2 + \xi_{kl}^2}$

$(k,l) \leftarrow (i,j)$

$\xi_{max_{ij}} = f_{i,j}(1 + \xi_{ij})$

$\xi_{min_{ij}} = f_{i,j}(1 - \xi_{ij})$

if $Q_{ij} \geq \xi_{max_{ij}}$ **then**

"Ganancia en (i,j)"

else if $Q_{ij} \leq \xi_{min_{ij}}$ **then**

"Fuga en (i, j)"

end if

end for

end while

3.3.3 Elección de Parámetros del Algoritmo General

Los datos de lectura utilizados en el problema de estudio se detallan en la Tabla 3.1.

No. de Arco	Arco		Capacidad	Diferencial de Presión	Diámetro de Tubería	Diámetro de Orificio	Densidad Densidad	Coeficiente de compresibilidad
	Entrada	Salida	kg/s	Pa	m	m	kg/m3	
1	0	1	20	93300	0.1540	5.684E-2	878	0.61
2	0	2	20	93300	0.1540	5.684E-2	878	0.61
3	0	3	20	93300	0.1540	5.684E-2	878	0.61
4	0	4	20	93300	0.1540	5.684E-2	878	0.61
5	0	5	20	93300	0.1540	5.684E-2	878	0.61
6	1	8	20	93300	0.1540	5.684E-2	878	0.61
7	2	6	20	93300	0.1540	5.684E-2	878	0.61
8	3	6	10	86916	0.1548	4.105E-2	878	0.61
9	3	7	10	86916	0.1548	4.105E-2	878	0.61
10	4	7	20	93300	0.1540	5.684E-2	878	0.61
11	5	11	20	93300	0.1540	5.684E-2	878	0.61
12	6	9	30	97970	0.1535	6.840E-2	878	0.61
13	7	10	30	97970	0.1535	6.840E-2	878	0.61
14	8	12	20	93300	0.1540	5.684E-2	878	0.61
15	9	12	30	97970	0.1535	6.840E-2	878	0.61
16	10	12	30	97970	0.1535	6.840E-2	878	0.61
17	11	12	20	93300	0.1540	5.684E-2	878	0.61

Tabla 3.1: Datos de entrada al simulador de la refinería.

En la Tabla 3.1 la primer columna se refiere al número asignado a cada arco de la red. En la segunda columna se tiene el nodo del que sale cada arco. En la tercer columna se tienen los nodos al que entra cada arco en la red. En la cuarta columna el flujo de masa en kg/s, siendo las condiciones normales de operación que tiene cada arco. En la quinta columna se encuentra el diferencial de presión obtenida del medidor de placa y orificio en Pascales (Pa). En la sexta columna, el diámetro de la tubería para cada arco de la red, unidades en metros (m). En la séptima columna, el diámetro de orificio de la placa que se acopla al medidor, unidades en metros (m).

La octava columna, el coeficiente de compresibilidad característico de cada medidor, se fija en 0.61 ya que es uno de los valores más comunes en este tipo de medidores (Geankopolis, 1998), adimensional.

Los valores obtenidos mediante esta lectura se transformaron a flujos másicos a través de las ecuaciones (3.1), (3.2), (3.3) y (3.4).

$$v_0 = \frac{c_0}{\sqrt{\left(1 - \left(\frac{D_0}{D_1}\right)^4\right)}}, \sqrt{\frac{2(p_1 - p_2)}{\rho}}, \tag{3.1}$$

$$V = \frac{v_0 D_0^2}{4}, \tag{3.2}$$

$$\rho = \frac{m}{V}, \tag{3.3}$$

$$m = \rho V. \tag{3.4}$$

La ecuación (3.1) corresponde al cálculo de la velocidad tangencial en el área del medidor. En la ecuación (3.2), se utiliza el volumen de un cilindro para conocer el volumen por unidad de tiempo del fluido. La ecuación (3.3) es la ecuación de densidad, en la cual se despeja la masa quedando así como la ecuación (3.4), para conocer el flujo másico del fluido.

Por otra parte, como ayuda en la generación de pérdidas en los arcos de la red de transporte se generan números de manera uniforme dentro de un rango de 20 % por encima y por debajo de lo valores de condiciones operacionales leídas en la Tabla 3.1. Además, generamos un número aleatorio de arcos donde se crea la pérdida, teniendo así las diferentes condiciones de operación de la refinería, las cuales incluyen: pérdidas en cada arco de la red, sin pérdidas en los arcos y pérdidas entre un arco y todos los arcos. Al realizar este algoritmo en múltiples ocasiones, nos permite tener el comportamiento de una refinería a través del tiempo, simulando lo que ocurre dentro de una refinería real.

Capítulo 4

Métodos de Solución para Detección de Pérdidas en Refinerías

En este capítulo hablamos de la configuración y los parámetros de la red neuronal artificial (RNA) propuesta con base a la salida de la base de datos (flujos de masa y su correspondiente detección) generada mediante la simulación, que representa el comportamiento de la refinería durante un determinado tiempo. Esta generación de la base de datos se muestra en el Capítulo 3.

4.1 Descripción de la Red Neuronal Artificial

Recordando la teoría de redes neuronales artificiales que se presenta en el Capítulo 2 y la generación de la base de datos de los flujos de masa, incluyendo pérdidas, en refinerías que se realiza en el Capítulo 3. Realizamos una red neuronal con la estructura del perceptrón multicapa, ya que se ha visto que tiene buenas soluciones en la clasificación de señales y en predicción de datos (Marín, 2010), en nuestro caso se utiliza para realizar clasificación. Como se presenta en el Capítulo 2 en la teoría de redes neuronales multicapa, el perceptrón multicapa consta de tres capas: capa de entrada, capa oculta y capa de salida.

La red de la refinería propuesta consta de diecisiete arcos, nuestra propuesta

de red neuronal consta de diecisiete nodos en la capa de entrada, un número desconocido, por el momento, de neuronas en la capa oculta, éste valor se explica más adelante cómo se obtuvo, y diecisiete nodos en la capa de salida. Una vez que tenemos la estructura de la red neuronal, esta tiene que ser entrenada con un conjunto de datos o una base de datos que nos permita conocer el comportamiento de los datos para poder llevar a cabo una clasificación eficiente. La base de datos que utilizamos para entrenar a la red neuronal será la obtenida del simulador de flujos de masa. Los valores de la capa de entrada son los flujos de masa obtenidos en cada arco y los valores en las neuronas de la capa de salida deseados (valores objetivo) son la detección preliminar de cada arco. La estructura de la red neuronal se observa en la Figura 4.1

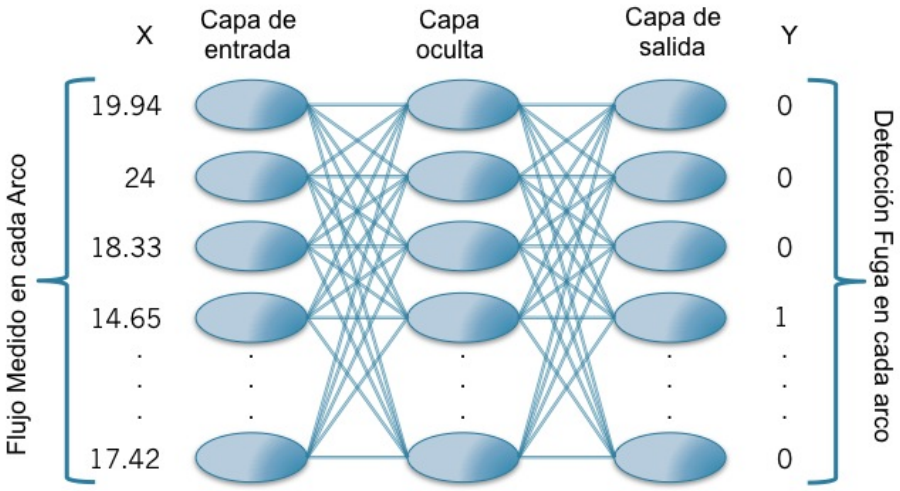


Figura 4.1: Red Neuronal Artificial. Ambas, las capas de entrada y salida tienen diecisiete nodos

4.2 Parámetros de la Red Neuronal Artificial

En el Capítulo 2 vimos que las redes neuronales tienen gran cantidad de parámetros en las etapas de procesamiento para obtener la respuesta deseada, entre los que se encuentran: el número de neuronas en la capa oculta, pesos iniciales de

la red, velocidad de entrenamiento, función de activación para las neuronas en la capa oculta, entre otras. En éste trabajo fijamos la mayoría de los parámetros en sus valores por defecto. Los parámetros modificados son: 1) la función de activación en la capa oculta y en la capa de salida para el entrenamiento de la red neuronal; 2) la función de entrenamiento de la red neuronal; 3) el porcentaje de separación de datos en entrenamiento, validación y prueba; y 4) el número de neuronas en la capa oculta.

4.2.1 Entrenamiento de la Red Neuronal Artificial

Elección de los pesos iniciales

Hacemos una asignación de pesos pequeños generados de forma aleatoria, en un rango de valores entre -0.5 y 0.5 o algo similar. Para que tengamos una diversificación de la solución y consigamos (mediante diferentes inicios de los pesos del algoritmo de retropropagación) encontrar un valor mejor en la función objetivo (Marín, 2010).

Elección de la arquitectura de la red

Respecto a la elección de la arquitectura de la red, se sabe que para la mayoría de problemas prácticos bastará con utilizar una sola capa oculta (Marín, 2010). El número de neuronas de la capa de entrada está determinado por el número de variables predictoras. El número de neuronas de la capa de salida está determinado bajo el mismo esquema que en el caso anterior. Cuando intentamos discriminar entre dos categorías, bastará con utilizar una única neurona. El número de neuronas ocultas determina la capacidad de aprendizaje de la red neuronal. Recordando el problema del sobreajuste, debemos usar el mínimo número de neuronas ocultas con las cuales la red rinda de forma adecuada. Esto se consigue evaluando el rendimiento de diferentes arquitecturas en función de los resultados obtenidos con el grupo de validación.

La red neuronal artificial propuesta en esta tesis consta 17 neuronas en la capa de entrada (número de arcos presentes en la red de la refinería), 17 neuronas en la

capa de salida y para determinar el número de neuronas en la capa oculta realizamos la evaluación de diferentes arquitecturas. Dicha evaluación se presenta en el Capítulo 4 y los resultados de la evaluación en el Capítulo 5.

Elección de la tasa de aprendizaje y factor momento

El valor de la tasa de aprendizaje (η) controla el tamaño del cambio de los pesos en cada iteración de la metodología de la red neuronal (algoritmo de retropropagación). Se deben evitar dos extremos: un ritmo de aprendizaje demasiado pequeño puede ocasionar una disminución importante en la velocidad de convergencia y la posibilidad de acabar atrapado en un mínimo local; en cambio, un ritmo de aprendizaje demasiado grande puede conducir a inestabilidades en la función de error, lo cual evitará que se produzca la convergencia debido a que se darán saltos en torno al mínimo sin alcanzarlo (Marín, 2010).

Por tanto, se recomienda elegir un ritmo de aprendizaje lo más grande posible sin que provoque grandes oscilaciones. En general, el valor de la tasa de aprendizaje suele estar comprendido entre 0.05 y 0.5 (Marín, 2010).

El factor momento (α) acelera la convergencia de los pesos. Suele tomar un valor próximo a 1 (por ejemplo, 0.9) (Marín, 2010). Nosotros utilizamos este valor.

Función de activación de las neuronas ocultas y de salida

Una red neuronal típica se puede caracterizar por la función de base f y la función de activación. Cada nodo suministra un valor y_j a su salida. Este valor se propaga a través de la red mediante conexiones unidireccionales hacia otros nodos de la red. Asociada a cada conexión hay un peso sináptico denominado w_{ij}, que determina el efecto del nodo j-ésimo sobre el nodo i-ésimo. Las entradas al nodo i-ésimo que provienen de las otros nodos son acumulados junto con el valor umbral δ_i, y se aplica la función base f, obteniendo u_i. La salida final y_i se obtiene aplicando la función de activación sobre u_i.

La función de base tiene dos formas típicas:

- Función lineal: El valor de red es una combinación lineal de las entradas,

$$u_i(w, x) = \sum jw_{ij}x_j. \tag{4.1}$$

- Función radial: es una función de base de segundo orden no lineal. El valor de red representa la distancia a un determinado patrón de referencia,

La función que utilizamos en ésta tesis es la función lineal debido a su simplicidad de cálculo.

El valor de red, expresado por la función de base, $u(w, x)$, se transforma mediante una función de activación no lineal. Hay dos formas básicas que cumplen esta condición: la función lineal (o identidad) y la función sigmoidal (logística o tangente hiperbólica). Solo se habla de la función sigmoidal ya que fue la que fue utilizada en esta tesis.

- Función sigmoidal

$$f(u_i) = \frac{1}{1 + \exp -\frac{u_i}{\sigma^2}}. \tag{4.2}$$

Donde $f(u_i)$ es la función sigmoidal, u_i es la entrada a la función y σ^2 es la varianza de los datos.

Para aprovechar la capacidad de las redes neuronales de aprender relaciones complejas o no lineales entre variables, se recomienda la utilización de funciones no lineales al menos en las neuronas de la capa oculta (Marín, 2010). Por tanto, en general se utiliza una función sigmoidal (logística o tangente hiperbólica) como función de activación en las neuronas de la capa oculta. En tareas de clasificación, las neuronas normalmente toman la función de activación sigmoidal. En cambio, en tareas de predicción o aproximación de una función, generalmente las neuronas toman la función de activación lineal. La función sigmoidal (4.2) se utiliza como parámetro de la red neuronal artificial propuesta en nuestra metodología presentada en el Capítulo 4, ya que los valores que se obtienen de esta función están en un rango

de 0 a 1. Además, la tarea impuesta para la red neuronal artificial es de clasificación de flujos en los arcos de la red de la refinería.

Evaluación del rendimiento del modelo

Una vez seleccionado el modelo de red cuya configuración de parámetros ha obtenido la mejor ejecución ante el conjunto de validación, debemos evaluar la capacidad de generalización de la red de una forma completamente objetiva a partir de un tercer grupo de datos independiente, el conjunto de prueba.

Cuando se trata de estimar una función, normalmente utilizamos la media cuadrática del error para evaluar la ejecución del modelo (Marín, 2010).

En problemas de clasificación de patrones es mejor la frecuencia de clasificaciones correctas e incorrectas. Se puede construir una tabla de confusión y calcular diferentes índices de asociación y acuerdo entre el criterio y la decisión tomada por la red neuronal (Marín, 2010).

Recopilación de elección de parámetros

Los detalles de los parámetros que elegimos para la red neuronal se presentan a continuación:

- Ecuación de la red neuronal.
- Función de activación de la capa oculta: Logarítmica sigmoidal.
- Función de activación de la capa de salida: Lineal.
- Función de entrenamiento: Levenberg-Marquardt (obtiene buenos resultados en clasificación (Demuth et al., 2009)).
- Porcentaje de separación: 80 % Entrenamiento, 10 % Validación y 10 % Prueba.
- Número de neuronas ocultas: parte experimental que utilizamos para obtener su valor en el cual se obtienen buenos resultados.

En la etapa de entrenamiento la superficie de respuesta generada por el algoritmo se presentan óptimos locales, por lo que se deben de utilizar estrategias que nos permitan salir de los óptimos locales con el fin de llegar al óptimo global, pero estas estrategias no aseguran llegar a los óptimos globales. Además, se conoce que los pesos iniciales de la red neuronal impacta de manera significativa el resultado obtenido en la clasificación o predicción de los valores deseados con respecto a los valores obtenidos por la red neuronal.

Por estos motivos decidimos repetir varias veces el entrenamiento de la red neuronal, es decir, se inicia el peso de todas las conexiones de la red neuronal, con el fin de escapar de los óptimos locales y obtener mejores soluciones. Para el problema decidimos realizar veinte repeticiones, aproximadamente el 120 % del número de neuronas en la capa de entrada, en este caso diecisiete. Este valor fue tomado por recomendación del Dr. Mauricio Cabrera quien tiene gran experiencia en el uso de redes neuronales artificiales.

Una vez que elegimos los parámetros, hacemos un preprocesamiento en donde los datos de entrada se normalizan a valores entre -1 y 1, para no causar problemas de dimensiones en las variables. Como los valores objetivo solo cuentan con valores de 0 y 1 no se requiere normalizarlos.

Capítulo 5

Resultados Experimentales

En este capítulo se muestran los resultados de las herramientas presentadas en el Capítulo 3 que proponemos para resolver el problema presentado en la tesis. Las herramientas son: la generación de instancias, que incluye el simulador de la refinería dando de salida una base de datos de los flujos de masa en los arcos vista en el Capítulo 3 y la segunda es la clasificación posterior de la base de datos mediante redes neuronales artificiales vista a detalle en el Capítulo 4.

La experimentación se realizó en una computadora Macbook Pro 10.6.8, con dos procesadores de 2.26 GHz Quad-Core Intel Xeon, memoria RAM de 12Gb 1066 MHz DDR3 Ram y 531.37 GB de disco duro. La simulación, detección preliminar de fugas y determinación de condiciones de operación se realizaron en el lenguaje de programación Python 2.7.2, mediante el compilador de la consola en la misma computadora. Para la clasificación mediante redes neuronales se utilizó el software comercial Matlab 2009 con la ayuda del paquete para redes neuronales versión 6 (Demuth et al., 2009).

5.1 Generador de Instancias

El generador de instancias nos permite conocer el comportamiento de la refinería, por lo tanto mediante el algoritmo de Ford Fulkerson Modificado que proponemos, obtenemos una base de datos que incluye el flujo de masa generado en cada arco de la red y la generación de perdidas en los mismos, por lo tanto, tenemos para

cada arco su correspondiente valor dependiendo de si el arco cuenta con pérdida o no (1 si se tiene pérdida, 0 caso contrario).

Por otra parte, las características de la salida obtenida del generador de instancias es de la siguiente manera:

- El rango de arcos que pueden tener fugas se encuentra entre 0 y 17 arcos.
- Cada arco en la red podrá generar, dependiendo de si es elegido o no, una perdida entre el 80 % y 120 % de su flujo de masa en condiciones de operación normal.

Para obtener el desempeño de la generación de instancias realizamos el algoritmo cien veces. Obteniéndose que en promedio hay 6.135 arcos con pérdida (en el arco de la refinería), esto es el promedio cuando se realiza el algoritmo, el número de arcos que cuentan con pérdida es de seis. Las pérdidas son detectadas realizándose un balance en cada arco de la red y se comprueba si el flujo se encuentra por debajo de las condiciones de operación para determinar si existe un arco con fuga en la red.

Arco	Flujo de Masa	Detección de fuga
1	20	0
2	20	0
3	20	0
4	16.41	1
5	16.41	1
6	20	0
7	20	0
8	10	0
9	10	0
10	16.41	1
11	16.41	1
12	30	0
13	16.41	1
14	20	0
15	30	0
16	16.41	1
17	16.41	1

Tabla 5.1: Ejemplo salida de la base de datos de flujos y su correspondiente detección de fugas.

En la Tabla 5.1 mostramos una representación de la salida del generador de instancias, solo para una repetición para fines prácticos. La primer columna contiene el numero del arco en la red de la refinería. La segunda columna el flujo de masa correspondiente a cada arco en kg/m^3. La tercer columna corresponde a la detección de fugas en el arco con respecto a si se encuentra fuera del rango aceptado de operación. Lo que se muestra es una representación ya que la salida se obtiene en un vector donde los primeros elementos del vector son los flujos de masa seguidos de la detección de si tiene fuga en el arco o no (ejemplo: $20, 20, 20, ..., 16.41, 16.41, 0, 0, ..., 1, 1.$)

Además, como se menciona en el Capítulo 3, el simulador de la refinería, tiene la capacidad de generar diferentes condiciones de operación de la refinería. Puede generar desde cero arcos con pérdida hasta diecisiete arcos con perdida, incluyendo todos los arcos intermedios (un arco con pérdida, dos arcos con pérdida, ...).

5.2 Clasificador de los Flujos en la Refinería

En esta sección, hablamos de la experimentación que se realizó para tratar la salida obtenida por el generador de instancias (base de datos de flujos de masa y su correspondiente detección). Como se vio en el Capítulo 5 utilizamos la red neuronal artificial para clasificar la salida del generador de instancias.

Primero realizamos la experimentación de los parámetros que se utilizan para la red neuronal artificial, en donde el parámetro que varía es el número de neuronas ocultas, el cual nos permite ajustar los valores de mejor manera, pero tenemos que tener cuidado de no utilizar un número muy grande porque se sobreajusta el modelo.

Para conocer el número de neuronas en la capa oculta, se varía el número de neuronas en la capa oculta en un rango de 1 a 10 con incrementos de 1 $(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$, y en otra experimentación, en un rango de 20 a 60 con incrementos de 10 $(20, 30, 40, 50, 60)$, haciendo un total 15 variaciones del número de neuronas en la capa oculta. Además, la base de datos generada para la experimentación es de 100 pares de datos (flujo de masa en cada arco y detección correspondiente a cada arco).

Como se vió en el Capítulo 2, los valores de entrada, para el entrenamiento, de la red neuronal artificial se separan en tres conjuntos: entrenamiento, validación y prueba. Además, como se vio en el Capítulo 3 el porcentaje de cada conjunto es: 80 % de los valores de entrada para entrenamiento, 10 % Validación y 10 % prueba.

Para medir el desempeño de la respuesta otorgada por la red neuronal artificial en cada prueba, de las 15 variaciones realizadas, se toman dos criterios de evaluación:

1) la sumatoria de los errores cuadrados de la diferencia entre los valores de salida objetivo y los valores de salida de la red neuronal (Figura 5.1); y 2) el porcentaje de acierto de cada uno de los 100 conjuntos de 17 entradas y 17 salidas (Figura 5.2), que calculamos de la diferencia de valores de salida objetivo contra los valores obtenidos por la red neuronal dividiéndolo entre el numero total de salidas en cada conjunto y obteniéndose un promedio de todos los conjuntos. Los resultados obtenidos se presentan a continuación.

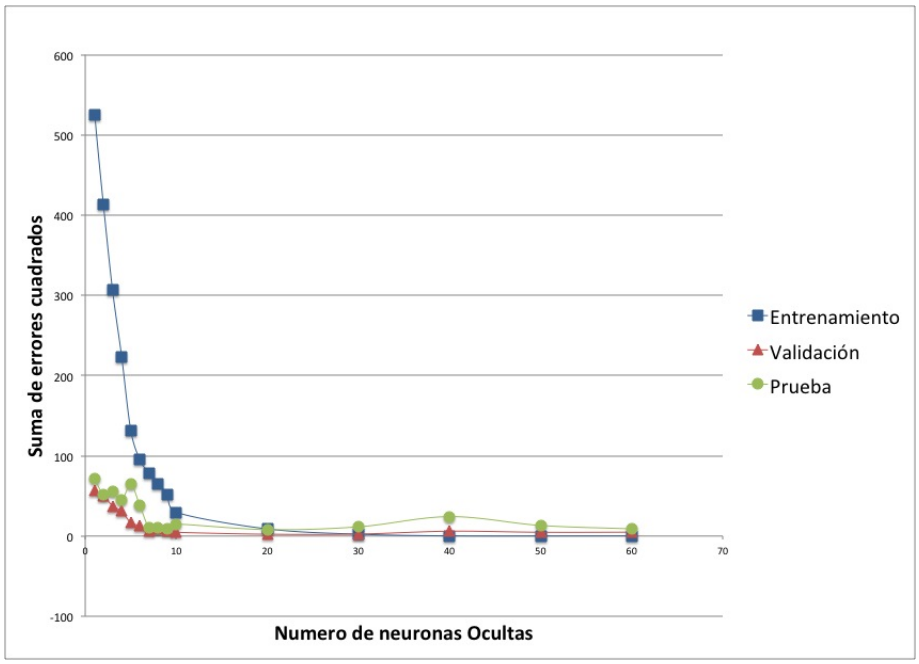


Figura 5.1: Suma de los errores cuadrados entre los valores objetivo y los valores clasificados por la red neuronal.

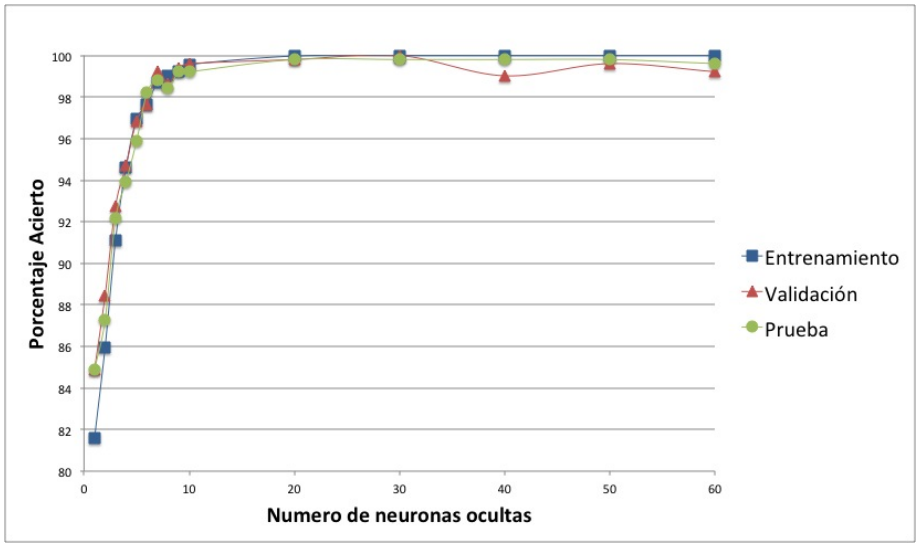


Figura 5.2: Porcentaje de aciertos del método propuesto para la clasificación del flujo en la red neuronal.

En la Figura 5.1 tenemos en el eje de las x el número de neuronas en la capa oculta, y en el eje de las y la suma de los errores cuadrados entre los valores objetivo y los valores clasificados por la red neuronal. Los cuadrados representan los datos que se encuentran en el conjunto de entrenamiento, los triángulos representan los datos del conjunto de validación y los círculos representan los datos del conjunto de prueba. Además, podemos observar que al aumentar en número de neuronas en la capa oculta disminuye el la sumatoria de errores cuadrados entre la salida deseada y la salida de la red neuronal, como es de esperar.

En la Figura 5.2 tenemos en el eje de las x el número de neuronas en la capa oculta, y en el eje de las y el porcentaje de aciertos del método propuesto para la clasificación del flujo en la red neuronal. Además, observamos como al aumentar en número de neuronas en la capa oculta aumenta el porcentaje de aciertos en la clasificación de los flujos de masa en cada arco, como es de esperar. Además, tanto en la Figura 5.1 como en la Figura 5.2 observamos que al llegar a las 10 neuronas en la capa oculta se obtiene buen resultado y permanece constante al aumentar el número de neuronas en la capa oculta. Por lo que se decide que el número de neuronas en

la capa oculta sea fijado a 10 neuronas.

Una vez ajustado el parámetro del número de neuronas en la capa oculta procedemos a realizar la experimentación del rendimiento que tiene la red neuronal para clasificar los flujos de los arcos y poder determinar si en el arco existe o no una perdida, para datos que no han sido vistos por la red neuronal. Para esto generamos una base de datos de 1000 valores con la cual se compara la salida de la red neuronal artificial cuando se entrena con diferentes cantidades de base de datos. Utilizamos cuatro conjuntos para el entrenamiento de la red neuronal: el primer conjunto contiene una base de datos de 100 valores obtenidos en el generador de instancias; el segundo conjunto contiene 232 valores obtenidos de los experimentos suficientes para conocer el comportamiento de un proceso, en el cual se varían las variables en tres niveles. Los niveles para cada variable (arco) son los flujos máximos que se pueden tener, calculados en base al error en cada arco y las condiciones de operación; el tercer conjunto involucra al primer y segundo conjunto dando un total de 332 valores de entrada y salida; y el cuarto conjunto se obtienen 768 conjuntos del generador de instancias para complementan los 332 valores de entrada y salida, y obtener 1000 valores para el entrenamiento de la red neuronal. Los resultados obtenidos de la experimentación se presentan en las Figuras 5.3, 5.4, 5.5 y 5.6.

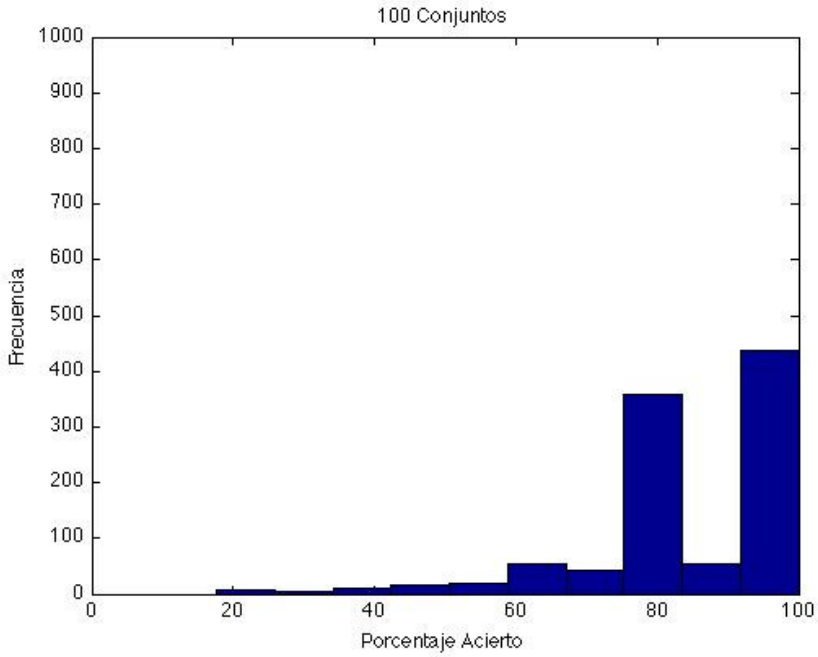


Figura 5.3: Comportamiento de la red neuronal al predecir 1000 valores de prueba, mediante 100 conjunto de datos para entrenar la red neuronal.

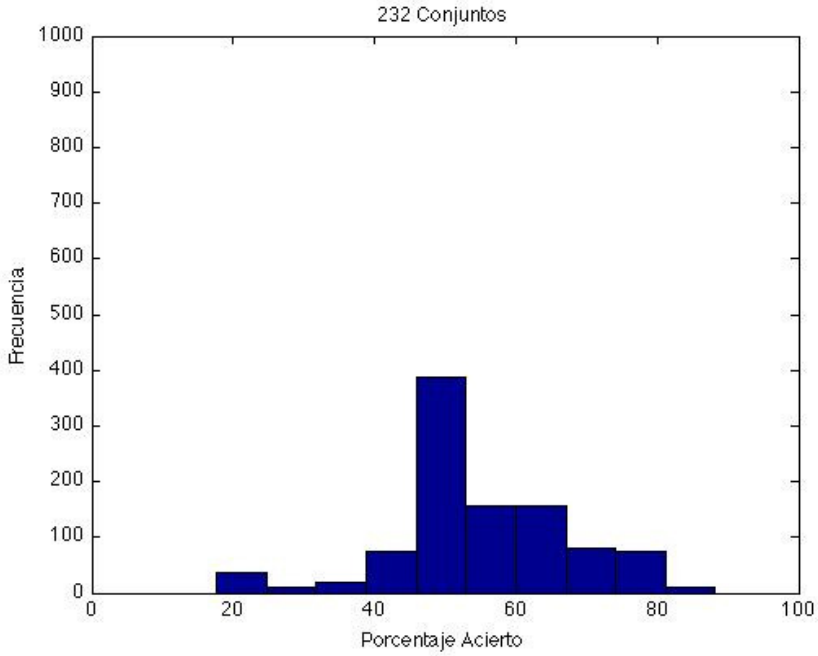


Figura 5.4: Comportamiento de la red neuronal al predecir 1000 valores de prueba, mediante 232 conjunto de datos para entrenar la red neuronal.

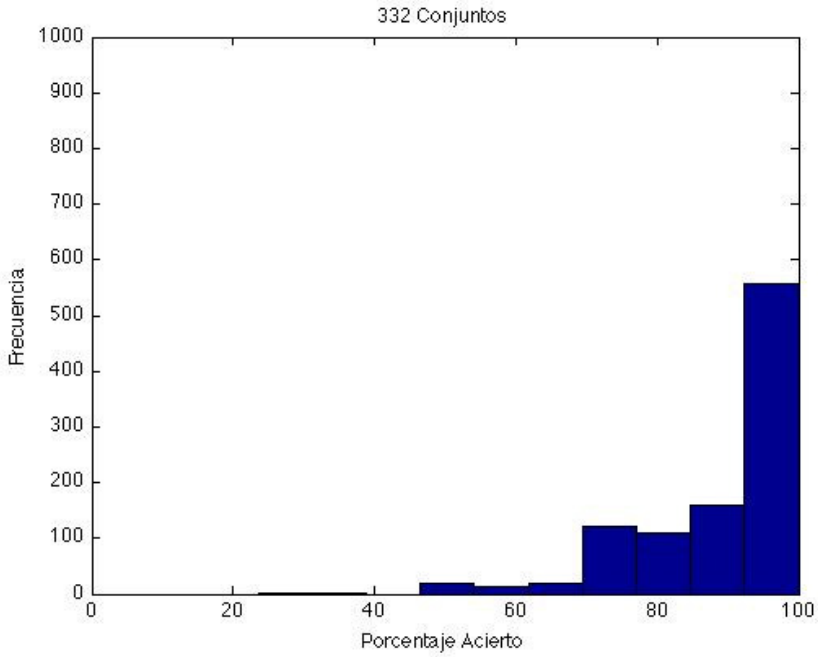


Figura 5.5: Comportamiento de la red neuronal al predecir 1000 valores de prueba, mediante 332 conjunto de datos para entrenar la red neuronal.

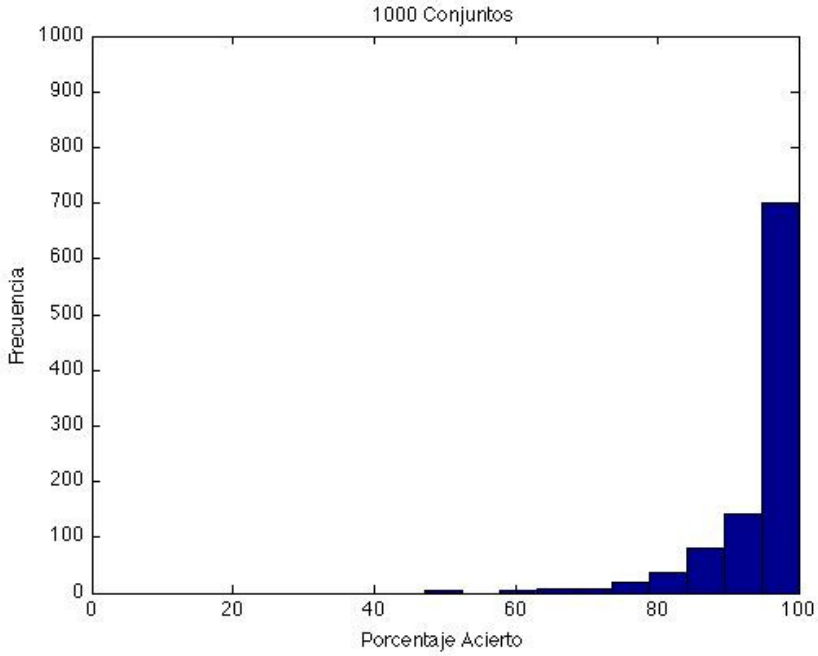


Figura 5.6: Comportamiento de la red neuronal al predecir 1000 valores de prueba, mediante 1000 conjunto de datos para entrenar la red neuronal.

En la Figura 5.3 se tiene un histograma en donde el eje x es el porcentaje de acierto y el eje y es la frecuencia con la que aparece. La Figura 5.3 corresponde a la prueba de clasificación para el conjunto de 100 valores. Observamos que el rango de clasificación entre 10 % y 100 %, y si queremos obtener un 100 % de clasificación solamente el 35 % de los datos llega a este valor. Era de esperar que no se consiguiera el 100 % de clasificación ya la red neuronal artificial que proponemos no ha tenido tantos datos de entrada para hacer una buena clasificación.

En la Figura 5.4 se tiene un histograma en donde el eje x es el porcentaje de acierto y el eje y es la frecuencia con la que aparece. La Figura 5.4 corresponde a la prueba de clasificación para el conjunto de 232 valores. Observamos que el rango de clasificación entre 20 % y 80 %. Si queremos obtener un 100 % de clasificación no lo conseguiríamos ya que ninguna prueba llego a obtener un 100 % de clasificación. Se empeora la respuesta porque se introdujo a la red neuronal artificial datos que se encuentran en los extremos de operación de los flujos en los arcos de la red de la refinería y este no se adapta a los pequeños cambios que pueda haber de flujos.

En la Figura 5.5 se tiene un histograma en donde el eje x es el porcentaje de acierto y el eje y es la frecuencia con la que aparece. La Figura 5.5 corresponde a la prueba de clasificación para el conjunto de 332 valores. Observamos que el rango de clasificación entre 40 % y 100 %, y si queremos obtener un 100 % de clasificación solamente el 45 % de los datos llega a este valor. Aumenta el porcentaje de clasificación correcta como era de esperar ya que ahora tiene mas datos de entrada que tienen diferentes condiciones para comparar con los datos que queremos clasificar.

En la Figura 5.6 se tiene un histograma en donde el eje x es el porcentaje de acierto y el eje y es la frecuencia con la que aparece. La Figura 5.6 corresponde a la prueba de clasificación para el conjunto de 1000 valores. Observamos que el rango de clasificación entre 10 % y 100 %, y si queremos obtener un 100 % de clasificación solamente el 75 % de los datos llega a este valor. Se obtiene un muy buen porcentaje de clasificación al entrenarse la red con un mayor numero de elementos, ya que se le añaden nuevos valores (nuevas condiciones de operación) y permiten a la red

neuronal tener menos error al hacer la clasificación.

Como vemos al ir aumentando el número de datos que damos de entrada a la red neuronal artificial se obtiene un mayor porcentaje de clasificación correcta. Como es de esperar ya que tiene más variedad de condiciones de operación en las que se basa la red neuronal artificial.

Capítulo 6

Conclusiones

Por último en esta sección mostramos las conclusiones derivadas de los métodos propuestos y los resultados de la experimentación realizada. En la Sección 6.1 presentamos las conclusiones generales de esta tesis, en la Sección 6.2 las contribuciones realizadas y el trabajo futuro para este problema lo presentamos en la Sección 6.3.

6.1 Conclusiones

- Comprobamos que es posible detectar fugas en proceso de refinamiento del petróleo usando modelos gráficos que incorporan datos de balance de masa para obtener una clasificación suficientemente precisa.
- El generador de instancias de la refinería propuesto para asemejar a una refinería genera en promedio 6.135 fugas a lo largo de la red en un rango de 0 a 17 fugas. Por lo tanto, el generador de instancias tiene diferentes condiciones de operación de la refinería en donde podemos o no tener pérdidas en la red.
- La red neuronal propuesta obtiene buenos resultados de clasificación ya que está alrededor del 99 % siendo un buen resultado y solamente necesita de 10 neuronas en la capa oculta. Con el fin de obtener mejores porcentajes de acierto en la red de refinería, se necesita un número mayor de datos para entrenar a la red neuronal.

- Al utilizar red neuronal se obtiene un buen desempeño para la clasificación de los flujos de masa en los arcos de una red para detectar las fugas.
- Como es de esperar, al entrenar la red neuronal con un mayor número de datos se obtiene un mayor porcentaje de acierto y un menor error cuadrado entre los valores objetivo y los valores producidos por la red neuronal.

6.2 Contribuciones

- Desarrollamos dos herramientas como ayuda en la detección de fugas en refinerías. La primera herramienta nos permite hacer una simulación simplificada de una refinería en la cual se generan fugas aleatoriamente en los arcos de la red y nos ayuda además a tener una detección preliminar de las fugas. La segunda herramienta que utilizamos para clasificación en donde se obtuvieron buenos resultados para la detección de fugas para el caso de estudio de la refinería en estado estable.
- El método propuesto es de gran utilidad para resolver el problema de detección de fugas debido a su práctica implementación y aplicación.

6.3 Trabajo Futuro

- Agregar al generador de instancias más parámetros que nos permitan conocer el comportamiento real de una refinería.
- Realizar una simulación en donde involucren los diferentes compuestos que tienen las refinerías, ya que por el momento se esta tomando en la red el petróleo.
- Extender el problema a diferentes procesos en donde se puedan involucrar medidores del tipo placa y orifico para utilizar la metodología propuesta.

- Comparar el método propuesto con diferentes metodologías existentes y crear un mapa en donde podamos dar qué métodos son convenientes dependiendo del número de arcos que tenga la red y del número de procesos.

Índice de figuras

Índice de tablas

Apéndice A

Código Computacional

A.1 Código General de Detección de Fugas en Refinerías

```
1  ##************************************************************************
2  ##                          PISIS FIME UANL                               *
3  ##************************************************************************
4  ## Nombre del Programa: DeteccionDeFugas                                   *
5  ## Descripcion        : Programa que ayuda a la deteccion de fugas        *
6  ##                      en la red de una refineria                        *
7  ##                      Los pasos a seguir son:                           *
8  ##                      -Leer los parametros requeridos en una hoja de    *
9  ##                       Excel                                            *
10 ##                      - Se declaran e inicializan las variables         *
11 ##                        utilizadas en este algoritmo.                   *
12 ##                      - Se generan las fugas de manera aleatoria en     *
13 ##                        los arcos de la red                             *
14 ##                      - Se realiza el algoritmo de Ford-Fulkerson, con  *
15 ##                        el fin de conocer los diferentes caminos que    *
16 ##                        hay en la red asi como calcular los errores en  *
17 ##                        los medidores.                                  *
18 ##                      - Realizar un Balance de Materia a lo largo de    *
19 ##                        la red.                                         *
20 ##                      - Determinar si existe alguna fuga en el nodo     *
21 ##                        donde se lleva a cabo el balance.               *
22 ##                      - Determinar si existe una fuga en alguno de los  *
23 ##                        arcos de la red.                                *
24 ##                      - Imprimir los datos de balances de nodos con     *
25 ##                        sus respectivo deteccion de fugas en el nodo,   *
26 ##                        asi como los flujos en los arcos y su           *
27 ##                        respectiva deteccion.                           *
```

```
28 #* Funcional          : Luis Alejandro Benavides Vazquez               *
29 #* Programador        : Luis Alejandro Benavides Vazquez               *
30 #* Fecha Creacion     : 1 Mayo del 2013 (fecha tentativa)              *
31 ************************************************************************
32 #*                        LOG DE MODIFICACIONES                       *
33 ************************************************************************
34 #* Descripcion        : Modificacion de programa base para estandares. *
35 #*                      Estandarizar el codigo para su mejor lectura y *
36 #*                      modificaciones posteriores                     *
37 #*                                                                     *
38 #* Funcional          : Luis Alejandro Benavides Vazquez               *
39 #* Programador        : Luis Alejandro Benavides Vazquez               *
40 #* Fecha Creacion     : 28 Agosto del 2013                             *
41 ************************************************************************
42 ************************************************************************
43 #* Descripcion        : Agregar funciones para mejor entendimiento     *
44 #*                      Funciones agregadas para separar la funcion:   *
45 #*                      eliminar_flujos                                *
46 #*                      - EliminarArcos                                *
47 #*                      - ObtenerFlujoYFugaMinimoDeCadaArcoPorCamino   *
48 #*                      - ObtenerErrorMinimoYMaximoEnCadaArcoDelCamino *
49 #*                      Ademas se agregaron las siguientes funciones.  *
50 #*                      - ObtenerFlujoMasicoMaximoYMinimoEnCadaArco    *
51 #*                      - BalanceDeMateriaPorNodo                      *
52 #*                      - DeteccionDeFugasEnNodos                      *
53 #*                      - DeteccionDeFugasEnArcos                      *
54 #*                      - ImprimirResultados                           *
55 #*                      Con el fin de depurar el codigo y dejarlo      *
56 #*                      mas estructurado, pasa poder realizar          *
57 #*                      modificaciones posteriores                     *
58 #* Funcional          : Luis Alejandro Benavides Vazquez               *
59 #* Programador        : Luis Alejandro Benavides Vazquez               *
60 #* Fecha Creacion     : 29 Agosto del 2013                             *
61 ************************************************************************
62 ************************************************************************
63 #* Descripcion        : Modificacion de programa DeteccionDeFugas      *
64 #*                      Utilizar el codigo para hacer el diseno de     *
65 #*                      experimentos.                                  *
66 #*                      Se agregaron las variables:                    *
67 #*                      - FlujoMasicoPorRepeticion                     *
68 #*                      - FlujoMasicoNivelPorRepeticion                *
69 #*                      - DeteccionFugasPorRepeticion                  *
70 #*                      Se agrego la parte de impresion para diseno    *
71 #*                      de experimentos                                *
72 #* Funcional          : Luis Alejandro Benavides Vazquez               *
```

```
#* Programador        : Luis Alejandro Benavides Vazquez                *
#* Fecha Creacion     : 18 Septiembre del 2013                          *
#**********************************************************************

#&====================================================================
#& Importar archivos y funciones
#&====================================================================
from __future__ import division
from time import time
import copy
import math
import random
from LecturaDeDatos import *
from FuncionesParaDeteccion import *
import sys

#&====================================================================
#& Leer variable de terminal
#&====================================================================
opcion = int(sys.argv[1])

#&====================================================================
#& Inicializar Tiempo
#&====================================================================
t1=time()

#&====================================================================
#& Declaracion de variables globales
#&====================================================================
FlujoMasicoEnCadaArcoPorCadaCamino = []             #Almacenara los valores de
    flujo
                                                    #de cada arco de todos los
                                                        caminos
FlujoFugaEnCadaArcoPorCadaCamino = []               #Almacenara los valores de
    flujo de
                                                    #la fuga de cada arco de todos
                                                        los
                                                    #caminos
TodosLosCaminos = []                                #Almacenara todos los caminos
                                                    #encontrados
FlujoMasicoMaximoEnCadaArco = []                    #Almacenara los valores de
    flujo de
                                                    #maximo de cada arco
FlujoMasicoMinimoEnCadaArco = []                    #Almacenara los valores de
    flujo de
```

```
112                                                 #minimo de cada arco
113 FlujoMasicoMaximoEnCadaArcoPorCamino = []       #Almacenara los valores de
        flujo de
114                                                 #maximo de cada arco de todos
                                                        los
115                                                 #caminos
116 FlujoMasicoMinimoEnCadaArcoPorCamino = []       #Almacenara los valores de
        flujo de
117                                                 #minimo de cada arco de todos
                                                        los
118                                                 #caminos
119 ErrorMaximo = []                                #Almacenar los errores maximos
120                                                 #de cada medidor
121 ErrorMinimo = []                                #Almacenar los errores minimos
122                                                 #de cada medidor
123 ErrorAcumuladoMaximo = []                       #Almacenar los errores
        acumulados
124                                                 #maximos de cada medidor
125 ErrorAcumuladoMinimo = []                       #Almacenar los errores
        acumulados
126                                                 #minimos de cada medidor
127 FlujoMasicoMaximo = []                          #Almacenar los flujos masicos
128                                                 #maximos de cada medidor
129 FlujoMasicoMinimo = []                          #Almacenar los flujos masicos
130                                                 #minimos de cada medidor
131 FlujoMasicoConFugas = []                        #Almacenar los flujos masicos
132                                                 #en cada arco con las fugas
133                                                 #generadas
134
135 FlujoMasicoPorRepeticion = []
136 FlujoMasicoNivelPorRepeticion = []
137 DeteccionFugasPorRepeticion = []
138
139 BalancePorNodo = []
140 BalanceMaximoPorNodo = []
141 BalanceMinimoPorNodo = []
142 FugaEnNodo = []
143 FugaEnArco = []
144
145 FlujoMaximo = 0
146 FlujoFugasMaximo = 0
147
148 #&=====================================================================
149 #& Declaracion de variables locales
150 #&=====================================================================
```

```
151 tempArcos = []                                          #ALmacena los arcos
152 tempCapacidades = []                                    #Almacena las capacidades
153 tempFlujoMasicoConFugas =[]
154 flujoMasicoEnCadaArcoDelCamino = []                    #Almacenara los valores de
        flujo
155                                                         #de cada arco en un camino
156 flujoFugaEnCadaArcoDelCamino = []                       #Almacenara los valores de
        flujo
157                                                         #de cada arco en un camino
158 ArcosParaElegirDondeAbraFuga = []                       #Arcos donde puede haber fugas
159 numeroDeFugas = 0
160 indiceArcoElectoParaFuga = 0
161 arcoConFuga = 0
162
163 #&==========================================================================
164 #& Inicializar varibles
165 #&==========================================================================
166 tempArcos = copy.copy( Arcos )
167 tempCapacidades = copy.copy( Capacidades )
168 FlujoMasicoConFugas = copy.copy( FlujoMasico )
169
170 s = Nodos[0]                    #Asignar a s (nodo suministro) el primer valor de
        la
171                                 #lista de nodos
172 t = Nodos[NumeroDeNodos-1]      #Asignar a t (nodo terminal) el ultimo valor de la
173                                 #lista de nodos
174 u = 1                           #Nodo al cual visitara [a,u] donde a es el nodo
        anterior
175 k = 0                           #Indice que nos servira para saber cuantos caminos
176                                 #existen
177 B = []                          #Lista para comparar la lista de Arcos,
178                                 #ya que se iran eliminando los elementos de la
179                                 #lista de Arcos
180
181
182 #&==========================================================================
183 #& Repeticiones de flujo para Diseno de Experimentos
184 #&==========================================================================
185 FlujoMasicoPorRepeticion, FlujoMasicoNivelPorRepeticion,
        DeteccionFugasPorRepeticion = ConvertirNivelesAFlujo(NumeroDeArcos,
        NumeroDeRepeticiones, NivelPorRepeticion, FlujoMasicoConFugas)
186
187 #&==========================================================================
188 #& Generador de Fugas
189 #&==========================================================================
```

```
190 #Probabilidad de que ocurra una fuga
191 #pf = random.randint(0,100)
192 numeroDeFugas = random.randint( 1, len(FlujoMasicoConFugas) )   #Numero de arcos
        que
193                                                                 #tendran fugas
194 for i in range(0,len(FlujoMasicoConFugas)):
195     ArcosParaElegirDondeAbraFuga.append(i)
196
197 for i in range(0,numeroDeFugas):
198     indiceArcoElectoParaFuga = random.randint(0,len(ArcosParaElegirDondeAbraFuga)
            -1)
199     arcoConFuga = ArcosParaElegirDondeAbraFuga[indiceArcoElectoParaFuga]
200     for j in range( 0, len(FlujoMasicoConFugas) ):
201         if arcoConFuga == j:
202             ArcosParaElegirDondeAbraFuga.pop( indiceArcoElectoParaFuga )
203             FlujoMasicoConFugas[j] = FlujoMasicoConFugas[j] * random.uniform(
                    0.80,1.2 )
204
205
206 #&=====================================================================
207 #& Almacenar matriz de flujo con fugas generadas en los arcos
208 #&=====================================================================
209 tempFlujoMasicoConFugas = copy.copy( FlujoMasicoConFugas )
210
211 #&=====================================================================
212 #& Algoritmo
213 #&=====================================================================
214 while (Arcos != B):
215     Camino = []     #Inicializar camino
216     while (s != t): #Ciclo para cada camino
217         #m=len(tempA)
218         if [s,u] in Arcos: #Si el arco [s,u] se encuentra en A
219             Camino.append( [s,u] )
220             s = u     #cambiar nodo anterior, para continuar el camino
221         u = u + 1
222     #Fin ciclo para camino
223     s = Nodos[0]  #Inicializar nuevamente el nodo s
224     u = 0     #Inicializar el nodo siguiente u
225
226
227     TodosLosCaminos.append(Camino)
228     flujoMinimo = ValorMinimoCamino(Camino, Arcos,Capacidades)          #Flujo que
            hay en
229                                                                         #el camino
```

```
230     flujoFugasMinimo = ValorMinimoCamino(Camino, Arcos, FlujoMasicoConFugas)   #
            Flujo de fugas
231                                                                                 #que hay en
                                                                                        el
232                                                                                 #camino
233     #Flujo maximo
234     FlujoMaximo += flujoMinimo
235
236     #Fujo maximo fugas
237     FlujoFugasMaximo += flujoFugasMinimo
238
239     #Agregado
240     Capacidades, Arcos = EliminarArcos(Camino, Arcos, Capacidades, flujoMinimo)
241
242     flujoMasicoEnCadaArcoDelCamino, flujoFugaEnCadaArcoDelCamino =
            ObtenerFlujoYFugaMinimoDeCadaArcoEnElCamino(Camino, flujoMinimo,
            flujoFugasMinimo)
243
244     ErrorAcumuladoMaximo, ErrorAcumuladoMinimo =
            ObtenerErrorMinimoYMaximoEnCadaArcoDelCamino(Camino)
245
246     #Calculo Flujo E max y Flujo e min
247     FlujoMasicoMaximo = MultiplicacionDeListas(flujoMasicoEnCadaArcoDelCamino,
            ErrorAcumuladoMaximo)
248     FlujoMasicoMinimo = MultiplicacionDeListas(flujoMasicoEnCadaArcoDelCamino,
            ErrorAcumuladoMinimo)
249
250     FlujoMasicoEnCadaArcoPorCadaCamino.append(flujoMasicoEnCadaArcoDelCamino)
251     FlujoMasicoMaximoEnCadaArcoPorCamino.append(FlujoMasicoMaximo)
252     FlujoMasicoMinimoEnCadaArcoPorCamino.append(FlujoMasicoMinimo)
253     FlujoFugaEnCadaArcoPorCadaCamino.append(flujoFugaEnCadaArcoDelCamino)
254
255     k += 1        #Se agrega 1 para seguir el otro camino
256
257 #&=====================================================================
258 #& Restaurar variable Arcos y capacidades
259 #&=====================================================================
260 Arcos = copy.copy( tempArcos )
261
262 #&=====================================================================
263 #& Flujo de masa maximo y minimo en cada Arco
264 #&=====================================================================
265 FlujoMasicoMaximoEnCadaArco, FlujoMasicoMinimoEnCadaArco =
        ObtenerFlujoMasicoMaximoYMinimoEnCadaArco(Arcos, TodosLosCaminos,
        FlujoMasicoMaximoEnCadaArcoPorCamino, FlujoMasicoMinimoEnCadaArcoPorCamino)
```

```
266
267 #&=========================================================================
268 #& Balance de Materia en la red y deteccion (Normal, Minimo, Maximo)
269 #&=========================================================================
270 BalancePorNodo, BalanceMaximoPorNodo, BalanceMinimoPorNodo =
        BalanceDeMateriaPorNodo(NumeroDeNodos, NumeroDeArcos, Arcos, FlujoMasicoConFugas
        , FlujoMasicoMaximoEnCadaArco, FlujoMasicoMinimoEnCadaArco)
271
272 #&=========================================================================
273 #& Deteccion de fugas en cada nodo
274 #&=========================================================================
275 FugaEnNodo = DeteccionDeFugasEnNodos(NumeroDeNodos, BalancePorNodo,
        BalanceMaximoPorNodo, BalanceMinimoPorNodo)
276
277 #&=========================================================================
278 #& Deteccion de fugas en cada arco
279 #&=========================================================================
280
281 FugaEnArco, FlujoMasico = DeteccionDeFugasEnArcos(TodosLosCaminos,
        FlujoMasicoMaximoEnCadaArcoPorCamino, FlujoMasicoMinimoEnCadaArcoPorCamino,
        FlujoFugaEnCadaArcoPorCadaCamino, tempArcos, FlujoMasico)
282
283 #&=========================================================================
284 #& Elegir modo de impresion
285 #& 0 - Imprimir Tiempo
286 #& 1 - Imprimir datos de Deteccion de fugas
287 #& 2 - Imprimir datos de diseno experimentos
288 #&=========================================================================
289
290 if (opcion == 1):
291     #&=====================================================================
292     #& Impresion resultados de deteccion
293     #&=====================================================================
294     ImprimirResultados(NumeroDeNodos, NumeroDeArcos, BalancePorNodo, FugaEnNodo,
            FlujoMasico, FugaEnArco)
295
296 elif (opcion == 2):
297     #&=====================================================================
298     #& Impresion resultados de diseno experimentos
299     #&=====================================================================
300     ImprimirResultadosDiseno(NumeroDeArcos, NumeroDeRepeticiones,
            FlujoMasicoPorRepeticion,FlujoMasicoNivelPorRepeticion,
            DeteccionFugasPorRepeticion)
301
302 elif (opcion == 0):
```

```
303     #&=========================================================================
304     #& Finaliza Tiempo de ejecucion
305     #&=========================================================================
306     print "Tiempo total algoritmo: ",time()-t1
```

A.2 Lectura de Datos

```
1  #*************************************************************************
2  #*                          PISIS FIME UANL                              *
3  #*************************************************************************
4  #* Nombre del Programa: LecturaDeDatos                                   *
5  #* Descripcion        : Lectura de los parametros de la hoja de excel    *
6  #*                                                                       *
7  #*                      Leer los parametros requeridos en una hoja de    *
8  #*                      Excel. Siendo los siguientes:                    *
9  #*                      - Numero de Arcos                                *
10 #*                      - Numero de Nodos                                *
11 #*                      - Arcos de la red                                *
12 #*                      - Capacidades en los arcos                       *
13 #*                      - Diferenciales de presion en cada arco          *
14 #*                      - Diametro de la tuberia                         *
15 #*                      - Diametro del orificio del medidor              *
16 #*                      - Densidad del compuesto                         *
17 #*                      - Coeficiente del medidor                        *
18 #*                                                                       *
19 #*                      Ademas se realiza el calculo del flujo masico    *
20 #*                      de cada arco en base a los parametros leidos     *
21 #*                      de la hoja de Excel                              *
22 #*                                                                       *
23 #* Funcional          : Luis Alejandro Benavides Vazquez                 *
24 #* Programador        : Luis Alejandro Benavides Vazquez                 *
25 #* Fecha Creacion     : 1 Mayo del 2013 (fecha tentativa)                *
26 #*************************************************************************
27 #*                        LOG DE MODIFICACIONES                          *
28 #*************************************************************************
29 #* Descripcion        : Modificacion de programa base para estandares.   *
30 #*                      Estandarizar el codigo para su mejor lectura y   *
31 #*                      modificaciones posteriores                       *
32 #*                                                                       *
33 #* Funcional          : Luis Alejandro Benavides Vazquez                 *
34 #* Programador        : Luis Alejandro Benavides Vazquez                 *
35 #* Fecha Creacion     : 28 Agosto del 2013                               *
36 #*************************************************************************
```

```
37 ##************************************************************************
38 #* Descripcion        : Se agrego la parte de repeticiones para el      *
39 #*                      diseno experimental                             *
40 #*                                                                      *
41 #* Funcional          : Luis Alejandro Benavides Vazquez                *
42 #* Programador        : Luis Alejandro Benavides Vazquez                *
43 #* Fecha Creacion     : 18 Septiembre del 2013                          *
44 ##************************************************************************
45
46 #&=====================================================================
47 #& Importar funciones
48 #&=====================================================================
49 from FuncionesParaDeteccion import CalculoDeFlujoMasico  #Funcion para
50                                                          #calcular el flujo
51                                                          #masico
52 import xlrd                                          #Permite leer archivos de Excel
53
54 #&=====================================================================
55 #& Declaracion de variables globales
56 #&=====================================================================
57 Arcos = []                                           #Arcos de la red
58 Nodos = []                                           #Nodos en la red
59 Capacidades = []                                     #Capacidades de los arcos, m3/s
60 DeltaP = []                                          #Diferencial de presion leida
       del
61                                                      #medidor, Pa, N/m2
62 D1 = []                                             #Diametro de la tuberia, m
63 D0 = []                                             #Diametro del Orificio, m
64 rho = []                                            #Densidad del compuesto, kg/m3
65 C0 = []                                             #Coeficiente caracteristico del
66                                                      #medidor
67 FlujoMasico = []                                    #Flujo masico del arco, kg/s
68
69 #&=====================================================================
70 #& Lectura de datos de la hoja de Excel
71 #&=====================================================================
72 Libro = xlrd.open_workbook( "Tesis.xlsx" )          #Abrir el archivo que contiene
       la
73                                                      #informacion de la red de
                                                            refineria
74 Hoja1 = Libro.sheet_by_index( 0 )                   #Acceder a la hoja que contiene
        la
75                                                      #informacion, en este caso
                                                            Hoja1
76
```

```
77 NumeroDeNodos = int( Hoja1.cell_value( rowx = 0, colx = 1 ) )  #Lectura del Numero
       de Nodos
78 NumeroDeArcos = int( Hoja1.cell_value( rowx = 1, colx = 1 ) )  #Lectura del Numero
       de Arcos
79
80 for i in range(0,NumeroDeArcos):
81     a1=int(Hoja1.cell_value(rowx=5+i, colx=1))         #Lectura de Arco de salida,
           i
82     a2=int(Hoja1.cell_value(rowx=5+i, colx=2))         #Lectura de Arco de entrada,
            j
83     Arcos.append([a1,a2])                            #Almacenar arco (i,j)
84     c1=Hoja1.cell_value(rowx=5+i, colx=3)              #Lectura de Capacidad en el
           arco
85     Capacidades.append(c1)                                    #Almacenar capacidad
           c(i,j)
86     dp1=Hoja1.cell_value(rowx=5+i, colx=4)             #Lectura de diferencial de
           presion
87     DeltaP.append(dp1)                               #Almacenar diferecial de
           presion
88                                                      #deltaP(i,j)
89     d1=Hoja1.cell_value(rowx=5+i, colx=5)              #Lectura del diametro de
           tuberia
90     D1.append(d1)                                    #Almacenar diametro D1(i,j)
91     d0=Hoja1.cell_value(rowx=5+i, colx=6)              #Lectura del diametro de
           orificio
92     D0.append(d0)                                    #Almacenar diametro D0(i,j)
93     r1=Hoja1.cell_value(rowx=5+i, colx=7)              #Lectura de densidad
           promedio en el
94                                                      #arco
95     rho.append(r1)                                   #Almacenar densidad rho(i,j)
96     c0=Hoja1.cell_value(rowx=5+i, colx=8)              #Lectura de coeficiente
           medidor
97     C0.append(c0)                                    #Almacenar coeficiente C0(i,j)
98
99 #&=====================================================================
100 #& Lectura de datos diseno experimental
101 #&=====================================================================
102 Hoja2 = Libro.sheet_by_index( 1 )                    #Acceder a la hoja que contiene
        la
103 #informacion, en este caso Hoja1
104 NumeroDeRepeticiones = 232
105
106
107 NivelPorRepeticion = []
108 for i in range(NumeroDeRepeticiones):
```

```
109     temp = []
110     for j in range(NumeroDeArcos):
111         elemento = int(Hoja2.cell_value(rowx=1+i, colx=j))
112         temp.append(elemento)
113     NivelPorRepeticion.append(temp)
114 
115 #&=====================================================================
116 #& Inicializar la varible Nodos
117 #&=====================================================================
118 for i in range(0,NumeroDeNodos):
119     Nodos.append(i)                                  #Almacenar el numero de cada
        nodo
120 
121 #&=====================================================================
122 #& Calculo de los flujos masicos en cada arco
123 #&=====================================================================
124 for i in range( 0,NumeroDeArcos ):
125     FlujoMasico.append( CalculoDeFlujoMasico( DeltaP[i], D1[i], D0[i], rho[i], C0[i
        ] ) )
```

A.3 Funciones para Detección de Fugas en Refinerías

```
1  ************************************************************************
2  **                          PISIS FIME UANL                           *
3  ************************************************************************
4  ** Nombre del Programa: FuncionesParaDeteccion                        *
5  ** Descripcion        : Recopilatorio de la funciones que se utilizaran *
6  **                      en el programa para ayuda en la deteccion de    *
7  **                      fugas en la red de refineria                    *
8  **                      Leer los parametros requeridos en una hoja de   *
9  **                      Excel                                           *
10 **                      Se programan las funciones requeridas en el     *
11 **                      algoritmo para deteccion de fugas, las          *
12 **                      funciones que se incluyen son las siguientes:   *
13 **                      - Raiz                                          *
14 **                      - Numero de Nodos                               *
15 **                      - Arcos de la red                               *
16 **                      - Capacidades en los arcos                      *
17 **                      - Diferenciales de presion en cada arco         *
18 **                      - Diametro de la tuberia                        *
19 **                      - Diametro del orificio del medidor             *
```

```
20 ##                      - Densidad del compuesto                        *
21 ##                      - Coeficiente del medidor                       *
22 ##                                                                      *
23 ##                      Ademas se realiza el calculo del flujo masico   *
24 ##                      de cada arco en base a los parametros leidos    *
25 ##                      de la hoja de Excel                             *
26 ##                                                                      *
27 ## Funcional          : Luis Alejandro Benavides Vazquez                 *
28 ## Programador        : Luis Alejandro Benavides Vazquez                 *
29 ## Fecha Creacion     : 1 Mayo del 2013 (fecha tentativa)                *
30 ************************************************************************
31 ##                        LOG DE MODIFICACIONES                         *
32 ************************************************************************
33 ## Descripcion        : Modificacion de programa base para estandares.  *
34 ##                      Estandarizar el codigo para su mejor lectura y  *
35 ##                      modificaciones posteriores                      *
36 ##                                                                      *
37 ## Funcional          : Luis Alejandro Benavides Vazquez                 *
38 ## Programador        : Luis Alejandro Benavides Vazquez                 *
39 ## Fecha Creacion     : 28 Agosto del 2013                               *
40 ************************************************************************
41 ************************************************************************
42 ## Descripcion        : Agregar funciones para mejor entendimiento       *
43 ##                      Funciones agregadas para separar la funcion:    *
44 ##                      eliminar_flujos                                 *
45 ##                      - EliminarArcos                                 *
46 ##                      - ObtenerFlujoYFugaMinimoDeCadaArcoPorCamino    *
47 ##                      - ObtenerErrorMinimoYMaximoEnCadaArcoDelCamino  *
48 ##                      Ademas se agregaron las siguientes funciones.   *
49 ##                      - ObtenerFlujoMasicoMaximoYMinimoEnCadaArco     *
50 ##                      - BalanceDeMateriaPorNodo                       *
51 ##                      - DeteccionDeFugasEnNodos                       *
52 ##                      - DeteccionDeFugasEnArcos                       *
53 ##                      - ImprimirResultados                            *
54 ##                      Con el fin de depurar el codigo y dejarlo       *
55 ##                      mas estructurado, pasa poder realizar           *
56 ##                      modificaciones posteriores                      *
57 ## Funcional          : Luis Alejandro Benavides Vazquez                 *
58 ## Programador        : Luis Alejandro Benavides Vazquez                 *
59 ## Fecha Creacion     : 29 Agosto del 2013                               *
60 ************************************************************************
61 ************************************************************************
62 ## Descripcion        : Agregar funciones para el diseno de experimentos*
63 ##                      para convertir datos a variables -1 y 1         *
64 ##                      y de impresion de estos valores                 *
```

```
65 #*                      - ConvertirNivelesAFlujo                          *
66 #*                      - ImprimirResultadosDiseno                        *
67 #* Funcional          : Luis Alejandro Benavides Vazquez                  *
68 #* Programador        : Luis Alejandro Benavides Vazquez                  *
69 #* Fecha Creacion     : 18 Septiembre del 2013                            *
70 ************************************************************************
71 #&====================================================================
72 #& Importar archivos
73 #&====================================================================
74 from __future__ import division
75 import math
76
77 #&====================================================================*
78 #&  Nombre de la Funcion/metodo: Raiz
79 #&  Descripcion: Se obtiene la raiz cuadrada de un numero.
80 #&====================================================================*
81 #&  Entrada <-  x, cualquier valor dentro de os numeros racionales
82 #&  Salida  ->  raiz cuadrada de x
83 #&====================================================================*
84 def RaizCuadrada(x):
85     return math.sqrt(x)
86
87 #&====================================================================*
88 #&  Nombre de la Funcion/metodo: SumaDeLista
89 #&  Descripcion: Suma todos los elementos de una lista.
90 #&====================================================================*
91 #&  Entrada <-  lista
92 #&  Salida  ->  suma
93 #&====================================================================*
94 def SumaDeLista(lista):
95     suma = 0
96     for i in range( 0, len(lista) ):
97         suma = suma + lista[i]
98     return suma
99
100 #&====================================================================*
101 #&  Nombre de la Funcion/metodo: SumaCuadradaDeLista
102 #&  Descripcion: Suma el cuadrado de cada elemento de una lista de
103 #&  valores, hasta el indice de la lista deseado.
104 #&====================================================================*
105 #&  Entrada <-  lista
106 #&              indice, valor hasta el cual se sumara la lista
107 #&  Salida  ->  SumaDeRaizCuadrada
108 #&====================================================================*
109 def SumaCuadradaDeLista(lista, indice):
```

```
110     suma = 0
111     for i in range( 0, indice ):
112         suma = suma + lista[i] ** 2
113     sumaRaizCuadrada = math.sqrt( suma )
114     return sumaRaizCuadrada
115 
116 #&=====================================================================*
117 #&  Nombre de la Funcion/metodo: ValorAcumuladoDeLista
118 #&  Descripcion: Se realiza una suma acumulada de los valores de la lista
119 #&               de cada indice, se llama a la funcion SumaCuadradaDeLista
120 #&               para regresar una lista con los valores acumulados hasta
121 #&               cada indice.
122 #&=====================================================================*
123 #&  Entrada <-  lista
124 #&  Salida  ->  listaAcumulada
125 #&=====================================================================*
126 def ValorAcumuladoDeLista(lista):
127     sum_acum = 0
128     listaAcumulada = []
129     for i in range( 0, len(lista) ):
130         sumaAcumulada = SumaCuadradaDeLista( lista, i + 1 )
131         listaAcumulada.append( sumaAcumulada )
132     return listaAcumulada
133 
134 #&=====================================================================*
135 #&  Nombre de la Funcion/metodo: MultiplicacionDeListas
136 #&  Descripcion: Multipica cada elemento de la lista 1 por su
137 #&               correspondiente elemento de la lista 2.
138 #&=====================================================================*
139 #&  Entrada <-  lista1, lista2
140 #&  Salida  ->  listaMultiplicada
141 #&=====================================================================*
142 def MultiplicacionDeListas(lista1, lista2):
143     listaMultiplicada = []
144     for i in range( len(lista1) ):
145         listaMultiplicada.append( lista1[i] * lista2[i] )
146     return listaMultiplicada
147 
148 #&=====================================================================*
149 #&  Nombre de la Funcion/metodo: CalculoFlujoMasico
150 #&  Descripcion: Calcula el flujo masico del arco de acuerdo a los
151 #&               parametros correspondientes de este arco, incluyendo los
152 #&               siguientes:
153 #&               - Diferenciales de presion en cada arco
154 #&               - Diametro de la tuberia
```

```
155 #&               - Diametro del orificio del medidor
156 #&               - Densidad del compuesto
157 #&               - Coeficiente del medidor
158 #&
159 #&               Con estos datos se procede a calcular la velocidad
160 #&               longuitudinal en m/s, con la ecuacion:
161 #&               v0 = C0 /(raiz(1-((D0/D1)**4)))*raiz(2*(DeltaP/rho))
162 #&               Una vez obtenido este dato se multiplica por el area del
163 #&               la tuberia para encontrar el flujo volumetrico en m3/s.
164 #&               Para pasar este valor a kg/s se utiliza la ecuacion de
165 #&               densidad, Densidad = Masa / Volumen.
166 #&=====================================================================*
167 #&  Entrada <-  DeltaP, D1, D0, rho, C0
168 #&  Salida  ->  flujoMasico
169 #&=====================================================================*
170 def CalculoDeFlujoMasico( DeltaP, D1, D0, rho, C0 ):
171     pi = 3.14159                              #Constante pi
172     raiz1 = RaizCuadrada( 1 - ( D0 / D1 ) **4 )     #Raices de la ecuacion para
            calculo
173     raiz2 = RaizCuadrada( 2 * ( DeltaP / rho ) )    #flujo longuitudinal de
            ecuacion:
174                                       #
175     v0 = C0 / raiz1 * raiz2             #Velocidad longitudinal, m/s
176     V = v0 * ( pi * D0 **2 / 4 )              #Flujo volumetrico, m3/s
177     flujoMasico =  V * rho          #Flujo masico, kg/s
178     return flujoMasico
179
180 #&=====================================================================*
181 #&  Nombre de la Funcion/metodo: NodosAdyacentesPosteriores
182 #&  Descripcion: Obtiene todos los nodos adyacentes posteriores a un
183 #&               nodo en la red.
184 #&=====================================================================*
185 #&  Entrada <-  nodo, lista
186 #&  Salida  ->  nodosAdyacentesPosteriores
187 #&=====================================================================*
188 def NodosAdyacentesPosteriores( nodo, lista ):
189     nodosAdyacentesPosteriores = []
190     for i in range( 0, len(lista) - 1 ):
191         if( [nodo,i] in lista ):
192             nodosAdyacentesPosteriores.append( i )
193     return nodosAdyacentesPosteriores
194
195 #&=====================================================================*
196 #&  Nombre de la Funcion/metodo: NodosAdyacentesAnteriores
197 #&  Descripcion: Obtiene todos los nodos adyacentes anteriores a un
```

```
198 #&               nodo en la red.
199 #&=========================================================================*
200 #&  Entrada <-  nodo, lista
201 #&  Salida  ->  nodosAdyacentesAnteriores
202 #&=========================================================================*
203 def NodosAdyacentesAnteriores( nodo, lista ):
204     nodosAdyacentesAnteriores = []
205     for i in range( 0, len(lista) - 1 ):
206         if( [i,nodo] in lista ):
207             nodosAdyacentesAnteriores.append( i )
208     return nodosAdyacentesAnteriores
209 
210 #&=========================================================================*
211 #&  Nombre de la Funcion/metodo: ArcosAdyacentesAnterioresYPosteriores
212 #&  Descripcion: Obtiene todos los arcos adyacentes anteriores
213 #&               y posteriores a un arco.
214 #&=========================================================================*
215 #&  Entrada <-  numeroDeNodos, numeroDeArcos, arcos
216 #&  Salida  ->  arcosAdyacentesAnteriores, arcosAdyacentesPosteriores
217 #&=========================================================================*
218 def ArcosAdyacentesAnterioresYPosteriores(numeroDeNodos, numeroDeArcos, arcos):
219     arcosAdyacentesAnteriores = []
220     arcosAdyacentesPosteriores = []
221     for i in range( 0, numeroDeNodos ):
222         arcosPosteriores = []
223         arcosAnteriores = []
224         for j in range(0, numeroDeArcos):
225             if( [i,j] in arcos ):
226                 arcosPosteriores.append( [i,j] )
227             elif( [j,i] in arcos ):
228                 arcosAnteriores.append( [j,i] )
229 
230         arcosAdyacentesPosteriores.append( arcosPosteriores )
231         arcosAdyacentesAnteriores.append( arcosAnteriores )
232     return arcosAdyacentesAnteriores, arcosAdyacentesPosteriores
233 
234 #&=========================================================================*
235 #&  Nombre de la Funcion/metodo: Minimo
236 #&  Descripcion: Obtiene el valor minimo de una lista.
237 #&=========================================================================*
238 #&  Entrada <-  lista
239 #&  Salida  ->  minimo
240 #&=========================================================================*
241 def Minimo(lista):
242     minimo = 1000
```

```
243     for i in range( 0,len(lista) ):
244         temp = lista[i]
245         if temp <= minimo:
246             minimo = temp
247     return minimo
248 
249 #&====================================================================*
250 #&  Nombre de la Funcion/metodo: ValorMinimoCamino
251 #&  Descripcion: Obtiene el valor minimo de flujo/capacidad un camino
252 #&====================================================================*
253 #&  Entrada <-  listaCamino, listaArcos, listaFlujo
254 #&  Salida  ->  valorMinimoCamino
255 #&====================================================================*
256 def ValorMinimoCamino(camino, arcos, flujo):
257     temp = []
258     for i in range( 0, len(camino) ):
259         a = arcos.index(camino[i])
260         temp.append(flujo[a])
261     valorMinimoCamino = Minimo(temp)
262     return valorMinimoCamino
263 
264 #&====================================================================*
265 #&  Nombre de la Funcion/metodo: EliminarArcos
266 #&  Descripcion: Elimina los Arcos de la matriz de arcos cuando no se
267 #&               puede enviar flujo por el arco, no tiene mas capacidad
268 #&====================================================================*
269 #&  Entrada <-  listaCamino, listaArcos, listaCapacidades
270 #&  Salida  ->  flujoMasico
271 #&====================================================================*
272 def EliminarArcos(camino,arcos, capacidades, valorFlujoMinimo):
273     for i in range(0, len(camino)):
274         indice = arcos.index(camino[i])
275         capacidades[indice] -=  valorFlujoMinimo
276         if ( capacidades[indice] < 0.0001 ):
277             capacidades.pop(indice)
278             arcos.pop(indice)
279     return capacidades, arcos
280 
281 #&====================================================================*
282 #&  Nombre de la Funcion/metodo: ObtenerFlujoYFugaMinimoDeCadaArcoPorCamino
283 #&  Descripcion: Obtener el flujo minimo de cada arco del camino
284 #&               y el flujo de la fuga en cada arco del camino
285 #&====================================================================*
286 #&  Entrada <-  listaCamino, valorFlujoMinimo, valorFlujoFugaMinimo
```

```
287 #&  Salida  ->  tempflujoMasicoEnCadaArcoDelCamino,
        tempflujoFugaEnCadaArcoDelCamino
288 #&==========================================================================*
289 def ObtenerFlujoYFugaMinimoDeCadaArcoEnElCamino(camino, valorFlujoMinimo,
        valorFlujoFugaMinimo):
290     tempflujoMasicoEnCadaArcoDelCamino = []
291     tempflujoFugaEnCadaArcoDelCamino = []
292     for i in range(0, len(camino)):
293         tempflujoMasicoEnCadaArcoDelCamino.append(valorFlujoMinimo)
294         tempflujoFugaEnCadaArcoDelCamino.append(valorFlujoFugaMinimo)
295     return tempflujoMasicoEnCadaArcoDelCamino, tempflujoFugaEnCadaArcoDelCamino
296
297 #&==========================================================================*
298 #&  Nombre de la Funcion/metodo: ObtenerErrorMinimoYMaximoEnCadaArcoDelCamino
299 #&  Descripcion: Obtener los errores acumulador de cada medidor en el
300 #&               arco de cada camino.
301 #&==========================================================================*
302 #&  Entrada <-  listaCamino
303 #&  Salida  ->  errorAcumuladoMaximo, errorAcumuladoMaximo
304 #&==========================================================================*
305 def ObtenerErrorMinimoYMaximoEnCadaArcoDelCamino(camino):
306     errorMaximo = []
307     errorMinimo = []
308     errorAcumuladoMaximo = []
309     errorAcumuladoMinimo = []
310     for i in range(0, len(camino)):
311         errorMaximo.append(0.05)
312         errorMinimo.append(0.05)
313     errorAcumuladoMaximo = ValorAcumuladoDeLista(errorMaximo)
314     errorAcumuladoMinimo = ValorAcumuladoDeLista(errorMinimo)
315     for i in range(len( errorAcumuladoMaximo)):
316         errorAcumuladoMaximo[i] += 1
317         errorAcumuladoMinimo[i] = 1 - errorAcumuladoMinimo[i]
318     return errorAcumuladoMaximo, errorAcumuladoMinimo
319
320 #&==========================================================================*
321 #&  Nombre de la Funcion/metodo: ObtenerFlujoMasicoEnCadaArco
322 #&  Descripcion: Se obtiene la matriz de los los flujos minimos y
323 #&               maximos de cada arco de la red
324 #&==========================================================================*
325 #&  Entrada <-  listaArcos, listaTodosLosCaminos,
326 #&              flujoMasicoMaximoEnCadaArcoPorCamino,
327 #&              flujoMasicoMinimoEnCadaArcoPorCamino
328 #&  Salida  ->  flujoMasicoMaximoEnCadaArco, flujoMasicoMinimoEnCadaArco
329 #&==========================================================================*
```

```
330 def ObtenerFlujoMasicoMaximoYMinimoEnCadaArco(arcos, todosLosCaminos,
        flujoMasicoMaximoEnCadaArcoPorCamino, flujoMasicoMinimoEnCadaArcoPorCamino):
331     flujoMasicoMaximoEnCadaArco = []
332     flujoMasicoMinimoEnCadaArco = []
333     contador = []
334     for i in range(0, len(arcos)):
335         flujoMasicoMaximoEnCadaArco.append(0)
336         flujoMasicoMinimoEnCadaArco.append(0)
337         contador.append(0)
338     for i in range(0, len(todosLosCaminos)):
339         k=0
340         for j in todosLosCaminos[i]:
341             if ( j in arcos ):
342                 indice = arcos.index(j)
343                 contador[indice] += 1
344                 if (contador[indice] == 2):
345                     flujoMasicoMaximoEnCadaArco[indice] +=
                            flujoMasicoMaximoEnCadaArcoPorCamino[i][k]
346                     flujoMasicoMinimoEnCadaArco[indice] +=
                            flujoMasicoMinimoEnCadaArcoPorCamino[i][k]
347                 else:
348                     flujoMasicoMaximoEnCadaArco[indice] =
                            flujoMasicoMaximoEnCadaArcoPorCamino[i][k]
349                     flujoMasicoMinimoEnCadaArco[indice] =
                            flujoMasicoMinimoEnCadaArcoPorCamino[i][k]
350             k += 1
351     return flujoMasicoMaximoEnCadaArco, flujoMasicoMinimoEnCadaArco
352
353 #&=====================================================================*
354 #&  Nombre de la Funcion/metodo: BalanceDeMateriaPorNodo
355 #&  Descripcion: Realizar un balance de materia en cada nodo
356 #%               teniendo en cuenta las entradas y salidas de cada
357 #&               nodo.
358 #&=====================================================================*
359 #&  Entrada <-  numeroDeNodos, listaArcos, listaArcosAdyacentesPosteriores,
360 #&              listaArcosAdyacentesAnteriores, flujoMasicoConFugas,
361 #&              flujoMasicoMaximoEnCadaArco, flujoMasicoMinimoEnCadaArco
362 #&  Salida  ->  balancePorNodo, balanceMaximoPorNodo, balanceMinimoPorNodo
363 #&=====================================================================*
364 def BalanceDeMateriaPorNodo(numeroDeNodos, numeroDeArcos, arcos,
        flujoMasicoConFugas, flujoMasicoMaximoEnCadaArco, flujoMasicoMinimoEnCadaArco):
365     balancePorNodo = []
366     balanceMaximoPorNodo = []
367     balanceMinimoPorNodo = []
368     entradaAlNodo = []
```

```
369     entradaMaximaAlNodo = []
370     entradaMinimaAlnodo = []
371     salidaDelNodo = []
372     salidaMaximaDelNodo = []
373     salidaMinimaDelNodo = []
374     arcosAdyacentesAnteriores = []
375     arcosAdyacentesPosteriores = []
376
377
378     arcosAdyacentesAnteriores, arcosAdyacentesPosteriores =
            ArcosAdyacentesAnterioresYPosteriores(numeroDeNodos, numeroDeArcos, arcos)
379
380     for i in range(0, numeroDeNodos):
381         sumaEntradas = 0
382         sumaEntradasMaxima = 0
383         sumaEntradasMinima = 0
384         sumaSalidas = 0
385         sumaSalidasMaxima = 0
386         sumaSalidasMinima = 0
387
388         #Suma de flujos de salida
389         for j in arcosAdyacentesPosteriores[i]:
390             indice = 0
391             if (j in arcos):
392                 indice = arcos.index(j)
393                 sumaSalidas += flujoMasicoConFugas[indice]
394                 sumaSalidasMaxima += flujoMasicoMaximoEnCadaArco[indice]
395                 sumaSalidasMinima += flujoMasicoMinimoEnCadaArco[indice]
396         salidaDelNodo.append(sumaSalidas)
397         salidaMaximaDelNodo.append(sumaSalidasMaxima)
398         salidaMinimaDelNodo.append(sumaSalidasMinima)
399         #Suma fujos de entrada
400         for j in arcosAdyacentesAnteriores[i]:
401             indice = 0
402             if (j in arcos):
403                 indice = arcos.index(j)
404                 sumaEntradas += flujoMasicoConFugas[indice]
405                 sumaEntradasMaxima += flujoMasicoMaximoEnCadaArco[indice]
406                 sumaEntradasMinima += flujoMasicoMinimoEnCadaArco[indice]
407         entradaAlNodo.append(sumaEntradas)
408         entradaMaximaAlNodo.append(sumaEntradasMaxima)
409         entradaMinimaAlnodo.append(sumaEntradasMinima)
410         balancePorNodo.append(entradaAlNodo[i] - salidaDelNodo[i])
411         balanceMaximoPorNodo.append(entradaMaximaAlNodo[i] - salidaMinimaDelNodo[i
                ])
```

```
412         balanceMinimoPorNodo.append(entradaMinimaAlnodo[i] - salidaMaximaDelNodo[i
                ])
413     return balancePorNodo, balanceMaximoPorNodo, balanceMinimoPorNodo
414
415 #&=====================================================================*
416 #&  Nombre de la Funcion/metodo: DeteccionDeFugasEnNodos
417 #&  Descripcion: Detectar las fugas en cada nodo de la refineria
418 #&=====================================================================*
419 #&  Entrada <-  listaCamino
420 #&  Salida  ->  errorAcumuladoMaximo, errorAcumuladoMaximo
421 #&=====================================================================*
422 def DeteccionDeFugasEnNodos(numeroDeNodos, balancePorNodo, balanceMaximoPorNodo,
        balanceMinimoPorNodo):
423     nodosConGanancia = []
424     nodosConFuga = []
425     balanceAcumuladoNodoGanancia = []
426     balanceAcumuladoNodoFuga = []
427     fugaEnNodo = []
428     for i in range(0,numeroDeNodos):
429         if (balancePorNodo[i] > balanceMaximoPorNodo[i]):
430             diferencia = balancePorNodo[i] - balanceMaximoPorNodo[i]
431             nodosConGanancia.append(i)
432             balanceAcumuladoNodoGanancia.append(balancePorNodo[i])
433         elif (balancePorNodo[i] < balanceMinimoPorNodo[i]):
434             diferencia = balancePorNodo[i] - balanceMinimoPorNodo[i]
435             nodosConFuga.append(i)
436             balanceAcumuladoNodoFuga.append(balancePorNodo[i])
437
438     for i in range(0, numeroDeNodos):
439         fugaEnNodo.append(0)
440
441     for i in nodosConFuga:
442         fugaEnNodo[i] -= 1
443
444     for i in nodosConGanancia:
445         fugaEnNodo[i] += 1
446
447     return fugaEnNodo
448
449 #&=====================================================================*
450 #&  Nombre de la Funcion/metodo: DeteccionDeFugasEnArcos
451 #&  Descripcion: Detectar las fugas en cada arco de la refineria
452 #&=====================================================================*
453 #&  Entrada <-  todosLosCaminos,flujoMasicoMaximoEnCadaArcoPorCamino,
454 #&              flujoMasicoMinimoEnCadaArcoPorCamino,
```

```
455 #&              flujoFugaEnCadaArcoPorCadaCamino, listaArcos, flujoMasico
456 #&  Salida  ->  fugaEnArco, flujoMasico
457 #&================================================================*
458 def DeteccionDeFugasEnArcos(todosLosCaminos,flujoMasicoMaximoEnCadaArcoPorCamino,
        flujoMasicoMinimoEnCadaArcoPorCamino, flujoFugaEnCadaArcoPorCadaCamino, arcos,
        flujoMasico):
459
460     arcosConGanancia = []   #arcos ganancia
461     arcosConFuga = []   #arcos fuga
462     flujoAcumuladoGanancia = []
463     flujoAcumuladoFuga = []
464     fugaEnArco = []
465     for i in range(0, len(todosLosCaminos)):
466         for j in range (0,len(flujoFugaEnCadaArcoPorCadaCamino[i])):
467             if (flujoFugaEnCadaArcoPorCadaCamino[i][j] >
                    flujoMasicoMaximoEnCadaArcoPorCamino[i][j]):
468
469                 diferencia = flujoFugaEnCadaArcoPorCadaCamino[i][j] -
                        flujoMasicoMaximoEnCadaArcoPorCamino[i][j]
470
471                 arcosConGanancia.append(todosLosCaminos[i][j])
472
473                 flujoAcumuladoGanancia.append(flujoFugaEnCadaArcoPorCadaCamino[i][j
                        ])
474             elif (flujoFugaEnCadaArcoPorCadaCamino[i][j] <
                    flujoMasicoMinimoEnCadaArcoPorCamino[i][j]):
475                 diferencia = 0
476                 diferencia = flujoMasicoMaximoEnCadaArcoPorCamino[i][j] -
                        flujoFugaEnCadaArcoPorCadaCamino[i][j]
477                 arcosConFuga.append(todosLosCaminos[i][j])
478                 flujoAcumuladoFuga.append(flujoFugaEnCadaArcoPorCadaCamino[i][j])
479
480     for i in range(0,len(arcos)):
481         fugaEnArco.append(0)
482
483     for i in arcosConFuga:
484         k = 0
485         if i in arcos:
486             indice = arcos.index(i)
487             fugaEnArco[indice] -= 1
488             flujoMasico[indice] = flujoAcumuladoFuga[k]
489         k += 1
490
491
492     for i in arcosConGanancia:
```

```
493         k = 0
494         if i in arcos:
495             indice = arcos.index(i)
496             fugaEnArco[indice] += 1
497             flujoMasico[indice] = flujoAcumuladoGanancia[k]
498         k += 1
499 
500     return fugaEnArco, flujoMasico
501 
502 #&=====================================================================*
503 #&  Nombre de la Funcion/metodo: ConvertirNivelesAFlujo
504 #&  Descripcion: Convertir los niveles leidos a variable de -1 y 1 ya a variables
    de
505 #&               flujo
506 #&               flujoMasicoNievl: la variable de flujo que nos indica que esta en
    rango
507 #&               -1, 0 o 1 de cada arco
508 #&=====================================================================*
509 #&  Entrada <-  numeroDeArcos, numeroDeRepeticiones, nivelPorRepeticion,
    flujoMasico
510 #&  Salida  ->  flujoMasicoPorRepeticion, flujoMasicoNivelPorRepeticion
511 #&=====================================================================*
512 def ConvertirNivelesAFlujo(numeroDeArcos, numeroDeRepeticiones, nivelPorRepeticion,
     flujoMasico):
513     flujoMasicoPorRepeticion = []
514     flujoMasicoNivelPorRepeticion = []
515     deteccionFugasPorRepeticion = []
516     for i in range(0,numeroDeRepeticiones):
517         tempFlujo=[]
518         tempNivel=[]
519         tempDetectar=[]
520         for j in range(0, numeroDeArcos):
521             if ( nivelPorRepeticion[i][j] == 1 ):
522                 tempFlujo.append(flujoMasico[j] * 0.8)
523                 tempNivel.append(-1)
524                 tempDetectar.append(-1)
525             elif ( nivelPorRepeticion[i][j] == 2 ):
526                 tempFlujo.append(flujoMasico[j])
527                 tempNivel.append(0)
528                 tempDetectar.append(-1)
529             elif ( nivelPorRepeticion[i][j] == 3 ):
530                 tempFlujo.append(flujoMasico[j] * 1.2)
531                 tempNivel.append(1)
532                 tempDetectar.append(0)
533         flujoMasicoPorRepeticion.append(tempFlujo)
```

```
534         flujoMasicoNivelPorRepeticion.append(tempNivel)
535         deteccionFugasPorRepeticion.append(tempDetectar)
536 
537     return flujoMasicoPorRepeticion, flujoMasicoNivelPorRepeticion,
            deteccionFugasPorRepeticion
538 
539 
540 #&=====================================================================*
541 #&  Nombre de la Funcion/metodo: ImprimirResultados
542 #&  Descripcion: Detectar las fugas en cada arco de la refineria
543 #&=====================================================================*
544 #&  Entrada <-  numeroDeNodos, numeroDeArcos, balancePorNodo, fugaEnNodo,
545 #&              flujoMasico, fugaEnArco
546 #&  Salida  ->  impresion de resultados
547 #&=====================================================================*
548 def ImprimirResultados(numeroDeNodos, numeroDeArcos, balancePorNodo, fugaEnNodo,
    flujoMasico, fugaEnArco):
549     #Imprimir los datos principales de la red, numero de nodos y numero de Arcos
550     print 0, numeroDeNodos, numeroDeArcos
551     #Identificador para nodos
552     print 1,
553     #Imprimir los balances realizados en cada nodo
554     for i in range(0,numeroDeNodos):
555         print balancePorNodo[i],
556     #Imprimir la deteccion de fuga en los nodos
557     for i in range(0,len(fugaEnNodo)):
558         print fugaEnNodo[i],
559 
560     print
561     #Identificador para arcos
562     print 2,
563     #Imprimir los flujos de cada nodo.
564     for i in range(0,len(flujoMasico)):
565         print flujoMasico[i],
566     #Imprimir la deteccion de fuga en los arcos
567     for i in range(0,len(fugaEnArco)):
568         print fugaEnArco[i],
569 
570 #&=====================================================================*
571 #&  Nombre de la Funcion/metodo: ImprimirResultadosDiseno
572 #&  Descripcion: Imprimir resultados del diseno de experimentos
573 #&=====================================================================*
574 #&  Entrada <-  numeroDeArcos, numeroDeRepeticiones, lujoMasicoPorRepeticion,
    flujoMasicoNivelPorRepeticion, deteccionFugasPorRepeticion
575 #&  Salida  ->  impresion de resultados
```

```
576 #&=========================================================================*
577 def ImprimirResultadosDiseno(numeroDeArcos, numeroDeRepeticiones,
        flujoMasicoPorRepeticion, flujoMasicoNivelPorRepeticion,
        deteccionFugasPorRepeticion):
578     for i in range(0,numeroDeRepeticiones):
579         print
580         print 3,
581         for j in range(0,numeroDeArcos):
582             print flujoMasicoPorRepeticion[i][j],
583     for i in range(0,numeroDeRepeticiones):
584         print
585         print 4,
586         for j in range(0,numeroDeArcos):
587             print flujoMasicoNivelPorRepeticion[i][j],
588
589     for i in range(0,numeroDeRepeticiones):
590         print
591         print 5,
592         for j in range(0,numeroDeArcos):
593             print deteccionFugasPorRepeticion[i][j],
```

Apéndice B

Código Computacional

B.1 Código General de Clasificación de Flujos en la Red de Refinería

```
%&*****************************************************************
%&*                          PISIS FIME UANL                      *
%&*****************************************************************
%&* Nombre del Programa: ClasificadorDeFugasEnArcos               *
%&* Descripcion        : Lectura de datos requeridos para el programa:  *
%&*                      - Valores obtenidos por el simulador en archivo *
%&*                        que contiene los valores tanto de flujo en   *
%&*                        los arcos como su correspondiente deteccion)  *
%&*                        Nombre del Archivo DatosDeFlujoEnArcos.txt   *
%&*                                                                     *
%&* Funcional          : Luis Alejandro Benavides Vazquez              *
%&* Programador        : Luis Alejandro Benavides Vazquez              *
%&* Fecha Creacion     : 1 Septiembre del 2013 (fecha tentativa)       *
%&*****************************************************************
%&*                        LOG DE MODIFICACIONES                    *
%&*****************************************************************
%&* Descripcion        : Modificacion de programa base para estandares.  *
%&*                      Estandarizar el codigo para su mejor lectura y  *
%&*                      modificaciones posteriores                     *
%&*                                                                     *
%&* Funcional          : Luis Alejandro Benavides Vazquez              *
%&* Programador        : Luis Alejandro Benavides Vazquez              *
%&* Fecha Creacion     : 28 Agosto del 2013                            *
%&*****************************************************************

%&=================================================================
%& Limpiar todas las variables y limpiar la terminal
```

```
28 %&=========================================================================
29 clc
30 clear all
31
32 %&=========================================================================
33 %& Variables
34 %&=========================================================================
35 %& - DatosDeFlujoEnArcos:   Almacena los flujos de cada arco y
36 %&                         su deteccion correspondiente.
37 %&
38 %& - FlujoMasico:           Almacena los valores extremos de las mediciones
39 %&                         de flujos (minimo y maximo).
40 %&
41 %& - FlujoNivel:            Almacena los valores transformados de -1, 0 y 1
42 %&                         de los flujos de cada arco, para los 232
43 %&                         pruebas requeridas (Composito Central).
44 %&
45 %& - DeteccionFlujoNivelParaDiseno: Almacena las detecciones
46 %&                         correspondientes de los flujos almacenados en
47 %&                         la variable FlujoNivel.
48 %&
49 %& - DatosPruebaArcos:      Almacena los valores de los flujos que se
50 %&                         requieren para realizar las pruebas de que tan
51 %&                         bien se efectua la clasificacion.
52 %&
53 %& - ValoresDeEntrada:      Valores que seran introducidos a la red
54 %&                         neuronal como entradas (inputs).
55 %&
56 %& - ValoresDeSalida:       Valores que seran introducidos a la red
57 %&                         neuronal como salidas (targets).
58 %&
59 %& - ValoresDeEntradaPrueba: Valores que seran introducidos para probar
60 %&                         la red neuronal.
61 %&
62 %& - ValoresDeSalidaPrueba: Valores que seran puestos para comparar con los
63 %&                         valores encotrados al clasificar los valores de
64 %&                         entrada de prueba.
65 %&=========================================================================
66
67 %&=========================================================================
68 %& Lectura de Datos
69 %&=========================================================================
70 DatosDeFlujoEnArcos = textread('DatosDeFlujoEnArcos.txt');
71 FlujoMasico = textread('FlujoMasicoParaDiseno.txt');
72 FlujoNivel = textread('FlujoConvertidoNivelParaDiseno.txt');
```

```
73 DeteccionFlujoNivelParaDiseno = textread('DeteccionFlujoNivelParaDiseno.txt');
74 DatosPruebaArcos = textread('DatosDeFlujoEnArcosPrueba.txt');
75
76 %%=====================================================================
77 %% Eliminar primer columna de cada variables (identificador)
78 %%=====================================================================
79 DatosDeFlujoEnArcos(:,1) = [];
80 FlujoMasico(:,1) = [];
81 FlujoNivel(:,1) = [];
82 DeteccionFlujoNivelParaDiseno(:,1) = [];
83 DatosPruebaArcos(:,1) = [];
84
85 %%=====================================================================
86 %% Numero de Arcos
87 %%=====================================================================
88 [NumeroDeArcos] = size(DatosDeFlujoEnArcos, 2) / 2;
89
90 %%=====================================================================
91 %% Valores de entrada y salida para el entrenamiento de la red neuronal.
92 %%=====================================================================
93 ValoresDeEntrada = DatosDeFlujoEnArcos( :, (1:NumeroDeArcos) );
94 ValoresDeSalida = DatosDeFlujoEnArcos( :, ( ( NumeroDeArcos + 1 ): NumeroDeArcos *
       2 ) );
95
96 %%=====================================================================
97 %% Valores de entrada y salida para la prueba de la red neuronal.
98 %%=====================================================================
99 ValoresDeEntradaPrueba = DatosPruebaArcos( :, (1:NumeroDeArcos) );
100 ValoresDeSalidaPrueba = DatosPruebaArcos( :, ( ( NumeroDeArcos + 1 ): NumeroDeArcos
        * 2) );
101
102 %%=====================================================================
103 %% Transponer matriz para que las filas sean los arcos y columnas
104 %% las repeticiones, esto para ponerlo en formato de Matlab para introducir
105 %% los datos a la red neuronal.
106 %%=====================================================================
107 ValoresDeEntrada = ValoresDeEntrada';
108 ValoresDeSalida = ValoresDeSalida';
109 FlujoMasico = FlujoMasico';
110 FlujoNivel = FlujoNivel' ;
111 DeteccionFlujoNivelParaDiseno = DeteccionFlujoNivelParaDiseno';
112 DeteccionFlujoNivelParaDiseno = abs(DeteccionFlujoNivelParaDiseno);
113 %%===============
114 %% ValoresPrueba
115 %%===============
```

```
116 ValoresDeEntradaPrueba = ValoresDeEntradaPrueba';
117 ValoresDeSalidaPrueba = ValoresDeSalidaPrueba';
118
119 %&=============================================================================
120 %& Numero de Replicas (repeticiones de datos de medidores)
121 %&=============================================================================
122 NumeroDeReplicas = size(ValoresDeEntrada,2);
123 NumeroDeExperimentos = size(FlujoNivel,2);
124 %&===============
125 %& ValoresPrueba
126 %&===============
127 NumeroDeReplicasPrueba = size(ValoresDeEntradaPrueba,2);
128
129 %&=============================================================================
130 %& Dejar valores de fugas (-1) y no fuga (0)
131 %& Para los datos de entrada a la red y los datos de prueba.
132 %&=============================================================================
133 for j = 1:NumeroDeArcos
134     for i = 1:NumeroDeReplicas
135         if ( ValoresDeSalida(j,i) > 0 )
136       ValoresDeSalida(j,i) = 0;
137         end
138         if ( ValoresDeSalida(j,i) < (-1) )
139       ValoresDeSalida(j,i) = -1;
140         end
141     end
142 end
143
144 %&===============
145 %& ValoresPrueba
146 %&===============
147 for j = 1:NumeroDeArcos
148     for i = 1:NumeroDeReplicasPrueba
149         if ( ValoresDeSalidaPrueba(j,i) > 0 )
150       ValoresDeSalidaPrueba(j,i) = 0;
151         end
152         if ( ValoresDeSalidaPrueba(j,i) < (-1) )
153       ValoresDeSalidaPrueba(j,i) = -1;
154         end
155     end
156 end
157
158 %&=============================================================================
159 %& Eliminar numeros negativos
160 %&=============================================================================
```

```
161 ValoresDeSalida = abs(ValoresDeSalida);
162 TempValoresDeSalida = ValoresDeSalida;
163 %&===============
164 %& ValoresPrueba
165 %&===============
166 ValoresDeSalidaPrueba = abs(ValoresDeSalidaPrueba);
167
168 %&=========================================================================
169 %& Cambiar los valores de entrada a un rango de -1 a 1 para
170 %& no tener problema de dimensiones.
171 %&=========================================================================
172 ValoresDeEntrada = mapminmax(ValoresDeEntrada);
173 FlujoNivel = mapminmax(FlujoMasico);
174 TempValoresDeEntrada = ValoresDeEntrada;
175 %&===============
176 %& ValoresPrueba
177 %&===============
178 ValoresDeEntradaPrueba = mapminmax(ValoresDeEntradaPrueba);
179
180 %&=========================================================================
181 %& Calcular el numero de repeticiones que se haran para cada red
182 %& neuronal (aproximadamente 1.2 veces el numero de variables)
183 %&=========================================================================
184 NumeroRepeticiones =1;% round(NumeroDeArcos * 1.2);
185
186 %&=========================================================================
187 %& Numero de neuronas ocultas
188 %&=========================================================================
189 a = 1:1:10;
190 b = 20:10:60;
191 NumeroDeNeuronasOcultas = 10;
192
193 %&=========================================================================
194 %& Almacener los datos de la experimentacion se tienen los datos de.
195 %& - Porcentaje de Acierto de Entrenamiento
196 %& - Porcentaje de Acierto de Validacion
197 %& - Porcentaje de Acierto de Prueba
198 %& - Tiempo de la red neuronal
199 %& - Suma de Errores Cuadrados de Entrenamiento
200 %& - Suma de Errores Cuadrados de Validacion
201 %& - Suma de Errores Cuadrados de Prueba
202 %& Quitar el simbolo "%" para poder obteer los datos
203 %& mencionados anteriormente.
204 %& Para almacenarlo en un archivo de Excel poner.
205 %& xlswrite('nombre_del_archivo.xls', Nombre_De_Variable,'Hoja 1', 'Celdas')
```

```
206 %&=============================================================================
207 %ExportacionDeDatosPorcentajeTiempoSSE = RedEntrenamientoValidacionPrueba(
        ValoresDeEntrada, ValoresDeSalida, NumeroDeNeuronasOcultas, NumeroRepeticiones);
208
209 %&=============================================================================
210 %& Prueba para 100 datos
211 %&=============================================================================
212 ValoresDeEntrada = TempValoresDeEntrada(:,1:100);
213 ValoresDeSalida = TempValoresDeSalida(:,1:100);
214 size(ValoresDeEntrada)
215
216 %&================================
217 %& Realizar repeticiones de
218 %& la red neuronal para encontrar
219 %& el mejor.
220 %&================================
221 Red = EncontrarMejorRedNeuronal(ValoresDeEntrada, ValoresDeSalida,
        NumeroDeNeuronasOcultas, NumeroRepeticiones);
222
223 ClasificacionDeteccionFlujoNivelParaDiseno = zeros(NumeroDeArcos,
        NumeroDeExperimentos);
224
225 ClasificacionDeteccionFlujoNivelParaDiseno = sim(Red,ValoresDeEntradaPrueba);
226
227 Coincidencia = ValoresDeSalidaPrueba == round(
        ClasificacionDeteccionFlujoNivelParaDiseno);
228
229 promedioCoincidencia = zeros(1,NumeroDeReplicasPrueba);
230 for i = 1:NumeroDeReplicasPrueba
231    promedioCoincidencia(1,i) = sum(Coincidencia(:,i))/NumeroDeArcos*100;
232 end
233
234
235 %&================================
236 %& Graficar el histograma de los
237 %& promedios de Coincidencia.
238 %&================================
239 subplot(2,2,1); hist(promedioCoincidencia);
240 axis([0 100 0 NumeroDeReplicasPrueba])
241 xlabel('Porcentaje Acierto')
242 ylabel('Numero de Repeticiones')
243 %&=============================================================================
244
245 %&=============================================================================
246 %& Prueba para el dieno de 232 datos
```

```
247 %%=============================================================================
248 ValoresDeEntrada = FlujoNivel;
249 ValoresDeSalida = DeteccionFlujoNivelParaDiseno;
250 size(ValoresDeEntrada)
251
252 %%================================
253 %% Realizar repeticiones de
254 %% la red neuronal para encontrar
255 %% el mejor.
256 %%================================
257 Red = EncontrarMejorRedNeuronal(ValoresDeEntrada, ValoresDeSalida,
        NumeroDeNeuronasOcultas, NumeroRepeticiones);
258
259 ClasificacionDeteccionFlujoNivelParaDiseno = zeros(NumeroDeArcos,
        NumeroDeExperimentos);
260
261 ClasificacionDeteccionFlujoNivelParaDiseno = sim(Red,ValoresDeEntradaPrueba);
262
263 Coincidencia = ValoresDeSalidaPrueba == round(
        ClasificacionDeteccionFlujoNivelParaDiseno);
264
265 promedioCoincidencia = zeros(1,NumeroDeReplicasPrueba);
266 for i = 1:NumeroDeReplicasPrueba
267    promedioCoincidencia(1,i) = sum(Coincidencia(:,i))/NumeroDeArcos*100;
268 end
269
270 %%================================
271 %% Graficar el histograma de los
272 %% promedios de Coincidencia.
273 %%================================
274 subplot(2,2,2); hist(promedioCoincidencia);
275 axis([0 100 0 NumeroDeReplicasPrueba])
276 xlabel('Porcentaje Acierto')
277 ylabel('Numero de Repeticiones')
278
279 %%=============================================================================
280 %% Prueba para 332 datos (100 de valores de entrada y 232 de diseno)
281 %%=============================================================================
282 ValoresDeEntrada = [TempValoresDeEntrada(:,1:100) FlujoNivel];
283 ValoresDeSalida = [TempValoresDeSalida(:,1:100) DeteccionFlujoNivelParaDiseno];
284 size(ValoresDeEntrada)
285
286 %%================================
287 %% Realizar repeticiones de
288 %% la red neuronal para encontrar
```

```
289 %& el mejor.
290 %&==============================
291 Red = EncontrarMejorRedNeuronal(ValoresDeEntrada, ValoresDeSalida,
        NumeroDeNeuronasOcultas, NumeroRepeticiones);
292
293 ClasificacionDeteccionFlujoNivelParaDiseno = zeros(NumeroDeArcos,
        NumeroDeExperimentos);
294
295 ClasificacionDeteccionFlujoNivelParaDiseno = sim(Red,ValoresDeEntradaPrueba);
296
297 Coincidencia = ValoresDeSalidaPrueba == round(
        ClasificacionDeteccionFlujoNivelParaDiseno);
298
299 promedioCoincidencia = zeros(1,NumeroDeReplicasPrueba);
300 for i = 1:NumeroDeReplicasPrueba
301    promedioCoincidencia(1,i) = sum(Coincidencia(:,i))/NumeroDeArcos*100;
302 end
303
304 subplot(2,2,3); hist(promedioCoincidencia);
305 axis([0 100 0 NumeroDeReplicasPrueba])
306 xlabel('Porcentaje Acierto')
307 ylabel('Numero de Repeticiones')
308
309 %&=====================================================================
310 %& Prueba para 1000 datos (768 de valores de entrada y 232 de diseno)
311 %&=====================================================================
312 ValoresDeEntrada = [TempValoresDeEntrada FlujoNivel];
313 ValoresDeSalida = [TempValoresDeSalida DeteccionFlujoNivelParaDiseno];
314 size(ValoresDeEntrada)
315
316 %&==============================
317 %& Realizar repeticiones de
318 %& la red neuronal para encontrar
319 %& el mejor.
320 %&==============================
321 Red = EncontrarMejorRedNeuronal(ValoresDeEntrada, ValoresDeSalida,
        NumeroDeNeuronasOcultas, NumeroRepeticiones);
322
323 ClasificacionDeteccionFlujoNivelParaDiseno = zeros(NumeroDeArcos,
        NumeroDeExperimentos);
324
325 ClasificacionDeteccionFlujoNivelParaDiseno = sim(Red,ValoresDeEntradaPrueba);
326
327 Coincidencia = ValoresDeSalidaPrueba == round(
        ClasificacionDeteccionFlujoNivelParaDiseno);
```

```
328
329 promedioCoincidencia = zeros(1,NumeroDeReplicasPrueba);
330 for i = 1:NumeroDeReplicasPrueba
331    promedioCoincidencia(1,i) = sum(Coincidencia(:,i))/NumeroDeArcos*100;
332 end
333
334 %&================================
335 %& Graficar el histograma de los
336 %& promedios de Coincidencia.
337 %&================================
338 subplot(2,2,4); hist(promedioCoincidencia);
339 axis([0 100 0 NumeroDeReplicasPrueba])
340 xlabel('Porcentaje Acierto')
341 ylabel('Numero de Repeticiones')
342
343 %&=====================================================================
344 %&  Fin
345 %&=====================================================================
```

B.2 Función Red Neuronal

```
1  function [net, trS, cvS, tstS] = RedNeuronal(inputs,targets,numHiddenNeurons)
2  %&===================================================================
3  %& Creacion de la Red
4  %&===================================================================
5  %& Funcion de transferencia de la capa de entrada a la capa de neuronas
6  %& ocultas (1) y de la capa de neurona oculas a la capa de salida (2).
7  %& transferFcn={'1' '2'};
8  transferFcn={'logsig' 'purelin'};
9
10 %& Funcion de entrenamiento para la red neuronal (1).
11 %& trainFcn={'1'};
12 trainFcn='trainlm';
13
14 %& Numero de neuronas en la capa de salida.
15 nout = size(targets,1);
16
17 %& Crear la red para neuronal como feedforward.
18 net = newff(minmax(inputs),targets,[numHiddenNeurons nout],transferFcn,trainFcn);
19
20 %Dividir el conjunto de datos para entrenamiento, validacion y prueba.
21 %trS = conjunto de entrenamiento.
22 %cvS = conjunto de validacion.
```

```
%tstS = conjunto de prueba.
[trS, cvS, tstS] = dividevec(inputs, targets, 6/36, 6/36);

%&=========================================================================
%& Entrenamiento de la Red Neuronal
%&=========================================================================
%& Inicializar los valores de los pesos de la red neuronal.
net=init(net);

%& Funcion de desempeño de la red neuronal.
net.performFcn = 'sse';

%& No desplegar nntraintool default is 1.
net.trainParam.showWindow=0;

%& Enrenar la red neuronal
[net,tr] = train(net, trS.P, trS.T, [], [], cvS, tstS);
%outputs=tr;
%outputs_pr = sim(net,tstS.P);
%outputs_en = sim(net,inputs);
% Plot
%plotperf(tr)
%plotfit(net,inputs,targets)
%plotregression(targets,outputs)
```

B.3 Función Valores de Entrenamiento, Validación y Prueba

```
function [exportacionDeDatosPorcentajeTiempoSSE] = RedEntrenamientoValidacionPrueba
    (valoresDeEntrada, valoresDeSalida, numeroDeNeuronasOcultas,
    numeroDeRepeticiones)



TiempoRedNeuronal = zeros(length(numeroDeNeuronasOcultas), numeroDeRepeticiones);
PorcentajeEntrenamiento = zeros(length(numeroDeNeuronasOcultas),
    numeroDeRepeticiones);
PorcentajeValidacion = zeros(length(numeroDeNeuronasOcultas), numeroDeRepeticiones)
    ;
PorcentajeTest = zeros(length(numeroDeNeuronasOcultas), numeroDeRepeticiones);
SSEEntrenamiento = zeros(length(numeroDeNeuronasOcultas), numeroDeRepeticiones);
SSEValidacion = zeros(length(numeroDeNeuronasOcultas), numeroDeRepeticiones);
```

```
11 SSEPrueba = zeros(length(numeroDeNeuronasOcultas), numeroDeRepeticiones);
12
13
14
15 for i=1:length(numeroDeNeuronasOcultas)
16     for j=1:numeroDeRepeticiones
17
18         numeroDeNeuronasOcultasEscalar = numeroDeNeuronasOcultas(i);
19         tic;
20         [Red, ConjuntoDeEntrenamiento, ConjuntoDeValidacion, ConjuntoDePrueba] =
               RedNeuronal(valoresDeEntrada, valoresDeSalida,
               numeroDeNeuronasOcultasEscalar);
21         tiempo=toc;
22         %&=====================================================================
23         %& Valores de entrada y valores objetivo del conjunto de datos
24         %&=====================================================================
25         ValorDeEntradaDelConjuntoDeEntrenamiento = ConjuntoDeEntrenamiento.P;
26         ValorObjetivoDelConjuntoDeEntrenamiento = ConjuntoDeEntrenamiento.T;
27         ValorDeEntradaDelConjuntoDeValidacion = ConjuntoDeValidacion.P;
28         ValorObjetivoDelConjuntoDeValidacion = ConjuntoDeValidacion.T;
29         ValorDeEntradaDelConjuntoDePrueba = ConjuntoDePrueba.P;
30         ValorObjetivoDelConjuntoDePrueba = ConjuntoDePrueba.T;
31
32         %&=====================================================================
33         %& Clasificacion de los datos de entrada
34         %&=====================================================================
35         SalidaDeClasificacionDelConjuntoDeEntrenamiento = sim(Red,
               ValorDeEntradaDelConjuntoDeEntrenamiento);
36         SalidaDeClasificacionDelConjuntoDeValidacion = sim(Red,
               ValorDeEntradaDelConjuntoDeValidacion);
37         SalidaDeClasificacionDelConjuntoDePrueba = sim(Red,
               ValorDeEntradaDelConjuntoDePrueba);
38
39         %&=====================================================================
40         %& Error entre valores objetivo y valores de clasificacion
41         %&=====================================================================
42         ErrorEntrenamiento = ValorObjetivoDelConjuntoDeEntrenamiento -
               SalidaDeClasificacionDelConjuntoDeEntrenamiento;
43         ErrorValidacion = ValorObjetivoDelConjuntoDeValidacion -
               SalidaDeClasificacionDelConjuntoDeValidacion;
44         ErrorPrueba = ValorObjetivoDelConjuntoDePrueba -
               SalidaDeClasificacionDelConjuntoDePrueba;
45
46         %&=====================================================================
47         %& Suma cuadrada de errores
```

```
48          %&=========================================================================
49          sseEntrenamiento = sse(ErrorEntrenamiento,
                ValorObjetivoDelConjuntoDeEntrenamiento,
                SalidaDeClasificacionDelConjuntoDeEntrenamiento);
50          sseValidacion = sse(ErrorValidacion, ValorObjetivoDelConjuntoDeValidacion,
                SalidaDeClasificacionDelConjuntoDeValidacion);
51          ssePrueba = sse(ErrorPrueba, ValorObjetivoDelConjuntoDePrueba,
                SalidaDeClasificacionDelConjuntoDePrueba);
52
53          SalidaDeClasificacionDelConjuntoDeEntrenamiento = round(
                SalidaDeClasificacionDelConjuntoDeEntrenamiento);
54          SalidaDeClasificacionDelConjuntoDeValidacion = round(
                SalidaDeClasificacionDelConjuntoDeValidacion);
55          SalidaDeClasificacionDelConjuntoDePrueba = round(
                SalidaDeClasificacionDelConjuntoDePrueba);
56
57
58          [filasDelConjuntoDeEntrenamiento, columnasDelConjuntoDeEntrenamiento] =
                size(ValorObjetivoDelConjuntoDeEntrenamiento);
59          [filasDelConjuntoDeValidacion, columnasDelConjuntoDeValidacion] = size(
                ValorObjetivoDelConjuntoDeValidacion);
60          [filasDelConjuntoDePrueba, columnasDelConjuntoDePrueba] = size(
                ValorObjetivoDelConjuntoDePrueba);
61
62          PorcientoAciertoEntrenamiento=sum(sum(
                ValorObjetivoDelConjuntoDeEntrenamiento ==
                SalidaDeClasificacionDelConjuntoDeEntrenamiento))/(
                filasDelConjuntoDeEntrenamiento*columnasDelConjuntoDeEntrenamiento)*100;
63          PorcientoAciertoValidacion=sum(sum(ValorObjetivoDelConjuntoDeValidacion ==
                SalidaDeClasificacionDelConjuntoDeValidacion))/(
                filasDelConjuntoDeValidacion*columnasDelConjuntoDeValidacion)*100;
64          PorcientoAciertoTest=sum(sum(ValorObjetivoDelConjuntoDePrueba ==
                SalidaDeClasificacionDelConjuntoDePrueba))/(filasDelConjuntoDePrueba*
                columnasDelConjuntoDePrueba)*100;
65
66          %Variables para almacenar tiempo, porcentaje de aciertos y error
67          TiempoRedNeuronal(i,j) = tiempo;
68          PorcentajeEntrenamiento(i,j) = PorcientoAciertoEntrenamiento;
69          PorcentajeValidacion(i,j) = PorcientoAciertoValidacion;
70          PorcentajeTest(i,j) = PorcientoAciertoTest;
71          SSEEntrenamiento(i,j) = sseEntrenamiento;
72          SSEValidacion(i,j) = sseValidacion;
73          SSEPrueba(i,j) = ssePrueba;
74
75
```

```
76
77      end
78  end
79  exportacionDeDatosPorcentajeTiempoSSE = [PorcentajeEntrenamiento'
        PorcentajeValidacion' PorcentajeTest' TiempoRedNeuronal' SSEEntrenamiento'
        SSEValidacion' SSEPrueba'];
80
81  %xlswrite('R_1_10.xls',ExportacionDeDatos,'Hoja 1','A1:T6')
82
83  end
```

B.4 Función Multistart para Encontrar Mejor Red Neuornal Artificial

```
1   function [RedBest] = EncontrarMejorRedNeuronal(valoresDeEntrada, valoresDeSalida,
        numeroDeNeuronasOcultas, numeroDeRepeticiones)
2
3
4   mejor = -1;
5
6
7   for i = 1:length(numeroDeNeuronasOcultas)
8       for j = 1:numeroDeRepeticiones
9
10          numeroDeNeuronasOcultasEscalar = numeroDeNeuronasOcultas(i);
11          %%=====================================================================
12          %% Entrenamiento de la Red Neuronal
13          %%=====================================================================
14          [Red] = RedNeuronal(valoresDeEntrada, valoresDeSalida,
                numeroDeNeuronasOcultasEscalar);
15
16          %%=====================================================================
17          %% Clasificacion de los datos de entrada
18          %%=====================================================================
19          SalidaDeClasificacion = sim(Red,valoresDeEntrada);
20
21          %%=====================================================================
22          %% Error entre valores objetivo y valores de clasificacion
23          %%=====================================================================
24          Error = valoresDeSalida - SalidaDeClasificacion;
25
26          %%=====================================================================
```

```
27          %& Suma cuadrada de errores
28          %&=========================================================================
29          SSE = sse(Error, valoresDeSalida, SalidaDeClasificacion);
30
31          %&=========================================================================
32          %& Almacenar red neuronal (Red)
33          %&=========================================================================
34          RedTemp = Red;
35
36          temporal = SSE;
37
38          %&=========================================================================
39          %& Encontrar mejor Red Neuroanl
40          %&=========================================================================
41          if (temporal > mejor)
42              mejor = temporal;
43              RedBest = RedTemp;
44          end
45
46      end
47  end
48
49
50  end
```

Bibliografía

Ahmad, A. y Hamid, M. K. A. (2003). Pipeline leak detection system in a palm oil fractionation plant using artificial neural network. *International Conderence on Chemical and Bioprocess Engineering*.

Ahuja, R. K., Magnanti, T. L., y Orlin, J. B. (1993). *Network Flows: Theory, Algorithms and Applications*. Prentice Hall.

Alegre-Lopez, A. C. (2003). *Simulación de Redes Neuronales Artificiales*. Universidad Nacional del Nordeste. Facultad de Ciencais Exactas, Naturales y Agrimesura. Licenciatura en Sistemas.

Atiya, A. (1991). *Learning Algorithms for Neural Networks*. PhD thesis, California Institute of Technology.

Barberii, E. E. (1998). *El Pozo Ilustrado*. FONCIED, cuarta edition.

Bazaraa, M. S., Jarvis, J. J., y Sherali, H. D. (2010). *Linear Programming and Network Flows*. Wiley, third edition.

Begovich, O. y Pizzano-Moreno, A. (2008). Application of a leak detection algorithm in a water pipeline prototype: Difficulties and solutions. In *5th International Conference on Electrical Engineering, Computing Science and Automatic Control*, page 5.

Bell, S. (1999). A beginnerÂ´s guide to uncertainity of measurement. *National Pysical Laboratory*, 2.

Bird, R. B., Stewart, W. E., y Lightfoot, E. N. (2006). *Fenómenos de Transporte*. Limusa Wiley, segunda edition.

Bishop, C. M. (1995). *Neural Network for Pattern Recognition*. Claredon Press.

Bondy, J. A. y Murty, U. S. R. (1976). *Graph Theory With Applications*. North-Holland, first edition.

Campos-Lopez, O. A. (2008). Programa de cómputo para dimensionalizar medidores de flujo por presión diferencial en líquidos. Master's thesis, Instituto Politécnico Nacional.

CONACYT-SENER-Hidrocarburos, F. S. (2012). Convocatoria 2012-01 del fondo sectorial conacyt-sener-hidrocarburos. `http://www.conacyt.gob.mx/FondosyApoyos/Sectoriales/InvestigacionBasicaAplicada/FondosSectorialesEnergia/Hidrocarburos/Documents/Demandas-Especificas_2012-01.pdf`.

da Silva, H. V., Kazuyuki-Morooka, C., Rizzo-Guilherme, I., Pelaquim-Mendes, J., y da Fonseca, T. C. (2004). Leak detection using a fuzzy system. *ABCM Symposium Series in Mechatronics*, 1:625–634.

de Oviedo, U. (20012). Simulación del flujo de aire en una tubería. aplicación: Calibración de una placa y orificio. `http://www.unioviedo.es/Areas/Mecanica.Fluidos/docencia/_asignaturas/medida_modelizacion_flujos/04_Placa%20Orificio.pdf`.

de Sousa, J. N., Holanda-Sodré, C., de Lima, A. B., y Farrias-Neto, S. (2012). Numerical analysis of heavy oil-water flow and leak detection in vertical pipeline. *Advances in Chemical Engineering and Science*.

Demuth, H., Beale, M., y Hagan, M. (2009). *Neural Network Toolbox*. Matlab, 6 edition.

Diestel, R. (2005). *Graph Theory*. Springer-Verlag Heidelberg, third edition.

Eurolab (2006). Guide to the evaluation of measurement uncertainity for quantitative test results. Technical report, Eurolab.

Fuentes-Mariles, O., Palma-Nava, A., y Fuentes-Mariles, G. (2007). Calibración del factor de fricción y detección de fugas en una red cerrada de tuberias de agua potable. *Revista Iberoamericana de Sistemas, Cibernética e Informática*.

Gallegos-Alvarez, L. (2011). Programa para el cálculo de tuberías y bombas centrífugas en procesos de refinación. Master's thesis, Universidad del Istmo.

García-Guitiérrez, L. (1998). Teoría de la medición de caudales y volúmenes de agua e instrumental necesario disponible en el mercado. *Medida y evaluación de las extracciones de agua subterránea, ITGE*.

Geankopolis, G. J. (1998). *Procesos de Transporte y Operaciones Unitarias*. CECSA.

Gonzáles, M. A. (2000). *Flujo de Fluidos en Fase Líquida*.

Jian, F. y Huaguang, Z. (2004). Oil pipeline leak detection and localization using double sensors pressure gradient method. *Proceedings of the 5Ih World Congress on Intelligent Control Proceedings of the 5th Congress on Intelligent Control and Automation*.

Jian, F., Huaguang, Z., Tieyan, Z., y Liu, D. (2006). Detection algorithm and aplication based on work status evaluator. *Proceedings of the 6th World Congres on Intelligent Control and Automation*.

Loza-Guerrero, D. A. (2012). La metrología de flujo de líquidos en México. Technical report, Centro Nacional de Metrología.

Marín, J. M. (2010). *Introducción a las redes neuronales aplicadas*. Universidad Carlos III de Madrid.

Mashford, J., Silva, D., Marney, D., y Burn, S. (2009). An approach to leak detection in pipe networks using analysis of monitored pressure by support vector machine. In *Third International Conference on network and System Security*, page 6.

McCabe, W. L., Smith, J. C., y Harriott, P. (1991). *Operaciones Unitarias en Ingeniería Química*. McGraw-Hill, cuarta edition.

Mexicana, I. P. (2010). La refinación en México. `http://www.industriapetroleramexicana.com/2010/07/la-refinacion-en-mexico/`.

Meyers, R. A. (2004). *Handbook of Petroleum Refining Processes*. McGraw-Hill, third edition.

Perry, R. H. y Green, D. W. (2010). *Manual del Ingeniero Químico*. McGraw-Hill, 7 edition.

Preparedness, A. I. y Program, P. (1999). *Technical Review of Leak Detection Technologies*. The Program.

Rosen, K. H. (2004). *Matemática Discreta y sus Aplicaciones*. McGraw-Hill, quinta edition.

Rumelhart, D. E., Hinton, G. E., y Williams, R. J. (1986). Learning representations by back-propagating errors. *Letters to Nature*.

Sáez-Ruiz, S. J. y Font-Avila, L. (2001). *Incertidumbre de la Medición: Teoría y Práctica*. L and S Consultores C.A., primera edition.

Schmid, W. A. y Lazos-Martinez, R. J. (2000). Guía para estimar la incertidumbre de la medición. *Centro Nacional de Metrología*.

Speight, J. G. (2002). *Handbook of Petroleim Product Analysis*. Wiley-Interscience.

Torres-Robles, R. y Castro-Arellano, J. J. (2002). *Análisis y Simulación de Procesos de Refinación del Petróleo*. Alfaomega, primera edition.

Verde, C. (2001). Multi-leak detection and isolation in fluid pipelines. *Control Engineering Practice*.

Weisser, R. (2010). Propagación de incertidumbre. Technical report, Analítica Weisser.

West, D. B. (2000). *Introduction To Graph Theory*. Prentice Hall, second edition.

Zhang, J. y Xu, L. (1999). Real time pipeline leak detection on shell's north western etylene pipe. *REL Instrumentation Limited*.

Ficha autobiográfica

Ing. Luis Alejandro Benavides Vázquez

Candidato para el grado de Maestría en Ciencias
en Ingeniería de Sistemas

Universidad Autónoma de Nuevo León

Facultad de Ingeniería Mecánica y Eléctrica

Tesis:

Detección de fugas en los procesos de una refinería

Me llamo Luis Alejandro Benavides Vázquez. Nací en Monterrey, Nuevo León, México un 28 de septiembre de 1989 y tengo 24 años. Empiezo mis estudios de primaria en la ciudad donde he vivido la mayor parte de mi vida en San Nicolás de los Garza, Nuevo León, México, hasta quinto año de primaria. En el 2000 tuve la oportunidad de viajar a España con mi familia por un periodo de 4 años y es ahí donde termino mis estudios de primaria y realizo mis estudios de secundaria. Me siento muy agradecido de haber vivido esa experiencia porque aprendí muchas cosas, cultura y además de conocer a muchos compañeros y amigos que me hicieron más amena mi estadía.

Siempre fui un buen estudiante, desde siempre me gustaban las matemáticas y resolver problemas concernientes con ellas. Regresando en agosto el 2004, pierdo un

semestre por no llegar en fechas para presentar el examen de ingreso al bachillerato. Durante esos seis meses, me inscribí en la Escuela Electrónica Monterrey en un curso de reparación de computadoras, para no dejar de estudiar durante un periodo largo de tiempo.

A inicios del 2005 me aceptan en la Preparatoria No. 16 de la Universidad Autónoma de Nuevo León, en donde realizo mis estudios de bachillerato, habiéndome graduado para finales del 2006. En esta etapa de mi vida fue la decisiva para la elección de lo que quería ser de grande. Como mencioné, siempre me han gustado las matemáticas, y realizando mis estudios de bachillerato me llamaron la atención dos materias mas: la física y la química, por lo que me decidí a estudiar una carrera que incluyera estas materias. Además, en esta etapa conocí a muchos compañeros, los cuales todavía tengo contacto y fue una de las mejores épocas de mi vida, tanto en la parte emocional como en la parte educativa de mi vida.

Inicialmente con las dudas que tuve de qué quería hacer de mayor, estaba decidido a presentar en la Facultad de Física y Matemáticas de la Universidad Autónoma de Nuevo León, pero no contaban con una carrera que fuera de Físico-Matemáticas. Por esta razón decliné mi decisión y por que platicando con mi madre me dice que hay una carrera en la Facultad de Ciencias Químicas, que lleva por nombre Ingeniería Química, y me menciona que las bases de esta carrera son las tres materias que me gustan: matemáticas, física y química. Por lo que decidí presentar para entrar en esta carrera. Me aceptaron.

Inicié mis estudios universitarios en la Universidad Autónoma de Nuevo León en la Facultad de Ciencias Químicas, para inicios del 2007, para obtener mi título de licenciatura en Ingeniería Química a finales del 2011. Fue una de las épocas mas bonitas de mi vida. Fue la época en donde conocí a grandes compañeros, que día con día, hacíamos llevadero las clases con la convivencia en el aula.

Además, ahí conocí a mi novia Margarita Cadena, con la cual actualmente llevo 5 años de relación. Entre mi familia y ella me han apoyado y ayudado con las

dificultades que me encontré en la carrera, simplemente con su compañía, o consejos. Les doy las gracias por haberme ayudado en esta etapa de mi vida.

Gracias a que el Dr. Oscar Chacon en el último semestre de mi carrera, fue a dar una platica del Posgrado en Ingeniería de Sistemas, en donde nos dio a conocer que se trataba la maestría, me puso a pensar que estaba muy relacionado con lo que había estudiado. Por lo que tome la decisión de entrar a esta maestría. Presente los exámenes de conocimiento generales de la maestría y fui a las entrevistas como nos lo indicaron. Después de unos meses me dan la noticia de que soy aceptado.

Para el 2012 inicio mis estudios de maestría en el Posgrado de Ingeniería de Sistemas de la Facultad de Ingeniería Mecánica y Eléctrica de la Universidad Autónoma de Nuevo León. Se me presenta otra vez la oportunidad de salir del país, esta vez a realizar una estancia de investigación en Puerto Rico con el Dr. Mauricio Cabrera Ríos y su grupo de investigación. Durante mi estancia realice investigación acerca de un tema que trato en mi tesis, aprendí mucho y me ayudo a terminar mi tesis. Además conocí muchos amigos con los que pude compartir diferentes puntos de vista, lo cual me ayudo a desenvolverme en diferentes ámbitos de mi vida personal. Al terminar la estancia regreso a México, para volver a ver a familia, mi novia y mis amigos. Y ahora si terminar el largo camino de la tesis, escribirla y dándole los últimos detalles para hacer la defensa de la tesis lo antes posible.