PEDECIBA Informática
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay

Reporte Técnico RT 10-08

Optimización de P4P utilizando algoritmos genéticos para GoalBit

Claudia Rostagnol

2010

Optimización de P4P utilizando algoritmos genéticos para GoalBit
Claudia Rostagnol
ISSN 0797-6410
Reporte Técnico RT 10-08
PEDECIBA
Instituto de Computación – Facultad de Ingeniería
Universidad de la República
Montevideo, Uruguay, 2010

Optimización de P4P utilizando Algoritmos Genéticos para GoalBit

Claudia Rostagnol
Facultad de Ingeniería, UdelaR

***Abstract*—El uso de aplicaciones a través de Internet continua creciendo día a día. Una gran cantidad de estas aplicaciones son las denominadas *peer-to-peer* (P2P), y están consumiendo un importante porcentaje (entre 60 y 70 porciento) del ancho de banda de los proveedores de servicio a Internet (Internet Service Providers - ISPs).**

Este incremento en el consumo del ancho de banda global no sólo puede ocacionar problemas económicos a los ISPs, sino que puede traer problemas a las aplicaciones P2P que podrían verse bloqueadas por políticas de los ISPs. Para aplicaciones de transferencia de archivos (música, vídeos, juegos, etc.) puede que una limitante en el ancho de banda no genere grandes problemas, pero para aplicaciones de *streaming* en vivo de video o televisión, como por ejemplo GoalBit, el ancho de banda es crucial para el cumplimiento y calidad del servicio al usuario. Resulta entonces conveniente poder establecer un mecanismo por el cual las aplicaciones P2P y los ISPs puedan compartir información para lograr sus objetivos (evitar altos consumos de ancho de banda en los ISP y ofrecer un buen servicio y calidad a los usuarios de las aplicaciones P2P). Con este objetivo surge el proyecto P4P: *Proactive Provider Participation for P2P*, llevado adelante por la Universidad de Yale, Estados Unidos. En este artículo presentamos una aplicación del P4P para GoalBit, utilizando un Algortmo Genético Multiobjetivo para optimizar el uso de los recursos, minimizando el tráfico entre los ISPs al mismo tiempo que se maximiza el contenido recibido por los clientes (y por ende la calidad del *streaming*).

***Index Terms*—GoalBit, P2P, P4P, Algoritmos Genéticos, Problemas Multiobjetivo.**

I. Introducíon

Un problema básico pero no menos importante es cómo las aplicaciones utilizan los recursos, entre ellos la red controlada por los ISPs. Este problema particular se conoce como problema del control de tráfico [10] y puede tener gran impacto en el rendimiento (*performance*) de las aplicaciones, la eficiencia y economía de los ISPs y en la complejidad del sistema.

Dada la creciente cantidad de aplicaciones P2P en la red, los ISPs han incorporado nuevas técnicas de control de tráfico, algunas de las cuales no resultan ser muy colaborativas con las aplicaciones P2P. Por ejemplo, para aplicaciones específicas con protocolo abierto, se pueden instalar dispositivos de *caching* para controlar o reducir el consumo de ancho de banda de la aplicación. Si las aplicaciones P2P tuvieran conocimiento de la topología, estado y políticas de la red, sin necesidad de utilizar sus propios mecanismos para ello (como ingeniería inversa), entonces estas podrían adaptarse de forma de evitar saturar enlaces de los ISPs, y así lograr mejoras no sólo para sí mismas sino para los ISPs. Esto parece beneficiar a ambas partes (las aplicaciones P2P y los ISPs) y en esto se ha basado el grupo de Yale para realizar su *framework* P4P, el cual será presentado brevemente en la sección II.

El presente trabajo busca incorporar el P4P a la plataforma GoalBit [2] para mejorar el uso que ésta realiza de los recursos de la red. GoalBit se encuentra implementada en base a BitTorrent [1], [9], otra reconocida aplicación P2P que ocupa aproximadamente el 35% del tráfico en Internet [7]. Su objetivo es distribuir televisión y vídeo en vivo a sus clientes en base a una red *peer-to-peer*, por lo que es crucial asegurar que cada cliente recibirá el contenido en tiempo y forma, ya que de otra manera la aplicación sería inútil (estas aplicaciones se conocen como aplicaciones de tiempo real). En la seccion III presentamos brevemente GoalBit.

Para poder mejorar GoalBit mediante el uso de P4P, se utilizará un Algoritmo Genético (AG) que optimice el consumo del ancho de banda entre los ISPs y a su vez el contenido recibido por los clientes de la aplicación. En la sección IV se describe el problema con mayor profundidad y se presenta el modelo matemático que se pretende optimizar. A continuación, en la sección V, se describe el Algoritmo Genético Multiobjetivo que será utilizado en dicha tarea, incluyendo sus operadores, representación de las soluciones, funciones objetivo y datos de prueba. Los resultados obtenidos a partir de la ejecución de dicho algoritmo se presentan en la sección VI. Por último, en la sección VII, se presentan las conclusiones del trabajo y los trabajos a futuro.

II. P4P: Proactive Provider Participation for P2P

P4P es una iniciativa internacional iniciada por la Universidad de Yale, en la que participan varias compañías, entre ellas operadoras como Telefónica, Verizon o AT&T, y proveedores P2P como BitTorrent y Pando Networks. Los dos objetivos principales de P4P son [10]: (1) facilitar a las aplicaciones de la red, en particular las aplicaciones P2P, alcanzar el mejor rendimiento y eficiencia en el uso de los recursos de la red; y (2) permitir a los ISPs alcanzar la eficiencia en el uso de sus recursos para satisfacer los requerimientos de las aplicaciones, reducir sus costos e incrementar sus ingresos.

Los resultados hasta el momento muestran que el P4P duplica la velocidad de las descargas a través del esquema adaptado del P2P [10]. En el esquema tradicional del P2P, los usuarios que buscan archivos se conectan a nodos que actúan como "directorios" de contenido (*trackers*), poniéndolos en contacto de forma "indiscriminada" con quienes ofrecen el archivo solicitado. El P4P añade al esquema nuevos nodos pertenecientes a los ISPs, los cuales informan quién está más

cerca del usuario solicitante. Dado que el protocolo IP no implementa funcionalidades sobre el estado de la red en cuanto a carga y ubicación de los dispositivos que hacen uso del protocolo, son los ISPs quienes mantienen y distribuyen esta información. La misma debe ser almacenada en nodos para tales efectos, que en P4P son llamados ***iTrackers***, los que se encuentran en el nivel de aplicación del protocolo TCP/IP. La figura 1 muestra un esquema general del P4P.

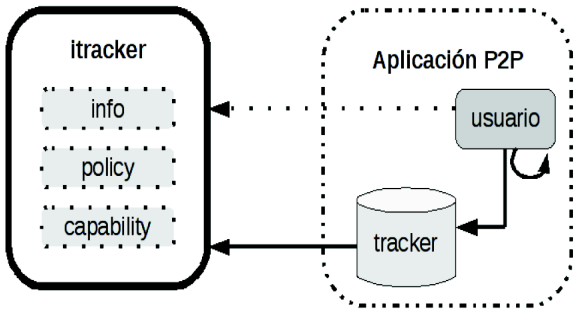

Fig. 1. Arquitectura del *framework* P4P

Cuando un usuario solicita un archivo a la aplicación P2P, ésta consulta al *iTracker* de P4P con la dirección IP de dicho usuario. Si el P4P contiene información sobre dicho usuario, retornará a la aplicación los usuarios dentro del mismo proveedor a los que el usuario podría solicitar el recurso que necesita. De esta forma la aplicación P2P siempre intentará poner en contacto usuarios "cercanos" que pertenezcan a un mismo ISP. Si esto no es posible, entonces sí se establecerá una cominicación entre usuarios de diferentes ISPs. Con este mecanismo, los usuarios se ven beneficiados con una mayor velocidad de descarga, ya que en general recibirán contenido desde usuarios de un mismo ISP (cercanos geográficamente y evitando saltos de enlace); mientras que los ISPs se benefician de un menor consumo de ancho de banda y por tanto menor costo económico.

Es importante resaltar que P4P no es un protocolo P2P, sino un medio para que los ISPs optimicen el tráfico de las redes P2P. Tampoco impone a las aplicaciones P2P determinadas conexiones o restringe otras, simplemente aporta información útil a la aplicación para que ésta pueda optimizar sus enlaces. Es responsabilidad de la aplicación utilizar o no dicha información.

III. GoalBit

Goalbit es el primer sistema P2P de código abierto (*open source*) para distribución de flujos de vídeo en tiempo real sobre Internet. La forma de realizar esta distribución se basa en BitTorrent [9], donde el contenido es dividido en varios flujos (*streams*) que son enviados y recibidos por diferentes clientes (*peers*). Cada cliente recibe varios de estos flujos de diferentes orígenes y es capaz de reconstruir el *streaming* recibido por partes desde esos flujos independientes.

A. Conceptos

En BitTorrent, existe una entidad o nodo central denominada ***tracker*** que contiene la lista de *peers* que se encuentran descargando un archivo (contenido) determinado. A través de esta lista, cada *peer* conoce en cada momento el subconjunto de otros *peers* que se encuentran descargando el mismo archivo (este subconjunto se conoce como ***swarm***).

Cada archivo es distribuido por la red en piezas denominadas ***chunks***. Cada *peer* conoce quien posee cada *chunk* en su *swarm* y realiza pedidos punto a punto para conseguir dichas piezas. Algunas piezas son más "raras" que otras, y por tanto más preciadas, por lo que los *peers* tratan de conseguir dichas piezas primero (esto se conoce como política ***rarest-first***). A su vez los *peers* envían primero los *chunks* a sus pares que les envían con mayor tasa, aplicando así una política ***tit-for-tat - TFT***.

Goalbit no puede utilizar este mecanismo tal como lo implementa BitTorrent, ya que el contenido de GoalBit se compone de flujos continuos en vivo y no de archivos con un tamaño fijo que sólo pueden ser abiertos luego de completada su descarga. De todas formas, los conceptos aquí presentados sí son utilizados por GoalBit. La forma cómo el protocolo de BitTorrent es modificado para contenido en vivo se explica en [3].

B. Contenido en vivo

Las aplicaciones P2P para contenido en vivo (**P2PTV**) poseen mayores restricciones que las de distribución de archivos, ya que son aplicaciones de tiempo real y los clientes puede que sólo se conecten unos pocos minutos.

Para el caso de GoalBit, el contenido puede ser generado mediante diferentes dispositivos (tarjeta capturadora, cámara Web, otro *stream* http/mms/rtp, un archivo, etc.), "encodeado" (*encoded*) a diferentes formatos y "muxeado" (*muxed*) en diferentes contenedores o *muxers*. El flujo es encapsulado en un *GoalBit Packetized Stream (GBPS)* [3] a partir del cual se generan los *chunks* que son distribuidos mediante el *GoalBit Transport Protocol (GBTP)* [3].

C. Plataforma GoalBit

Goalbit se compone de 4 tipos diferentes de nodos o entidades [1] (ilustrados en la figura 2):

1) ***Broadcaster***: Es el responsable de obtener el contenido desde algún dispositivo e inyectarlo en la red.
2) ***Tracker***: Contiene información sobre los *peers* que se encuentran conectados en la red. Es análogo al *tracker* en BitTorrent.
3) ***Peers***: Son los usuarios finales del sistema, que consumen y comparten contenido entre sí.
4) ***Super-Peers***: Son *peers* especiales, con mayor disponibilidad y capacidad de ancho de banda, que ayudan principalmente en la distribución inical del contenido.

[1]El nodo Web-Server no es parte de los componentes básicos de la plataforma y puede estar o no. Es un contenedor de archivos GoalBit (`.goalbit`) que contienen información de contenido (similar a los `.torrent` de BitTorrent)

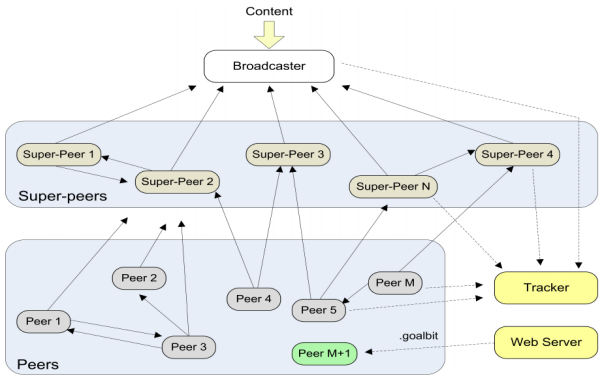

Fig. 2. Arquitectura de la plataforma GoalBit

D. Comunicación

A continuación se presenta muy brevemente algunos conceptos relevantes sobre la comunicación que se realiza entre las diferentes entidades de la plataforma. Esta información puede ser profundizada en [3].

1) Comunicación con el *tracker*: La comunicación entre el *tracker* y los *peers* se realiza de dos formas diferentes: la inicial, cuando el *peer* ingresa a la red; y durante la "estadía" del *peer* en la red. Ambos casos son llevados a cabo sobre protocolo HTTP/HTTPS y se basan en un único tipo de mensaje llamado *announce*. El *peer* envía información tal como su identificador, si está ingresando o saliendo de la red, el contenido que solicita, el puerto donde recibe los mensajes, la última pieza recibida si corresponde, las tasas de bajada y subida, etc. Por su parte el *tracker* responde informando su identificador, la lista de *peers* a los que se puede solicitar el contenido deseado, el máximo y mínimo identificador de pieza que puede ser solicitado, entre otros.
2) Comunicación entre *peers*: La comunicación entre *peers* comienza una vez que el *tracker* ha informado al nuevo *peer* cual será su *swarm* (su lista de *peers*). Esta comunicación se realiza a través del protocolo TCP/IP, donde los mensajes enviados pueden ser de dos tipos: mensajes de control y mensajes de solicitud. Los mensajes de control sirven para informar el estado de un *peer* a otro (conocer que piezas posee otro *peer* - BITFIELD, WINDOW-UPDATE, etc. -, informar si aún está conectado - KEEPALIVE -, etc.). Los mensajes de solicitud son utilizados para pedir piezas (habilitar la solicitud de piezas - UNCHOKE -, bloquear la solicitud de piezas - CHOCKE -, solicitar una pieza determinada a un *peer unchocked* - PIECE -, realizar el pedido de *unchocke* - INTERESTED -, etc.)

IV. El problema GoalBit + P4P

Una vez presentados ambos conceptos (P4P y GoalBit) se describe cuál es el problema que se desea atacar y cómo. En particular se verá por qué y para qué incorporar P4P en GoalBit, así como el modelo matemático detrás de P4P.

A. Motivación

Los sistemas P2P, como por ejemplo GoalBit, poseen una muy alta flexibilidad para configurar y organizar su red de usuarios y controlar el tráfico que circula por ella. Un usuario interesado en una pieza de contenido puede obtenerla de muchas fuentes, pero no todas serán igualmente beneficiosas para él, la aplicación o el ISP. Se debe, por tanto, poner importante cuidado e inteligencia en la selección de pares o fuentes de contenido de los cuales cada usuario obtiene las piezas. En particular, la selección de piezas de GoalBit está determinada por su *tracker* y por la política *tit-for-tat* como se mencionó anteriormente en la sección III.

Incorporando P4P al esquema GoalBit y suponiendo que la red cuenta con K *swarms* en un ISP, cada *peer* del *swarm* obtiene de su *tracker* un identificador del *swarm* ($swarmID$), y a su vez, dicha información es reportada al *iTracker* P4P. De esta forma el *iTracker* mantiene un registro de los *peers* en un determinado *swarm*, incluyendo el número de *peers* en la misma región (*Point of Presence* - *PoP*), así como las capacidades de bajada y subida de cada uno. Con estos datos el *iTracker* mapea los PoPs con PIDs (ID de una región o PoP, se utiliza para preservar la privacidad de los datos del ISP) y construye una topología abstracta de la red P2P en forma de grafo ($G = (V, E)$), donde cada nodo representa un PID y las aristas son *links* entre los PIDs. Además, se cuenta con la información relacionada con las aristas como por ejemplo: la cantidad b_e de tráfico de fondo (*background traffic*, tráfico perteneciente a otras aplicaciones) de cada arista $e \in E$, la capacidad c_e de cada arista $e \in E$ y un indicador $I_e(i, j)$ de si la arista $e \in E$ se encuentra en la ruta que une los nodos i y j.

Si GoalBit puediese contar con esta información, entonces el *tracker* tendría más elementos para poder determinar la lista de *peers* que le entrega a cada nuevo *peer* que se conecta a la red. Éste es el desafío que se plantea con la incorporación de P4P a GoalBit.

B. Modelo matemático para P4P

A continuación se formaliza el modelo matemático detrás de P4P para obtener las funciones que se desean optimizar.

Conceptos básicos utilizados:

1) Se tienen K contenidos descargados de forma P2P.
2) Datos conocidos por el *iTracker* del P4P (datos del problema)
   a) Los *peers* de la red son agrupados en subconjuntos llamados PID.
   b) El conjunto de aristas físicas del ISP (*links* entre los PIDs) será denotado por E.
   c) La capacidad remanente de transferencia de bajada de *peers* del PID_i para el contenido k será indicada por d_i^k.
   d) La capacidad remanente de transferencia de subida de *peers* del PID_i para el contenido k será indicada por u_i^k.
   e) La cantidad de tráfico de fondo de cada arista $e \in E$ será denotada por b_e. Esta cantidad será un valor entre 0 y 1, indicando el porcentaje de la capacidad

del enlace que se encuentra ocupado por tráfico de otras aplicaciones.

f) La capacidad real de cada arista $e \in E$ será denotada por c_e.

g) La capacidad virtual de cada arista *interdomain* $e \in E$ será denotada por v_e.

h) Se utiliza un indicador $I_e(i,j)$ para saber si la arista $e \in E$ se encuentra en la ruta entre los nodos i y j.

3) Datos que se deben calcular (objetivo del problema)

a) La cantidad de tráfico de bajada de *peers* del PID_i a *peers* del PID_j para el contenido k será indicada por t_{ij}^k. Ésta es la variable objetivo del problema.

El objetivo de las redes P2P es la maximización de la transferencia total para cada contenido (maximizar el *Throughput* del sistema), para lo cual tenemos la siguiente ecuación[10]:

$$max \sum_i \sum_{j \neq i} t_{ij}^k \tag{1}$$

sujeto a

$$\sum_{j \neq i} t_{ij}^k \leq u_i^k \; ; \forall PID_i$$

$$\sum_{j \neq i} t_{ij}^k \leq d_j^k \; ; \forall PID_j$$

$$t_{ij}^k \geq 0 \; ; \forall i \neq j$$

El objetivo de los ISPs es la minimización de las transferencias en sus enlaces mas costosos (minimizar el *Maximum Link Utilization - MLU*), para lo cual tenemos la siguiente ecuación [10]:

$$min \; max_{e \in E} \; b_e + \sum_k \sum_i \sum_{j \neq i} \frac{t_{ij}^k . I_e(i,j)}{c_e} \tag{2}$$

sujeto a

$$t_{ij}^k \geq 0 \; ; \forall k, i \neq j$$

$$\forall e_{interdomain} \sum_k \sum_i \sum_{j \neq i} t_{ij}^k * I_e(i,j) \leq v_e$$

Ambos objetivos influyen en la determinación del tráfico óptimo entre PIDs (es decir en la determinación de los t_{ij}^k). Se buscará una solución de compromiso entre ambos objetivos.

C. Implementación para GoalBit

En GoalBit, el modelo P4P será implementado en el *tracker*. Esto signica que el *tracker* deberá poseer toda la información para ejecutar una metaheurísitca (en este caso un Algoritmo Genético) que encuentre una solución al modelo. Como resultado, cada determinado tiempo se actualizarán los valores t_{ij}^k. Con estos valores el *tracker* calculará los siguientes pesos:

$$w_{ij}^k = \frac{t_{ij}^k}{\sum_{j \neq i} t_{ij}^k} \; ; \forall PID_i, \forall PID_j, i \neq j \tag{3}$$

donde w_{ij}^k representa el porcentaje de todo el tráfico saliente del PID_i que va al PID_j. Para que los *peers* puedan comenzar a intercambiar piezas primero deben conocerse. Cuando el *peer* p_1 se conecta a la red para obtener el contenido k, éste le solicita al *tracker* una lista de otros *peers* (con un máximo de 100) que están viendo el contenido k. También puede solicitar dicha lista durante la ejecución si considera que la lista que ya posee es pequeña. En estos casos, suponindo que $p_1 \in PID_i$, el *tracker* enviará una lista de *peers* en el PID_j con las tuplas (IP:puerto, w_{ij}^k).

El algoritmo de *optimistic unchoking* de cada *peer* elegirá al azar entre sus *peers choked* de forma ponderada según los w. Por tanto estadísticamente los *peers* se comunicarán respetando las ponderaciones de los w y de esta forma cumplirán con las cantidades de tráfico t_{ij}^k deseadas.

El modelo P4P contempla los intereses globales de la red P2P y los intereses del ISP pero no contempla directamente los intereses personales de cada *peer*. Es sabido que los *peers* en una red P2P son muy heterogéneos en sus capacidades de ancho de banda. BitTorrent determina una justicia implicita con aquellos *peers* que no ofrecen suficiente contenido a la red, donde los que menos colaboran más lento descargan. En cambio en una red P2PTV, si un *peer* no mantiene la tasa de descarga significa que no puede recibir el contenido (percibiendo la red como de muy mala calidad y malgastando completamente los recursos que se le asignaron). Es necesario entonces asegurar una calidad mínima a la mayoría de los usuarios, y por tanto, si se tiene que dejar de enviar contenido a algún *peer*, que sea a los que menos recursos ofrecen.

Cuando un *peer* respeta los pesos w en su *optimistic unchoking*, la red le impone un sesgo sobre con quienes hablar. Si un *peer* se encuentra recibiendo con mala calidad es de esperar que comience a pensar más en mejorar esta situación que en acompañar un crecimiento ordenado de la red (producto del modelo P4P). Como el *tracker* conoce las calidades de cada *peer* en cada momento, puede decidir que aquellos que reciben con peor calidad no utilicen los pesos w hallados con P4P sino simplemente usar una distribución uniforme como es el BitTorrent normal. Si se hace esto, luego de un tiempo el *peer* con mala calidad se conectará a los *peers* que le den más piezas y no a los que topológicamente estén cerca. Más aún, es posible que el *tracker* frente a esta situación le envíe algunos *peers* con pesos ficticios altos de los cuales se conoce *a priori* su disposición a dar piezas, es decir los *Super-peers*.

Se plantea el siguiente esquema de cálculo de pesos considerando las calidades percibidas por los *peers*: El *peer* $p_1 \in PID_i$ está evaluando hacer *optimisitc unchoking* con el *peer* $p_2 \in PID_j$. Usando el modelo P4P, p_2 sería elegido con una probabilidad w_{ij}^k, mientras que usando el esquema de BitTorrent, sería elegido con una probabiliad de $\frac{1}{||swarm||}$. Si el *peer* está viendo con una calidad qp_1 y el resto de los *peers* en su *swarm* tienen una calidad promedio q, se define la probabilidad de *optimisitc unchoking* $w_{p_{1_{ij}}}^k$ como:

$$q_{p_1} < q \Longrightarrow w_{p_{1,ij}}^k = \frac{1}{\parallel swarm \parallel} \tag{4}$$

$$q_{p_1} \geq q \Longrightarrow w^k_{p_{1,ij}} = (\frac{w^k_{ij} - \frac{1}{\|swarm\|}}{1-q})(q_{p_1} - q) + \frac{1}{\| \ swarm \ \|} \quad (5)$$

Este sería un posible esquema para incorporar P4P a GoalBit. Cabe aclarar que este modelo aún no ha sido implementado, por lo que no se cuenta con datos reales de los beneficios de P4P en GoalBit. A continuación se presenta el algoritmo genético que se utiliza para optimizar el problema P4P y obtener los datos necesarios para calcular los pesos w aquí mencionados.

V. Algoritmo Genético Multiobjetivo para P4P

Dada la naturaleza multiobjetivo del problema, resulta intuitivo pensar en la utilización de un Algoritmo Genético Multiobjetivo (MOEA) para resolverlo (en [6] se presenta una introducción a los problemas de optimización multiobjetivo - MOP).

No se ha encontrado en la literatura referente a P4P algún estudio que aborde el tema desde un punto de vista multiobjetivo. El grupo que propone el *framework* P4P realiza una simplificación del modelo matemático a una sola función objetivo bi-nivel, con varias restricciones. Resulta entonces novedoso probar un enfoque multiobjetivo para poder comparar los resultados y ofrecer además diferentes soluciones posibles, que podrán ser testeadas o seleccionadas dependiendo de la situación, otra información adicional o algún mecanismo de toma de decisiones externo al AG.

Los problemas multiobjetivo se caracterizan por tener una familia o conjunto de posibles alternativas o soluciones que, en principio, deben ser consideradas equivalentes ante la ausencia de mayor información que permita una selección o preponderancia de unas sobre otras. Estas soluciones del espacio de búsqueda del problema no pueden ser todas optimizadas o mejoradas al mismo tiempo, ya que una mejora en una de ellas implica una desmejora en otras. Esto se conoce como ***optimalidad de Pareto*** y el conjunto de soluciones encontrado es el ***óptimo de Pareto***.

Más formalmente (tomando como ejemplo la minimización, pero siendo igualmente aplicable para maximización) definimos la *optimalidad de Pareto* como sigue [5]:

$$f(X) = (f_1(X), ..., f_n(X))$$

donde n es la cantidad de componentes o funciones f_k, con $k = 1, ..., n$, que componen el vector de funciones f, aplicado a un vector de variables X en un universo U. Entonces, un vector $X_u \in U$ se dice óptimo de Pareto si y sólo si no existe $X_v \in U$ para el cual $v = f(X_v) = (v_1, ..., v_n)$ domine a X_u, o lo que es lo mismo

$$\forall i \in \{1, ..., n\} \ , v_i \leq u_i \wedge \forall i \in \{1, ..., n\} \ , v_i < u_i$$

Al conjunto de valores funcionales ($f(X)$) del óptimo de Pareto (X_u) se le conoce como ***frente de Pareto***.

El MOEA a implementar intentará encontrar el óptimo de Pareto para el problema de P4P planteado. Dado que no se conoce el frente de Pareto para este problema, se deberá aproximar el mismo experimentalmente, para luego poder aplicar el algoritmo y optimizar las funciones de forma que la relación entre ellas se aproxime lo más posible al frente.

A. Representación de la solución

El resultado del algoritmo debe dar un conjunto de soluciones posibles, donde cada solución debe contener un valor óptimo para la tasa de transferencia (t^k_{ij}) de cada contenido k entre un par de PIDs (PID_i y PID_j). La tasa t^k_{ij} es un valor real acotado entre 0 y MAX-TASA, por tanto, cada solución se compondrá de varios vectores de reales (uno por cada arista PID_i-PID_j).

Dado que este problema ya ha sido abordado anteriormente por otro grupo de investigacion cercano, se ha reutilizado la representación por ellos propuesta así como el código que permite realizar la carga de datos desde archivos a la estructura de la representación de las soluciones. Dicha estructura se compone de un vector de 3 dimensiones, donde una de las dimensiones representa los contenidos, y las otras dos los nodos que conforman el grafo P4P. De esta forma tendremos una matriz *nxn*, con *n* la cantidad de nodos, por cada contenido k. Cada entrada ij de dicha matriz contendrá el valor del tráfico t^k_{ij} para el contenido k entre los nodos i y j. Si entre dos nodos no hay transferencia el valor será 0. La figura 3 muestra esquemáticamente la representación que se utiliza para la matriz de transferencia de un determinado contenido k. La figura 4 muestra la representación general de una solución para todos los contenidos.

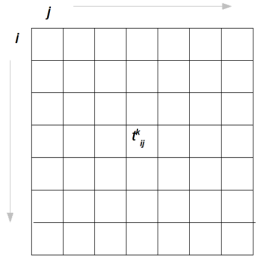

Fig. 3. Matriz de transferencia para un contenido dado (parte de la representación de una solución del AG)

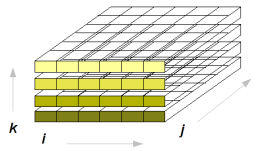

Fig. 4. Representación completa de una solución del AG

El valor máximo de cada entrada de la matriz (MAX-TASA) podría ser el máximo valor real representable por la

computadora donde se ejecuta el algoritmo, pero dado que esto genera un espacio de búsqueda muy grande y dadas las restricciones presentadas en las ecuaciones 1 y 2, el valor de MAX-TASA se calcula a partir de los datos de entrada del problema como: el mínimo entre la capacidad de transferencia de bajada (d_i^k) y la capacidad de transferencia de subida (u_i^k) del PID_i, ya que ningún tráfico podrá ser mayor que la capacidad de subida del PID que envía ni la capacidad de bajada del PID que recibe.

B. Funciones objetivo

Las funciones objetivo de este problema coinciden con las dos ecuaciones presentadas en el modelo matemático:

$$f_1(X) = max \sum_i \sum_{j \neq i} t_{ij}^k$$

$$f_2(X) = min\ max_{e \in E}\ b_e + \sum_k \sum_i \sum_{j \neq i} \frac{t_{ij}^k . I_e(i,j)}{c_e}$$

ambas con sus respectivas restricciones. Estas funciones son aplicadas a cada individuo de la representación elegida y de esta forma determinar la optimalidad de las mismas.

C. Operadores

El **operador de cruzamiento** se aplica según la probabilidad de cruzamiento p_c, tomando dos soluciones p_1 y p_2 del *maiting pool* para generar dos soluciones hijas h_1 y h_2 utilizando cruzamiento BLX (Eshelman e Shaffer,1993). Se crean dos nuevas soluciones y cada elemento de dichas soluciones es generado de la siguiente forma:

$$h_1[i,j,k] = \gamma * p_1[i,j,k] + (1-\gamma) * p_2[i,j,k]$$
$$h_2[i,j,k] = (1-\gamma) * p_1[i,j,k] + \gamma * p_2[i,j,k]$$

El **operador de mutación** se aplica con probabilidad de mutación p_m, tomando una solución p_1 para generar una nueva solución h_1 utilizando una distribución Normal ($N(\mu, \sigma)$). La nueva solución hija toma al padre y modifica todas las posiciones de una matriz para un contenido dado (la estructura de 3 dimensiones se modificada solo en 2 de ellas, seleccionando al azar la posición de la tercera dimensión). Una vez elegida al azar la posición k de un contenido, los valores de transferencia para dicho contenido se modifican de la siguiente forma:

$$h_1[i,j,k] = p_1[i,j,k] + N(\mu, \sigma)$$
$$h_1[i,j,t] = p_1[i,j,t]\ ; \forall t \neq k$$

D. Datos de entrada, Configuración de parámetros y Frente de Pareto

Se cuenta con un buen conjunto de datos de prueba reales, correspondientes a los datos que proporcionaría el *iTracker* del P4P: topología de la red, capacidades de subida y bajada de los PIDs, etc. Estos datos han sido recabados por un ISP local (Antel) y procesados para generar los archivos de entrada del algoritmo. Parte de estos datos ha sido estimada, ya que no se cuenta con información real referente a otros ISPs. Por otro lado, sólo se incluye información relevante para el ISP local, de forma de poder optimizar el uso del enlace *interdomain* del mismo.

De todo este conjunto de datos (12 archivos), se utiliza aproximadamente el 15% (2 archivos) para calibrar el algoritmo, ajustar los parámetros del mismo y obtener el frente de Pareto del problema.

Los parámetros a calibrar son los siguientes:

1) probabilidad de cruzamiento p_c: se prueba con los valores 0.7, 0.8 y 0.9.
2) probabilidad de mutación p_m: se prueba con los valores 0.01, 0.05 y 0.001.
3) cantidad de iteraciones del algoritmo: se prueba con los valores 5000, 10000 y 25000.

Dado que tenemos 3 parámetros para calibrar, y cada uno con 3 valores posibles, tenemos 27 combinaciones diferentes para probar. Cada combinación es ejecutada 10 veces con cada uno de los 2 archivos, almacenando las soluciones obtenidas en cada caso. Durante estas ejecuciones se va obteniendo el frente de Pareto del problema de la siguiente forma:

1) en la primer ejecución, no tenemos frente aún, así que todas las soluciones no dominadas pasan a formar parte del frente de Pareto.
2) en la siguiente ejecución, se toman las soluciones no dominadas de la ejecución, se unen a las que ya están en el frente de Pareto y se actualiza el frente de Pareto con las soluciones no dominadas de la unión.
3) se repite el paso anterior hasta finalizar las ejecuciones.

Una vez que se finalizan las ejecuciones, se cuenta con el frente de Pareto del problema y, por tanto, se pueden calcular las métricas para determinar que configuración de parámetros es la más apropiada. La gráfica del frente de Pareto obtenido se muestra en la figura 5.

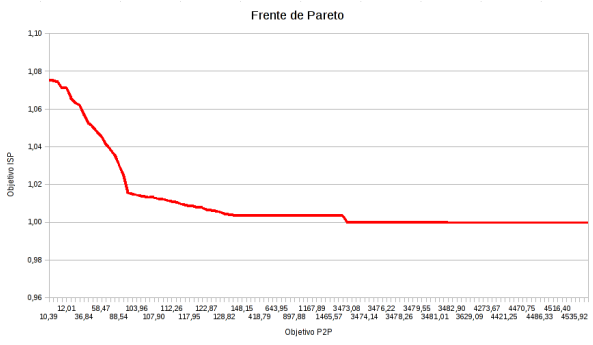

Fig. 5. Frente de Pareto para el problema P4P

Las métricas utilizadas para la evaluación de la calibración son las siguientes: Hipervolumen, Additive Epsilon, Distancia Generacional, Distancia Generacional Inversa y Spread. En la figura 6 se pueden observar las gráficas para cada métrica.

La figura 7 muestra parte de los resultados (los mejores) de dichas métricas para las ejecuciones de calibración realizadas. Como se puede observar, la configuración 24 es la mejor para 3 de las 5 métricas analizadas, y además está por encima del promedio en las otras 2 métricas, por lo que se concluye que es la configuración más adecuada para el problema.

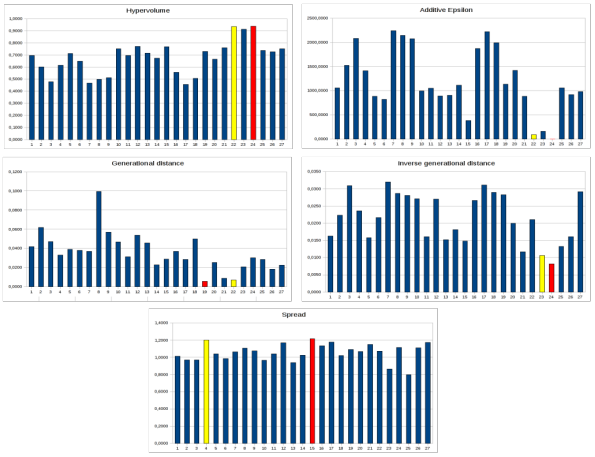

Fig. 6. Gráficas de las métricas evaluadas durante la calibración del AG

	hypervolume	execution	epsilon	execution	gen_distance	execution	inv_gen_distance	execution	spread	execution
1	0,9388	24	0,0111	24	0,0056	19	0,0082	24	1,2169	15
2	0,9356	22	92,3071	22	0,0071	22	0,0106	23	1,2021	4
3	0,9130	23	160,2611	23	0,0089	21	0,0117	21	1,1787	17
4	0,7725	12	381,6420	15	0,0183	26	0,0133	25	1,1755	27
5	0,7695	15	820,1611	6	0,0208	23	0,0149	15	1,1690	12
6	0,7616	21	885,7487	21	0,0226	27	0,0152	13	1,1591	21
7	0,7532	27	885,8560	5	0,0228	14	0,0158	5	1,1362	16
8	0,7523	10	892,1076	12	0,0255	20	0,0160	26	1,1136	24
9	0,7383	25	901,7029	13	0,0283	17	0,0161	11	1,1120	26
10	0,7297	19	921,6996	26	0,0286	25	0,0163	1	1,1052	8
11	0,7283	26	980,9824	27	0,0287	15	0,0182	14	1,0908	19
12	0,7172	13	992,8392	10	0,0302	24	0,0200	20	1,0771	9
13	0,7134	3	1049,5899	11	0,0314	11	0,0211	22	1,0736	22

Fig. 7. Resumen de las métricas de calibracion del AG

Los datos de la configuración 24, y por tanto los que serán utilizados de aquí en más, son los siguientes:

1) probabilidad de cruzamiento p_c: 0.7
2) probabilidad de mutación p_m: 0.05.
3) cantidad de iteraciones del algoritmo: 25000.

E. Población inicial

La población inicial se genera aleatoriamente, respetando las restricciones inherentes al problema P4P descritas en el modelo matemático presentado en la sección IV-B.

F. Factibilidad de las soluciones

Se define utilizar sólo ***soluciones factibles*** (que se encuentran dentro del espacio de búsqueda del problema y cumplen con las restricciones del mismo). Luego de la aplicación de los operadores del algoritmo, se debe determinar si las soluciones hijas son factibles o no. Para aquellas soluciones no factibles se realiza un proceso de factibilización de la solución, modificando levemente los valores para cumplir con las restricciones.

Para las restricciones del objetivo P2P, el ***proceso de factibilicación*** toma cada matriz de nodos de la solución y recorre las filas para comprobar las restricciones (la suma the *thoughput* debe ser mayor o igual que cero y menor que la capacidad de *upload* del nodo de la fila que se está comprobando). Cuando una fila no cumple la restricción del *upload* se calcula el exceso (*throughput* - *upload*) de la fila, y se decrementa aleatoriamente un valor de la fila en un porcentaje (determinado aleatoriamente) del exceso. Este proceso se continúa hasta que el exceso sea menor o igual a 0 (la figura 8 muestra el pseudocódigo del proceso de factibilización de una fila). Una vez comprobadas todas las filas de una matriz dada, se pasa a comprobar las columnas, calculando el exceso con respecto a la capacidad de download del nodo correspondiente a la columna (*throughput* - *download*). Se aplica un proceso idéntico al de las filas pero para las columnas.

```
//P2P factibilization process for row i and content k

double exceso = throughput - upload;
double diff = exceso;

while( diff > 0 ){

.       int index = rand( 0, cantNodos );  // get a randomic position in the row
.       while ( index == i ) //we need a PID different than PID_i
.       .       index = rand( 0, cantNodos ); // get a randomic position in the row
.
.       double percent = rand( 0, 1 );

.       double value = data[i,index,k] - (exceso * percent);
.
.       if ( value < lowerLimit )
.       .       value = lowerLimit;
.
.       diff = diff - data[i,index,k] - value;
.       data[i,index,k] = value;
}
```

Fig. 8. Pseudocódigo del proceso de factibilización de soluciones del AG para el objetivo P2P

Luego, para las restricciones del objetivo del ISP, se recorren todas las aristas *interdomain* del problema (para el caso en cuestión, dados los datos de entrada, todas las aristas son enlaces interdomain) y se comprueba que el tráfico que circula por el enlace no sea mayor que la capacidad virtual del mismo. En caso que la restricción no se cumpla, se aplica un procedimiento similar al anterior, decrementando aleatoriamente valores de transferencia (t_{ij}^k) que pasan a través del enlace. El pseudocódigo del proceso de factibilización en base a las restricciones del ISP se muestra en la figura 9.

```
//ISP factibilization process for interdomain link e

double exceso = interdomain_traffic( e ) - v_e;
double diff = exceso;

while( diff > 0 ){
.       int k = rand( 0, cantContenidos-1 );  // get a randomic content
.       for each PID i {
.       .       for each PID j {
.       .       .       if ( i != j and hayCaminoInterdomain( e, i, j ) ) {
.       .       .       .       double percent = rand( 0, 1 );
.       .       .       .       double value = data[ i, j, k ] - (exceso * percent);
.       .       .       .
.       .       .       .       if ( value < lowerLimit )
.       .       .       .       .       value = lowerLimit;
.       .       .       .
.       .       .       .       diff = diff - data[ i, j, k ] - value;
.       .       .       .       data[ i, j, k, ] = value;
.       .       .       }
.       .       }
.       }
}
```

Fig. 9. Pseudocódigo del proceso de factibilización de soluciones del AG para el objetivo del ISP

G. Método NSGA-II y framework jMetal

Se utiliza el método NSGA-II [4] para la resolución del problema, ya que logra buenos resultados para algoritmos multiobjetivo basados en frente de Pareto y no utiliza población secundaria. El pseudocódigo del NSGA-II que se utiliza se presenta en la figura 10.

Para la implementación del algoritmo se utiliza el *framework* jMetal (versión 2.2), que posee una implementación en

```
Inicializar(P(0))
generacion = 0
Evaluar(P(0))
mientras (no CriterioParada) hacer
    R = Padres ∪ Hijos
    Frentes = Sorting No Dominado(R)
    NuevaPop = ∅
    i=1
    mientras |NuevaPop| + |Frentes(i)| ≤ sizepop
        Calcular Distancia de Crowding (Frentes(i))
        NuevaPop = NuevaPop ∪ Frentes(i)
        i++
    Sorting por Distancia (Frentes(i))
    NuevaPop = NuevaPop ∪ Frentes(i)[1:(sizepop - |NuevaPop|)
    Hijos = Seleccion y Reproduccion(NuevaPop)
    generacion ++
    P(generacion) = NuevaPop
retornar Mejor Solucion Hallada
```

Fig. 10. Pseudocodigo del NSGA-II [8]

java para el NSGA-II. El diagrama de clases del *framework* se muestra en la figura 11.

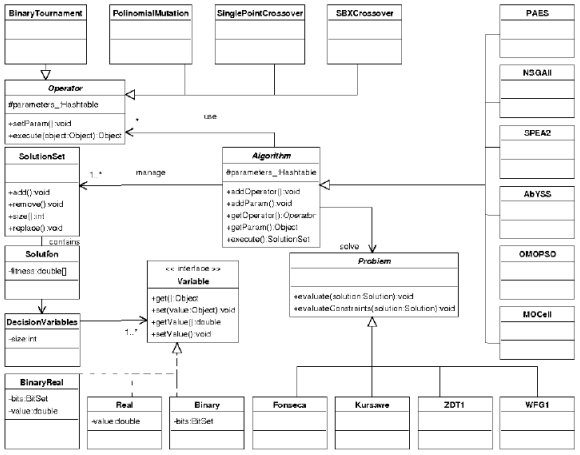

Fig. 11. Diagrama de clases del *framework* jMetal

Dicho *framework* fue utilizado como librería (.jar), extendiendo las clases necesarias para modelar el problema P4P. A continuación se listan las clases implementadas, indicando la superclase que extienden o la interfaz que implementan, y su objetivo para la resolución del problema. En la figura 12 se muestra el diagrama de clases parcial de esta implementación.

1) Packages: Se han utilizado los mismos paquetes *java* que utiliza jMetal para mantener la coherencia en la nomenclatura y ubicación de las clases. Por tanto, el proyecto *goalbitP4P* posee una estructura similar a la de jMetal, conteniendo solamente aquellos paquetes necesarios para colocar las clases que extienden una clase *jMetal* o implementan una interfaz *jMetal*. En este sentido se crean los paquetes *base* (conteniendo a su vez otros subpaquetes como *operator* y *variable*), *metaheuristics*, *problems* y *util*. También se ha agregado un paquete propio del proyecto para contener clases auxiliares requeridas por el problema (*data*). La estructura de paquetes se muestra en la figura 13.

2) Representación: Todo problema a resolver con un algoritmo genético tiene una o más variables de decisión, que determinan que tan adaptado es un individuo. En *jMetal*, las variables de decisión son modeladas mediante la clase *DecisionVariables*, que a su vez contiene un conjunto de instancias de *Variable*. Por como hemos modelado la representación de nuestro problema, se define la clase *GoalBitP4PVariable*, que extiende *Variable* y contiene la estructura con toda la información referente a P4P (los nodos, los contenidos, las capacidades de *upload* y *download*, las matrices de contenido, etc.). A su vez la clase *DecisionVariables* es extendida por

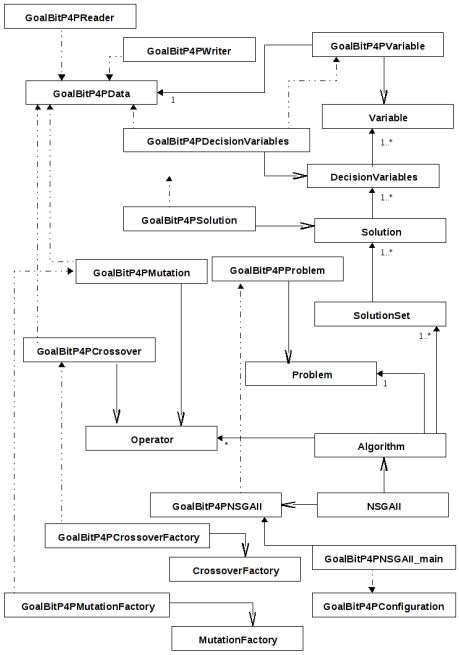

Fig. 12. Diagrama de clases del proyecto GoalBitP4P

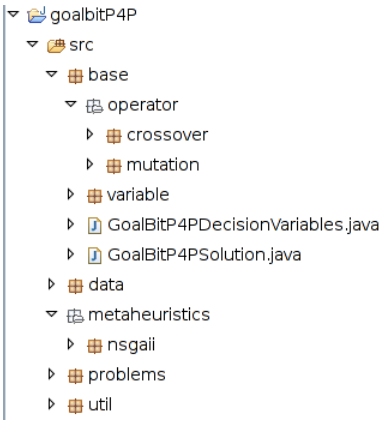

Fig. 13. Estructura de paquetes del proyeto *goalbitP4P*

GoalBitP4PDecisionVariables para contener una [2] instancia de tipo *GoalBitP4PVariable* en vez de *Variable*.

Cada individuo de una población es representado por la clase *Solution*, que ha sido extendida por *GoalBitP4PSolution* para contener el conjunto de variables de decisión *GoalBitP4PDecisionVariables* en lugar de instancias genéricas de *DecisionVariables*. Cada individuo contiene además de las variables, algunos datos necesarios para poder ser comparados con otros: valores para cada función objetivo, distancia de *crowding*, *ranking* del NSGA-II, etc. Estos datos son cargados durante la ejecución del algoritmo.

Las soluciones son agrupadas en una población, modelada mediante la clase *SolutionSet*.

3) Operadores: Los operadores utilizados en el algoritmo han sido descriptos en la sección V y son implementados dentro del paquete *operator*. Para el cruzamiento se define la clase *GoalBitP4PCrossover*, mientras que para la mutación se define la clase *GoalBitP4PMutation*, implementadas como se ha definido anteriormente y extendiendo la clase *Operator* de *jMetal*. Ambas clases redefinen el método abstracto *execute(Object)::Object*. Para el caso del cruzamiento el parámetro de entrada y de salida es una pareja de individuos (*GoalBitP4PSolution*), mientras que para la mutación tanto el parámetro de entrada como el de salida es un único individuo (*GoalBitP4PSolution*).

El operador de selección utilizado es el provisto por defecto por *jMetal* para el algoritmo NSGA-II (torneo), el cual se implementa en la clase *BinaryTournament2*. Dado que este operador sólo utiliza los atributos numéricos del individuo (y no su representación) para seleccionar los mejores individuos, no ha sido necesario extender esta clase.

En *jMetal* los operadores de mutación y cruzamiento son obtenidos a través de las *factories* correspondientes (*MutationFactory* y *CrossoverFactory* respectivamente). Ambas devuelven el operador según el nombre que se les pasa por parámetro. Esta correspondencia entre el nombre y el operador está *hardcoded* dentro de las *factories* por lo que se han extendido para contemplar los nuevos operadores del problema GoalBit + P4P (*GoalBitP4PMutationFactory* y *GoalBitP4PCrossoverFactory* respectivamente).

4) Datos de P4P: Las clases relacionadas con los datos del problema P4P se encuentran dentro del paquete *data*. Como ya fue mencionado en la subsección V-A, se ha reutilizado código para el manejo de los datos del problema. Para la carga de los datos desde archivo se utiliza la clase *GoalBitP4PReader*, la cual lee los datos y construye una instancia de *GoalBitP4PData*, conteniendo los nodos, aristas, capacidades de los enlaces, etc. Por su parte, la persistencia de datos hacia archivo se relaiza a través de la clase *GoalBitP4PWriter*.

Por último, se incluye una clase *GoalBitP4PConfiguration* donde se almacenan los datos de configuración para la ejecución del algoritmo. Dichos dato son cargados desde un archivo *.config* suministrado por la linea de comandos al ejecutar la aplicación.

[2]En nuestro caso cada instancia de *GoalBitP4PDecisionVariables* contiene una única instancia de *GoalBitP4PVariable*, puesto que nuestro problema cuenta con una sola variable de decisión.

5) Problema y NSGA-II: El *framework jMetal* contiene una amplia variedad de problemas implementados dentro del paquete *problems*. Simétricamente, dentro del proyecto *goalbitP4P*, se crea un paquete *problems* donde se define una subclase de la clase *Problem* de *jMetal*, denominada *GoalBitP4PProblem*. Esta subclase redefine los métodos *evaluate(Solution solution)* y *evaluateConstraints(Solution solution)*. El primero evalua las funciones objetivo para la solución recibida por parámetro, mientras que el segundo chequea las restricciones del problema y factibiliza la solución si ésta no cumple con alguna restricción.

El algoritmo NSGA-II se encuentra implementado en la clase *NSGAII* dentro del paquete *metaheuristicas.nsgaii*. Dicha clase define el método *SolutionSet execute()*, donde el algoritmo es ejecutado, retornando el conjunto de soluciones del problema. Esta clase ha sido copiada, renombrada y adaptada en *GoalBitP4PNSGAII* puesto que el conjunto de soluciones iniciales debe ser conformado por instancias de *GoalBitP4PSolution* y no de *Solution*. Al igual que la original, esta clase extiende de *Algorithm*.

6) Ejecutable: La clase *GoalBitP4PNSGAII_main* contiene el método *main* que ejecuta el algoritmo NSGA-II para el problema P4P. Esta clase es una copia y adaptación de la clase *NSGAII_main* de jMetal. El archivo con la confguración de parámetros a utilizar por el algoritmo puede ser pasado por consola. De no indicarse ningun archivo de configuración se utiliza un archivo con la configuración por defecto (configuration/GoalBitP4P.config). La figura 14 muestra el contenido de dicho archivo a modo de ejemplo.

```
#number of executions per file
number_executions       1

#operators parameters
crossover_probability   0.7
mutation_probability    0.05
population_size         50
algorithm_iterations    25000
crossover_alpha         0.5
mutation_mu             1.0
mutation_sigma          0.5

#directory where to place the solution files
output_path             output/
output_metrics          output/metrics

#file name where to save the pareto front
#output_pareto          pareto_front_prueba.pf

#file name where to read the pareto front
pareto_front            pareto/pareto_front.pf

#input files with P4P data
input                   input/example/input1.txt
input                   input/example/input4.txt
```

Fig. 14. Archivo de configuración por defecto del proyeto *goalbitP4P*

El proyecto ha sido empaquetado en el archivo ejecutable *GoalBitP4P_NSGAII.jar*. Para ejecutar el algoritmo se utiliza el siguiente comando en la consola (siendo opcional el archivo de configuración):

```
java jar GoalBitP4P_NSGAII.jar <config>
```

VI. Resultados

Una vez finalizada la etapa de calibración, se realizan varias ejecuciones del algoritmos con otros archivos de entrada

para ver el desempeño del algoritmo y obtener una solución aproximada al problema P4P.

Todas las ejecuciones han sido realizadas en la misma máquina en que se realizó la calibración de parámetros: Procesador Intel Core 2 Duo de 2.0 GHz y 3Gb de RAM; sistema operativo Debian Lenny 2.6.26-2-486.

Para esta etapa de ejecución del algoritmo se han seleccionado 10 archivos diferentes. Todos los archivos cuentan con un único enlace *interdomain* como fue mencionao anteriormente, ya que en este caso sólo interesa el enlace *interdomain* del ISP local.

Utilizando los valores de probabilidad de cruzamiento, probabilidad de mutación y cantidad de iteraciones del algoritmo calculados anteriormente, se realizan 3 ejecuciones del algoritmo por cada archivo de entrada, de forma de obtener valores promediales de las métricas. En este caso, las métricas analizadas son las mismas que en la etapa de calibración, incorporando además la cantidad de soluciones no dominadas que se obtienen en cada ejecución.

En la figura 15 se muestra una tabla con los valores promedio para cada uno de los 10 archivos procesados, y un promedio general para cada métrica, mientras que en la figura 16 se muestra gráficamente los mismos resultados. En cada gráfica se muestra una columna extra, coloreada en celeste, que representa el promedio general de todas las ejecuciones para cada métrica.

non_dominated	hypervolume	epsilon	spread	gen_distance	inv_gen_distance	time (seg)	time (min)
50	0,5500	2071,1167	0,8500	0,0300	0,0300	193	3
50	0,5752	1953,3424	0,8130	0,0264	0,0284	147	2
50	0,0466	4682,0400	0,8983	0,0703	0,0706	226	4
50	0,0000	8627,5323	0,9988	0,2110	0,1346	569	9
50	0,0000	6206,6700	0,9513	0,0982	0,0927	502	8
50	0,0000	13721,8441	1,0206	0,3806	0,2268	1002	17
50	0,0271	6038,4536	0,9826	0,1074	0,0893	476	8
50	0,0000	8071,6346	0,9604	0,1538	0,1240	641	11
50	0,0000	9212,0993	1,0152	0,1426	0,1446	648	11
50	0,0000	17506,1206	1,0151	0,5884	0,2977	1195	20
50	**0,1199**	**7809,0854**	**0,9505**	**0,1809**	**0,1239**	**560**	**9**

Fig. 15. Promedio de las métricas obtenidas en la ejecución del AG.

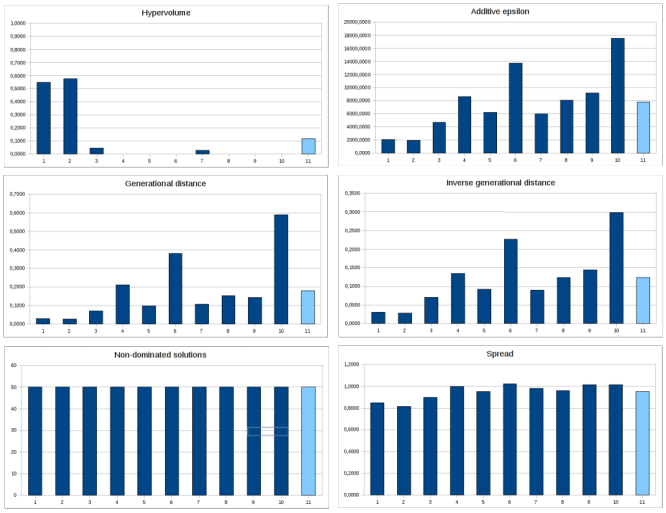

Fig. 16. Gráficas para las métricas obtenidas en la ejecución del AG.

Es importante destacar que todas las soluciones obtenidas en cada ejecución son no dominadas.

A. *Comparación con otros trabajos*

Se ha realizado un trabajo similar, utilizando otra metaheurística denominada *Simulated Annealing*, para resolver el problema P4P (no desde el punto de vista multiobjetivo). Lamentablemente, al momento de realizarse dicho trabajo, no se contaba con datos reales de entrada, utilizándose entonces datos generados aleatoriamente. Una vez recabados los datos reales con que se realiza el presente trabajo se comprueba que los datos de *Simulated Annealing* no eran del todo correctos, siendo inútil comparar los resultados con dicho trabajo.

Por otra parte, se cuenta con un análisis de P4P con los mismos datos de entrada de este trabajo, pero utilizando un *simplex* para la resolución del problema con un solo objetivo[3]. En este caso, los resultados presentados no son completos, por lo que resulta imposible la comparación. Sin embargo, dicho trabajo ha sido implementado en el emulador de GoalBit, comparando el desempeño y resultado de las ejecuciones con y sin P4P, logrando mejoras de un 40% en la reducción del tráfico en el enlace *interdomain* del ISP local, lo cual alienta a continuar con esta investigación.

Finalmente, un tercer estudio ha sido llevado a cabo utilizando un algoritmo de redireccionamiento de tráfico denominado *waterfilling*, implementado en Matlab[4]. Los resultados de dicho estudio se resumen en la figura 17.

archivo	Objetivo P2P	Objetivo ISP
input1	13100,98	1,00
input2	11447,62	1,00
input3	32047,55	0,99
input4	64409,21	0,99
input5	40511,78	1,00
input6	66582,57	1,00
input7	27625,3	1,00
input8	34692,32	1,00
input9	38804,02	1,00
input10	110544,22	1,00

Fig. 17. Resultado obtenido para las mismas entradas utilizando *waterfilling* en Matlab.

Se observa que los valores de transferencia obtenidos con dicha técnica superan los valores obtenidos con el algoritmo genético aquí presentado. Esto puede deberse a múltiples razones, como por ejemplo diferencias en las hipótesis del problema o los datos de entrada. En el presente trabajo se ha hecho énfasis en los valores de transferencia del PID_1 a través de su enlace *interdmain* con los demás PIDs, ignorando todos los restantes enlaces *interdomain* del grafo, dado que no se cuenta con datos reales para ellos. Sin embargo, en el trabajo de Matlab, todos los enlaces *interdomain* han sido tomados en cuenta para optimizar el tráfico. Por otra parte, si bien se presentan los valores de ambos objetivos (P2P e ISP), la optimización fue realizada de forma mono-objetivo, lo cual puede marcar también alguna diferencia en los resultados.

[3]Este trabajo ha sido realizado por un integrante del grupo GoalBit y no se cuenta con una referencia formal al mismo

[4]Este trabajo ha sido desarrollado por el Ing. Pablo Romero en el marco de su tesis de maestría, la cual no ha sido presentada aún, por lo que no se cuenta con una referencia formal a la documentación.

VII. Conclusiones y trabajo a futuro

En el presente trabajo se ha presentado el problema P4P, que busca optimizar el uso de la red para las aplicaciones P2P, cada día más utilizadas. También se presentó la aplicación P2P GoalBit, una adaptación de BitTorrent y VLC para transmisión de video en tiempo real, a la que se desea incorporar P4P para mejorar su desempeño.

Tomando el modelo matemático de P4P presentado, se definió un algoritmo genético multiobjetivo para la optimización del tráfico P2P que circula por enlaces *interdomain*. Es importante resaltar que el enfoque multiobjetivo es original para este problema, dado que hasta ahora se han realizado optimizaciones de un solo objetivo, reduciendo el modelo multiobjetivo a un modelo bi-nivel de un único objetivo. Este nuevo enfoque constituye un aporte para el prolema P4P, proporcionando además un frente de Pareto para el mismo.

Lamentablemente, por falta de tiempo, no ha sido posible incorporar el algoritmo genético a GoalBit para la selección de pares utilizando los pesos w presentados en la sección IV-C, pero resulta prometedor continuar con esta tarea dados los resultados obtenidos con otra heurística probada en el emulador de GoalBit (como fue mencionado en la sección VI-A), la cual ha logrado una reducción del 40% de las transferencias *interdomain*.

La incorporación del algoritmo genético a GolBit permitiría analizar el desempeño del algoritmo en un marco real, y conocer su aporte tanto para GoalBit como para los ISPs (en particular para el ISP local). Sin embargo, según los mismos resultados mencionados en el párrafo anterior, la mejora del 40% en la reducción del tráfico *interdomain* conllevó una degradación en la calidad persibida por los usuarios, debido a una alta tasa de rebuffering. Esto plantea un nuevo problema y desafío a ser investigado para conocer la viabilidad de P4P en aplicaciones P2P de tiempo real, como es el caso de GoalBit.

Por otra parte, se presentaron resultados de otra técnica diferente, denominada *waterfilling*, implementada en Matlab, la que ofrece resultados diferentes a los obtenidos con el algortmo implementado en este trabajo. Esto marca también una pauta para continuar investigando en el área y probando diferentes heurísticas de optimización, así como cuestionar la conveniencia o no de analizar el problema desde un punto de vista multi-objetivo o mono-objetivo.

Por último, cabe resaltar la dificultad de conseguir buenos datos de prueba para este problema, ya que los mismos son recabados por los ISPs y suelen ser muy reservados.

References

[1] Bittorrent web-site.

[2] Goalbit web-site.

[3] Mara Elisa Bertinat, Daniel De Vera, Daro Padula, Franco Robledo Amoza, Pablo Rodrguez-Bocca, Pablo Romero, and Gerardo Rubino. Goalbit: The first free and open source peer-to-peer streaming network. *Latin America Networking Conference*, September 2009. to be published.

[4] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm - nsga-ii. Technical Report 2000001, Kanpur Genetic Algorithms Laboratory (KanGAL), Indian Institute of Technology, Kanpur, India, 2000.

[5] C. M. Fonseca and P. J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. Technical report, Department of Automatic Control and Systems Engineering, The University of Sheffeld, Mappin Street, Sheffeld S1 3JD, U.K., April 1995.

[6] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. MA: Addison-Wesley, 1989. ISBN: 0201157675.

[7] Transport layer identication of P2P traffic. T. karagiannis and a. broido and m. faloutsos and k. claffy. In *Proceedings of ACM IMC 2004*, Taormina, Sicily, Italy, October 2004.

[8] S. Nesmachnow. Una versión paralela del algoritmo evolutivo para optimización multiobjetivo nsga-ii y su aplicacin al diseo de redes de comunicaciones conables. Technical Report TR0403, Centro de Calculo, Instituto de Computacion, Facultad de Ingeniera, Universidad de la Republica, Montevideo, Uruguay, April 1995.

[9] A. Norberg. Introduction to bittorrent. Technical Report TDBC85, Distributed systems C, Ume University, 2006.

[10] H. Xie, A. Krishnanmurthy, Y. R. Yang, and A. Silberschatz. P4p: Proactive provider participation for p2p. Technical Report YALEU/DCS/TR-1377, Department of Computer Science, Yale University, CT, USA, January 2007.