Metaheurísticas

Trabajo Práctico

2 de septiembre de 2016

Russo, Christian	679/10

Índice

1. Introducción Teórica

1.1. Colonia de Hormigas

Es una metaheuristica de la familia de PSO (Particle Swarm Optimization) basada en el comportamiento en grupo de las hormigas para definir el camino a un recurso deseado, en otras palabras es una metodología inspirada en el comportamiento colectivo de las hormigas en su búsqueda de alimentos. Es muy usada para solucionar problemas computacionales que pueden reducirse a buscar los mejores caminos o rutas en grafos. Es por eso que es muy importante recordar que las hormigas son prácticamente ciegas, y sin embargo, moviéndose al azar, acaban encontrando el camino más corto desde su nido hasta la fuente de alimentos (y regresar). Entre sus principales características se encuentran:

1. Una sola hormiga no es capaz de realizar todo el trabajo sino que termina siendo el resultado de muchas hormigas en conjunto.
2. Una hormiga, cuando se mueve, deja una señal química en el suelo, depositando una sustancia denominada **feromona**, para que las demás puedan seguirla.

De esta forma, aunque una hormiga aislada se mueva esencialmente al azar, las siguientes decidirán sus movimientos considerando seguir con mayor frecuencia el camino con mayor cantidad de feromonas.

La metaheurística general consiste en lo siguiente:

1. En principio, todas las hormigas se mueven de manera aleatoria, buscando por si solas un camino al recurso que estan buscando (una posible solución).
2. Una vez encontrada una solucion, la hormiga vuelve, dejando un rastro de feromonas; este rastro puede ser mayor o menor dependiendo de lo buena que sea la solución encontrada.
3. Utilizando este rastro de feromonas, las hormigas pueden compartir información entre sus distintos pares en la colonia.
4. Cuando una nueva hormiga inicia su trabajo, es influenciada por la feromona depositada por las hormigas anteriores, y así, aumenta las probabilidades de que esta siga los pasos de sus anteriores al acercarse a un recurso previamente encontrado.

En la **figura 1**, podemos ver una serie de iteraciones donde las hormigas llegan a la Fuente de comida y vuelven, dejando feromonas y en la siguiente iteración la solución se ve influenciada por la feromona.

Figura 1: Ejemplo convergencia a una solución

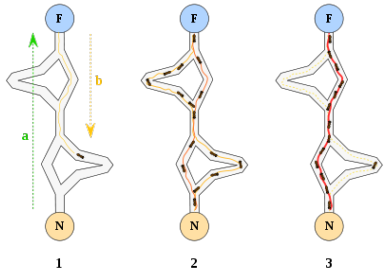
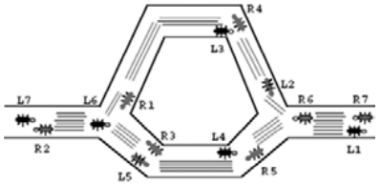


Figura 2: Ejemplo de uso de feromona



Finalmente se llega a una camino, el cual es elegido por casi todas las hormigas, siendo este la solución final.

En la **figura 2**, asumiendo que el número de lineas punteadas es proporcional a la cantidad de feromona, se puede ver como el camino inferior es más corto que el superior, por lo cual muchas más hormigas transitarán por éste durante el mismo período de tiempo. Esto implica, que en el camino más corto se acumula más feromona mucho más rápido. Después de cierto tiempo, la diferencia en la cantidad de feromona en los dos caminos es lo suficientemente grande para influenciar la decisión de las nuevas hormigas que entren a recorrer estas vías

Se puede ver que una gran ventaja de esta metaheurística es que puede construir una solución intercambiando información entre las distintas hormigas (soluciones), así generar una solución mejor de la que podrían generar individualmente.

Con el paso del tiempo, el rastro de feromonas comienza a evaporarse, y esto produce que los caminos pierdan su fuerza de atracción, cuanto más largo sea el camino, más tiempo demorara una hormiga en recorrerlo, más se evaporará la feromona y por ende serán menos frecuentado. Por su parte los caminos mas cortos (o mas óptimos) tendrán mayor cantidad de feromonas, por ende, mayor probabilidad de ser frecuentados.

ACO fue el primer algoritmo de optimizacion de Colonias de Hormigas desarrollado por Marco Dorigo en su tesis doctoral [3].

Algunas de las aplicaciones donde se utiliza esta metaheurística:

1. El problema del viajante de comercio (TSP)
2. Optimización para el diseño de circuitos lógicos combinatorios
3. Problemas de enrutamiento de vehículos
4. Problema de la asignación de horarios
5. Aplicaciones a análisis de ADN y a procesos de producción
6. Partición de un grafo en árboles:
7. Otros

Referencias

[1] https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms

[2] http://www-2.dc.uba.ar/materias/metah/meta2016-clase7.pdf

[3] http://people.idsia.ch/ gianni/Papers/CEC99.pdf

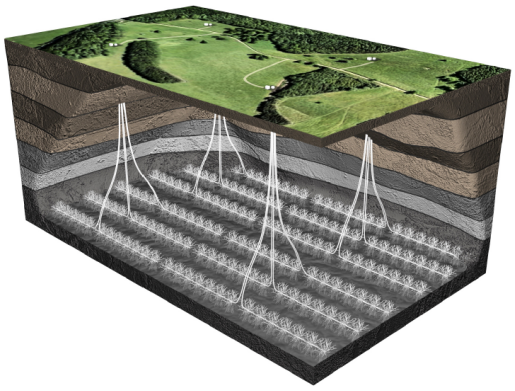
[4] Ant colony optimization: applications and trends. Carlos Algarin

2. El problema

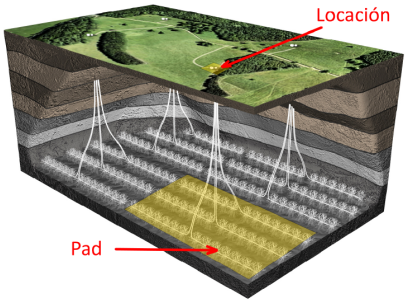
Un yacimiento petrolífero es una acumulación natural de hidrocarburos (gas natural y petróleo, entre otros) en el subsuelo. Debido a la creciente escasez de reservas de hidrocarburos acumulados en yacimientos convencionales, la industria del petróleo y diversos gobiernos nacionales han tornado su atención en las últimas décadas a la explotación de yacimientos no convencionales. Uno de los tipos de yacimientos más explorados está dado por las reservas de petróleo y gas natural almacenados en un tipo de rocas sedimentarias llamadas pelitas (shale), conocidos como yacimientos de *shale gas* y *shale oil*.

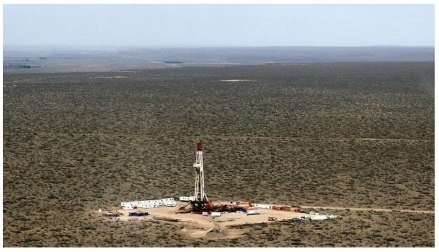
La explotación de este tipo de yacimientos utiliza métodos de fractura hidráulica, por medio de los cuales se generan fracturas en la roca madre para concentrar el petróleo y el gas natural y posteriormente proceder a su extracción. A pesar de que las primeras inyecciones de material para la extracción de hidrocarburos se remontan a la segunda mitad del siglo XIX, recién se comenzó a usar este tipo de métodos en forma extensiva a principios del siglo XXI, principalmente en Estados Unidos. Además de las reservas en Estados Unidos, en la última década se han descubierto enormes reservas de shale gas y shale oil en Argentina, Canadá y China.

Se describe el proceso de explotación de un yacimiento *shale*. En primer lugar, se realizan varias perforaciones verticales en el subsuelo que llegan hasta la roca madre. Como se ve a continuación:



El sector en la superficie alrededor de las bocas de pozo se denomina locación, y habitualmente ocupa un área rectangular de entre algunas decenas y unos pocos cientos de metros por lado. Estos equipos son los únicos que se ven en la superficie, y habitualmente su instalación involucra obras de nivelación del suelo y construcción de caminos de acceso. Como consecuencia, las locaciones no pueden estar sobre cursos de agua, barrancos o en sitios montanosos.





Locación en superficie para la explotación.

Cada perforación atraviesa la roca madre, y a lo largo de esta perforación se realizan los procesos de inyección de materiales para lograr la fractura de la roca. Luego, se utilizan las mismas para la extracción de los hidrocarburos que migran hacia las zonas de fractura.

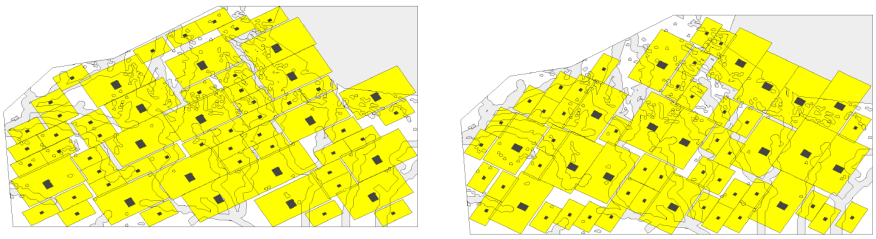
La zona explotada a partir de una locación se denomina pad, y tiene una forma típicamente rectangular.

Dadas estas características del problema queremos que las zonas de fractura en la roca madre no se deban superponer.

Al momento de planificar la explotación de un yacimiento no convencional, uno de los principales problemas a resolver es donde ubicar las locaciones y que tipo de explotación realizar en cada una (lo cual determina el tipo y tamano de los pads resultantes) con el objetivo de maximizar la producción y minimizar los costos y el impacto ambiental. Este problema se conoce con el nombre de optimización del área de drenaje, y como resultado se espera un plan de explotación que muestre las ubicaciones de locaciones y pads.

En la siguiente figura podemos ver el mapa de un yacimiento, y las configuraciones de pads que podemos usar para la explotación.

Los pads se deben ubicar siguiendo cierto ángulo α, llamado dirección de esfuerzo horizontal mínimo. Como por ejemplo:



Formalmente, los datos de entrada del problema están dados por los siguientes elementos:

1. El yacimiento $Y \subseteq \mathbb{R}^2$, que en este trabajo asumimos dado por un polígono en el plano (no necesariamente convexo). Todos los pads deben estar ubicados dentro del perímetro del yacimiento.

2. Una función $ogip : Y \to \mathbb{R}_+$ (original gas in place), que especifica la cantidad de shale gas esperada en cada punto del yacimiento, y el precio de venta $\rho \in \mathbb{R}_+$ de cada unidad extraída. Dado un pad $P \subseteq Y$ ubicado dentro del yacimiento, el gas total obtenido por explotar el pad esta dado por $ogip(P) := \int_P ogip(x)dx$

3. Un conjunto $S = \{S_1,...,S_k\}$ de configuraciones posibles de pads, que podemos utilizar para explotar el yacimiento. Para cada configuración $S \in S$, tenemos estos datos:

   a) Largo $lp_S \in \mathbb{R}_+$ y ancho $ap_S \in \mathbb{R}_+$ del pad, en metros.

   b) Largo $ll_S \in \mathbb{R}_+$ y ancho $al_S \in \mathbb{R}_+$ de la locación en metros, y asumimos que $ll_S < lp_S$ y $al_S < ap_S$.

   c) La locacion esta ubicada en el centro del pad, pero se puede mover algunos metros de este centro para evitar obstáculos geográficos. El parámetro de tolerancia $tol_S \in \mathbb{R}_+$ especifica la cantidad máxima de metros que el centro de la locación se puede mover con relación al centro del pad.

   d) Finalmente, se tiene el costo $c_S \in \mathbb{R}_+$ de construcción del pad. Dado un pad P correspondiente a la configuración S, definimos su margen neto como $neto(P) := \rho \, X \, ogip(P) - c_S$.

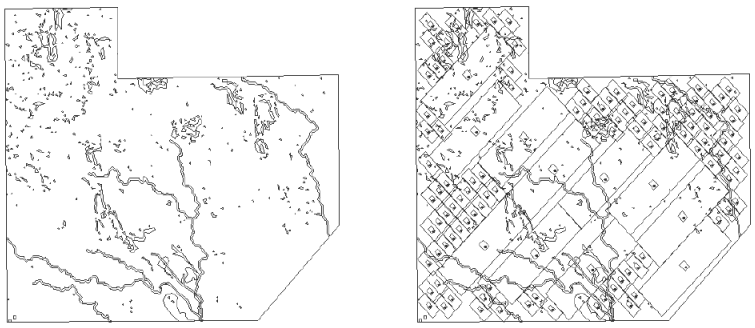
4. Un conjunto de obstáculos (habitualmente de índole geográfica) que las locaciones deben evitar. Consideramos que cada obstáculo esta dado por un polígono en el plano, y ninguna locación se puede superponer con ningún obstáculo.

5. Un ángulo $\alpha \in [0; 2\pi]$ de explotación ideal, denominado ángulo de esfuerzo horizontal miñimo, que especifica la orientación aproximada que deben tener los pozos horizontales sobre el yacimiento con relación al norte geográfico. Como esta orientación es aproximada, se tiene una tolerancia $\beta \in [0; 2\pi]$, que especifica que todos los pads deben estar orientados en un ángulo comprendido en el intervalo $[\alpha - \beta, \alpha + \beta]$.

El problema consiste en hallar un conjunto de pads Ps = $\{P_1, ..., P_n\}$ y un conjunto de locaciones Ls = $\{L_1, ..., L_n\}$ (de modo tal que la locación L_i corresponde al pad P_i, para i = 1,...,n) que maximice $neto(P) := \sum_{i=1}^{n} neto(P_i)$ de modo tal que se cumplan las siguientes restricciones:

1. Todos los pads deben estar contenidos dentro del yacimiento, es decir $P_i \subseteq Y$ para i = 1,...,n.
2. Como restricción, los pads de la solución no se deben superponer, dado que corresponden a zonas de fractura en la roca madre.
3. Cada pad y su locación deben responder a las especificaciones de una conguración de S. Es decir, para cada i = 1,...,n debe existir una configuración $S \in S$ tal que P_i tiene largo lp_S y ancho ap_S, L_i tiene largo ll_S y ancho al_S y su centro esta a no más de tol_S metros del centro de P_i, y finalmente P_i y L_i están orientados en un mismo ángulo, el cual debe estar entre $[\alpha - \beta, \alpha + \beta]$.
4. Ninguna locación de Ls se debe superponer con ningún obstáculo.

Por ejemplo, en la siguiente figura se muestra un yacimiento y los obstáculos dentro del yacimiento, y en la figura contigua se muestra una solución factible para $\alpha = \pi/4$ y para dos configuraciones posibles. Dado que la función ogip no siempre esta bien determinada de antemano (y en ocasiones se trabaja con estimaciones poco fiables de esta función) alternativamente se puede solicitar que se maximice el área total cubierta con los pads propuestos, en lugar del beneficio neto total obtenido. El algoritmo que se presenta en la próxima sección permite utilizar cualquiera de estas dos funciones objetivo, o una combinación lineal de ambas.



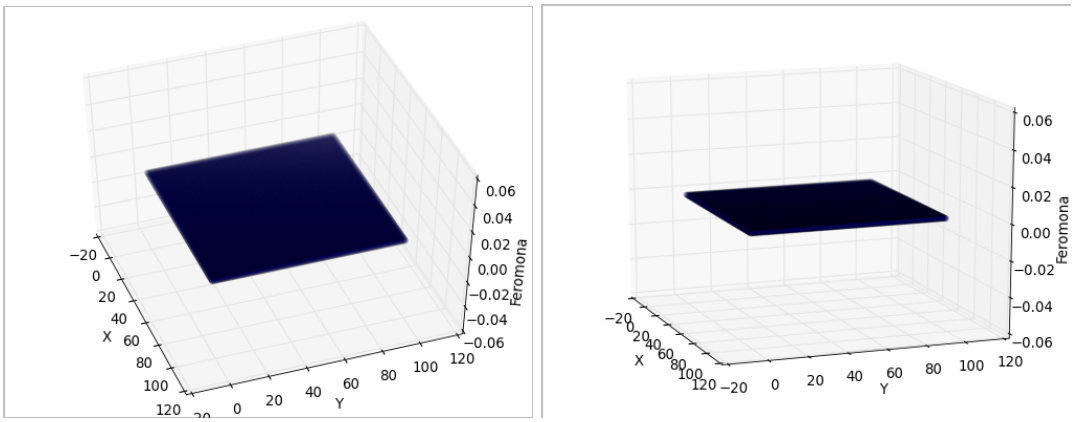
3. Algoritmo Propuesto

3.1. Explicacion

3.1.1. Colonia de hormigas

El algoritmo implementado esta basado en la metaheuristica de **colonia de hormigas**.

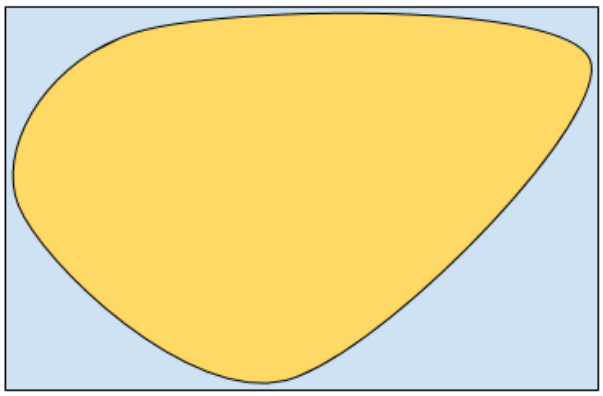
La idea general es tener una matriz que representa a la **feromona**, esta matriz reprensenta la region de entrada, en otras palabras el valor en la posicion (x,y) de la matriz de feromona representa el valor de la feromona en dicha posicion del suelo. En un principio se inicaliza con todos los valores en 0, como se ve a continuacion:



La matriz de fermona esta **discretizada** con un valor **configurable por parametro**. Por ejemplo, si la region es una region de 100x100 metros, pero la discretizacion de la feromona la seteamos en 10 metros, vamos a tener una matriz de feromona de 10x10. Es clave notar en este punto que cuanto menor es el valor de la discretizacion de la feromona mayor cantidad de puntos en la matriz y por lo tanto vamos a lograr mejores resultados pero a su vez resultados con tiempo computacional mas alto (ver seccion de experiemntacion).

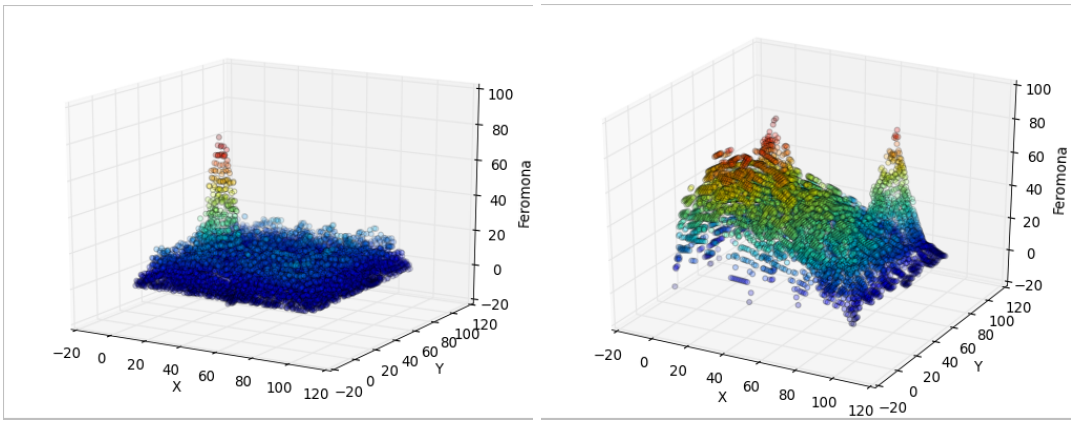
Por otro lado, contamos con una estructura auxiliar, una matriz de tamaños iguales que la matriz de feromona, esta estructura contiene 0 y 1 dependiendo de la **disponibilidad** del punto de la feromona. Esto es utilizado para saber cuando una feromona esta disponible, dado que los vamos a ir tapando con el correr de las iteraciones y por otro lado, en casos de que la region original no sea rectangular o este rotada, la matriz de feromona se arma de forma tal que la region quede incluida y los espacios fuera de la region se setean como **no disponibles** en esta nueva estructura. En la siguiente figura vemos un ejemplo donde la seccion amarrilla seria la region, y la azul seria la matriz de feromona, por lo tanto en la matriz de disponibilidad en los casilleros co-

rrespondientes a partes azules tendriamos un 1 indicando que no esta disponible esa feromona, ya que no esta dentro de la region.



Lo siguiente es, ejecutar una cantidad configurable de iteraciones el algoritmo obteniendo en cada solucion, un conjunto de soluciones que van **actualizando** la matriz de feromona y en cada paso del algortimo vamos **chequeando y guardandonos la mejor solucion**, siendo la mejor solucion la que tenga el valor mas alto del ogip tapado.

En las siguientes imagenes podemos ver algunos ejemplos de feromonas luego de varias iteraciones.



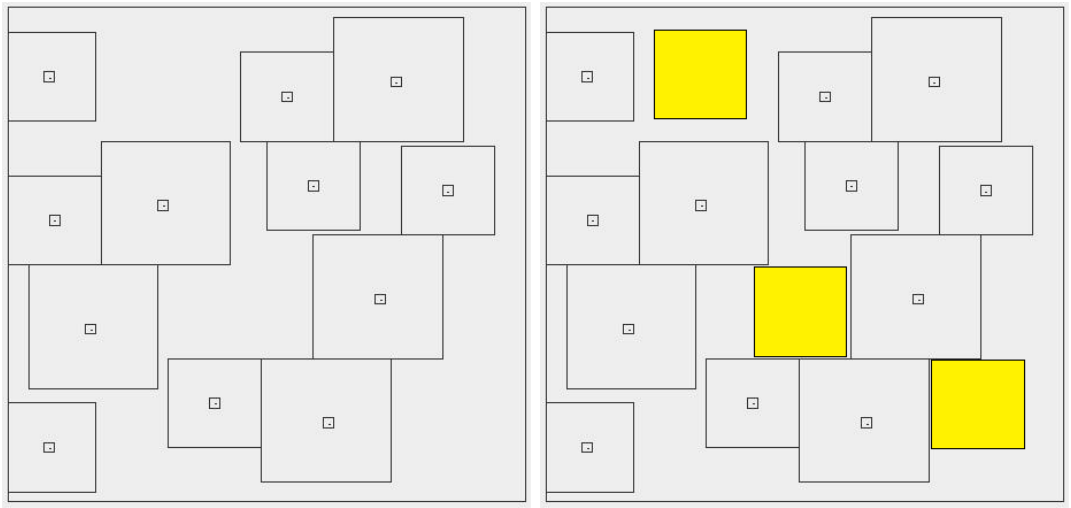
El algoritmo esta dividido en dos partes, la **iteracion inicial** y el **resto de las iteraciones** (decision tomado para facilitar la implementacion).

A continuacion se explica el trabajo de cada hormiga, es decir la forma de conseguir cada solucion dependiendo si es la iteracion inicial o el resto de las iteraciones.

Iteracion Inicial: En esta iteracion se crean una **cantidad configurable** de soluciones random, y para cada una de estas se actualiza la matriz de feromonas de la forma que corresponda.

Para crear una solucion random, la idea principal es meter pads centrados en puntos random de la region, teniendo en cuenta que sean validos (que esten dentro de la region, sin pisarse y sin interceptarce con una restrincion) hasta no poder meter mas pads. Notar que tambien se elije la semilla de forma random. El problema en este caso es decidir cuando ya no se pueder meter pads (dado que estamos trabajando en un plano **continuo**), por lo tanto se considero tener un **valor configurable** de intentos de meter pad fallidos. En caso de fallar en insertar el pad random esa cantidad de veces entonces se considera que no entra ningun pad mas y se retorna la solucion.

A continuacion podemos ver un ejemplo de una solucion random, donde termino de poner pads porque se **creyo** que no entraban mas, pero podemos ver en la otra figura, marcado con amarrillo 3 posibles pads que se nota que no se encontraron.



Resto de las iteraciones: En esta iteracion se crean una **cantidad configurable** de soluciones no random, y para cada una de estas se actualiza la matriz de feromonas de la forma que corresponda.

Para crear una **solucion no random**, la idea principal es agarrar el punto de la feromona mas **caliente** y generar una **cantidad configurable** de pads random que puedan tapar esa feromona.

Nota: si no consigo ningun pad random que tape dicho punto de la feromona (dado que los pads pueden ser invalidos, por ejemplo tocando una restriccion) , descarto ese punto de la feromona.

La manera de conseguir un pad random que tape un punto c, es tan simple como un pad posicionado en cualquier lugar de la region que tape a c. **Esto esta hecho de esta forma para que cada solucion sea distinta de las otras.**

Una vez tapado el punto de la feromona, se acomoda el pad, se tapan el resto de los puntos de la feromona que este pad tapo (tambien actualizamos la matriz de

disponibilidad) y se itera hasta no tener mas puntos feromona para tapar.

Actualizacion de la feromona: Para actualizar la feromona utilizamos una funcion llamada **esBuenaSolucion** que determina si una solucion es buena o mala. Contamos con un **parametro configurable** dado que se tiene 3 formas distintas de ver si una solucion es buena o mala

1. Opcion 0: Una solucion es buena si el ogip cubierto por esta solucion es mas que el 75 % del total del ogip, en otro caso es una mala solucion.
2. Opcion 1: Una solucion es buena si el ogip cubierto por esta solucion es mas que la mitad de la suma del maximo y minimo ogip hasta el momento, en otro caso es mala solucion.
3. Opcion 2: Una solucion es buena si el ogip cubierto por esta solucon es mas que el promedio de los ogip de las soluciones calculadas hasta el momento, en otro caso es mala solucion.

En caso de que la solucion sea buena se recorre la matriz de feromonas, aumentanto (calentando) en cada casillero (punto de la feromona) tapado por un pad en dicha solucion un valor igual al ogip en ese punto (normalizado) por una constante configurada por parametro. Es analogo para el caso de una solucion mala, solamente que disminuyendo (enfriando).

Importante: Tener en cuenta que para todos los casos, una vez elegido el pad que voy a agregar a mi solucion, a este pad lo **acomodo** haciendo que se mueva para la direccion mas cercana a otro pad o borde, hasta chocarce con el. De esta forma obtengo soluciones con pad pegados entre si y no aparecen huecos.

Una vez terminadas todas las iteraciones vamos a tener guardado la mejor solucion, el numero de iteracion de donde salio dicha solucion y el tiempo en conseguirla.

Notar que la mejor solucion no necesariamente es de la ultima iteracion, es por eso que la vamos guardando en cada iteracion y tambien vamos guardando el numero y tiempo de la iteracion de donde se origino la mejor solucion.

3.1.2. Colonia de hormigas Version Alternativa

Tambien se desarrollo un algoritmo extra, tambien basado en **colonias de hormigas**, implementado de forma casi identica al anterior, salvo que en lugar de tener una unica matriz de feromonas, tenemos una matriz de feromonas por cada semilla, es decir si tengo 5 semillas (tipo de pad) tengo 5 matrices de fermona y una unica matriz de disponibilidad.

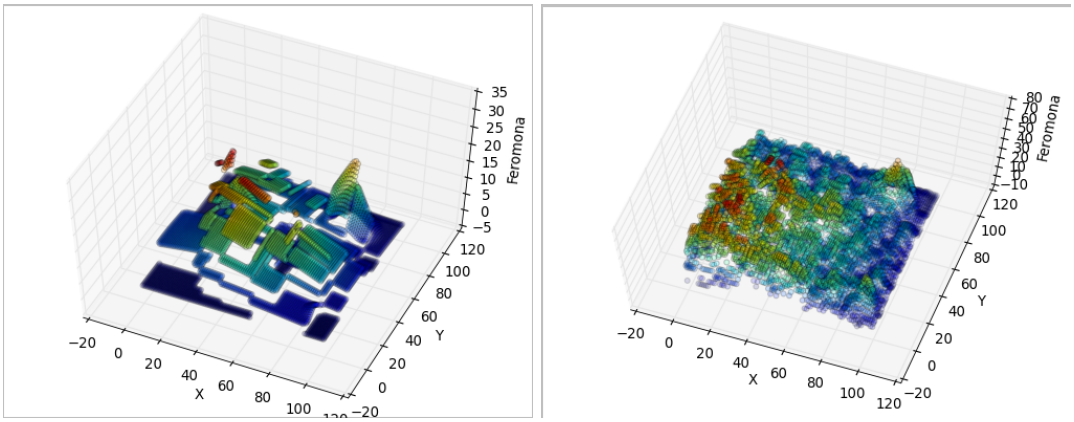
Entonces a la hora de actualizar la matriz de feromona, actualizamos para cada matriz los puntos tapados por los pads que tienen esa semilla. Por ejemplo, si tenemos

2 semillas, una grande y una chica, en una matriz de feromona vamos a actualizar los puntos tapados por los pads chicos y en la otra matriz de feromona actualizamos los puntos tapados por pads grandes.

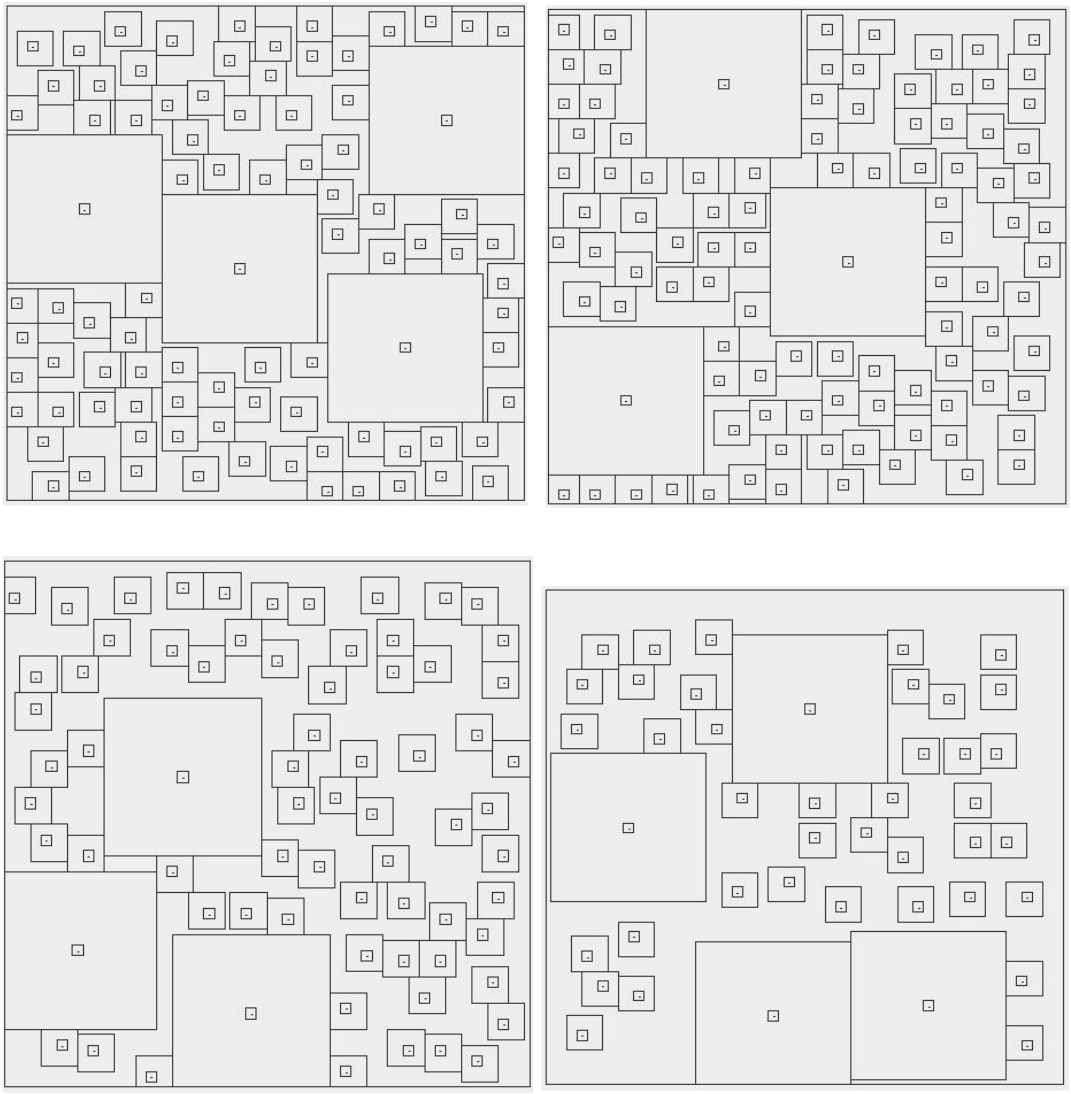
A la hora de buscar la feromona mas caliente, se busca entre todas las matrices de feromona.

El objetivo de esto es notar la variacion de la feromona, y tratar de ver que en algunos casos es conveniente poner los pads mas grandes en los puntos mas calientes mientras que los lugares restantes, por ejemplo a los costados de los lugares de alto valor, se agregan pads mas chicos.

A continuacion se muestran ejemplos de feromonas para esta version alternativa. Podemos ver en la figura de la izquierda como se modifico la feromona en los lugares donde tiene los pads mas grandes. Por otro lado, la feromona de la derecha se nota como esta modificada en los lugares que rodean a los pads mas grandes, concluyendo que se pusieron pads mas chicos en los alrededores de los mas grandes (ver seccion experiementacion)



A continuacion podemos ver algunos ejemplos de soluciones obtenidas sobre una misma instancia cambiando los parametros (ver mas en seccion experimentacion).



Notar que en los dos primeros casos no quedaron huecos mientras que en los ultimos dos si, esto se debe a la discretizacion de la feromona (en los ultimos dos casos la discretizacion es un numero mas alto, por lo tanto tenemos menos lugares que tapar, provocando huecos).

3.2. Pseudocodigo

3.2.1. Algoritmo principal

```
resolver() {
    inicializarFeromonas();
    ejecutarIteracionInicial();
    return ejecutarProximasSoluciones();
}
```

InicializarFeromonas() es una funcion que inicializa la feromona como una matriz del tamanio de la region, en caso que la region no se rectangular la inicializa con el rectangulo mas chico que contenga a la region.

Tambien inicializa una matriz disponibilidad del mismo tamanio que la feromona pero esta contiene 1 o 0 diciendonos si una feromona es validad o no, sea porque ya esta usada o porque la region es mas chica que la matriz de feromonas y en esa feromona no tenemos region.

```
ejecutarIteracionInicial() {
    soluciones = crearSolucionesRandom();
    for (Solucion solucion : soluciones) {
        if(esBuenaSolucion()){
            actualizarFeromona(solucion, OperacionFeromona.Calentar);
        } else {
            actualizarFeromona(solucion, OperacionFeromona.Enfriar);
        }
    }
}
```

ejecutarIteracionInicial() crea una cantidad seteada por parametro de soluciones randoms y para solucion chequea si es una buena o mala solucion, la funcion **esBuenaSolucion()** cambia segun un valor pasado por parametro, pero en definitiva, devuelve **true** si es una solucion considerada buena o **false** si es considerada mala.

En caso de que la solucion sea buena, **calentamos** la matriz de feromonas y **enfriamos** en caso contrario.

La funcion **actualizarFeromona()** simplemente recorre la matriz **calentando** o **enfriendo** cada valor respectivamente. Pero notar que la calienta **teniendo en cuenta el valor del ogip en ese punto**, es decir, se normaliza el ogip y en cada punto de feromona se calienta o enfria un valor igual a $escalar * feromonaNormalizadaEnElPunto$. Esto es para que el algoritmo de colonias de hormigas tenga en cuenta los valores originales del problema para generar sus soluciones.

La funcion **esBuenaSolucion()**, tiene 3 opciones que se cambian dependiendo de un valor pasado por parametro

1. Opcion 0: Una solucion es buena si el ogip cubierto por esta solucion es mas que el 75 % del total del ogip, en otro caso es una mala solucion.
2. Opcion 1: Una solucion es buena si el ogip cubierto por esta solucion es mas que la mitad de la suma del maximo y minimo ogip hasta el momento, en otro caso es mala solucion.
3. Opcion 2: Una solucion es buena si el ogip cubierto por esta solucon es mas que el promedio de los ogip de las soluciones calculadas hasta el momento, en otro caso es mala solucion.

```
crearSolucionesRandom() {
    ret = new Solucion()
    while (hasta que el area deje de cambiar) {
        Coordenada c = generarCoordenadaRandom()
        Pad pad = crearPadConSemillaRandamCentradaEnCoordenada(c)
        if (padValido(pad)){
            Pad padAcomodado = acomodarPad(pad);
            agregarPadASolucion(ret,padAcomodado);
        }
    }
    return ret;
}
```

La condicion del while corta cuando ya no se pueden meter mas pads en mi solucion random, esto se hace teniendo en cuenta una cantidad fijada por parametro de intentos de meter un pad, es decir, intento meter pads en la solucion y si la cantidad de veces que no pude meter es mayor al parametro seteado, se asume que no entran mas pads y sale del while.

Esto se hace para tratar de manejar la region en un plano **continuo** y para tratar de solucionar el problema de saber cuando ya no entran mas pads.

GenerarCoordenadaRandom() generada un x,y random dentro de la region.

crearPadConSemillaRandamCentradaEnCoordenada(c)() elije una semilla random y crea un pad centrado en c

padValido(pad) chequea si el pad no se pisa con ninguna restrincion, ni se va fuera de la region, ni se pisa con otro pad ya agregado a la solucion.

acomodarPad(pad) mueve el pad para una direccion random hasta chocarce son un borde u otro pad sin destapar el centro. Eso se hace para tratar de pegar todos los pads en la solucion.

agregarPadASolucion() agrega el pad a la solucion.

ejecutarProximasSoluciones() ejecuta una cantidad de veces igual a **cantIteraciones**, un algoritmo similar a **crearSolucionesRandom()**, llamado **generarSolucionesMaximaTemperatura()**. De esta forma se crean soluciones durante muchas iteraciones.

generarSolucionesMaximaTemperatura() es exactamente igual a **crearSolucionesRandom()** nada mas que cambiando el metodo **crearSolucionesRandom()** por **generarSolucionesMaximaFeromona()**. Esto significa que crea una cantidad configurable de soluciones de maxima feromona. Y para cada solucion chequea si es buena o mala actualizando la feromona como corresponda al igual que lo haciamos en la iteracion inicial.

Una solucion de maxima feromona es una solucion que tiene en cuenta el valor de la fermona para generarse y se genera con el siguiente algoritmo:

```
construirSolucionMaximaTemperatura() {
    sol = new Solucion();
    while (mientra que tenga feromonas disponibles) {
        Feromona p = getMaximaFeromona()
        if (es una feromona que se puede tapar) {
            for (int j = 0; j < getCantIntentosTaparFeromona(); j++) {
                nuevoPad = generarNuevoPad(p);
                if (esPadValido(nuevoPad)) {
                    nuevoPad = acomodarPad(nuevoPad);
                    agregarPadASolucion(nuevoPad);
                    break;
                }
            }
        }
    }
    return sol;
}
```

Mientras tenga feromonas disponible, es decir que todavia no las tapas (y estan dentro de la region) ejecuto todo el codigo dentro del while.

Obtengo la maxima feromona con **getMaximaFeromona()** y chequeo si es una posible feromona a tapar, dado que podria pasar que esa feromona este en una restrincion. Una vez que ya se que esa feromona la puedo tapar, trato de generar **getCantIntentosTaparFeromona()** pads (este valor es seteado por parametro).

La idea general es que para cada iteracion genero un pad random que tape a la feromona, chequeo si es valido, y si es valido lo agrego a la solucion y dejo de intentar tapar esta feromona.

Si no encontre ningun pad que sea valido y tape a la feromona en **getCantInten-**

tosTaparFeromona() intentos entonces ya esa feromona la descarto.

Notar que a los pads los acomodo (al igual que antes) para que queden pegados a otros pads o al borde.

Nota importante: Tanto en la generacion de soluciones random o las soluciones de maxima feromona, a la hora de fijarse si es una buena o mala solucion para actualizar la feromona, se chequea si es la **mejor solucion**, en caso de ser la mejor se **guarda**. Esto es para guardar la mejor solucion en el camino, podria llegar a pasar que la mejor solucion la encuentre en iteraciones iniciales y las siguientes sean peores.

3.2.2. Alternativa: Muchas Feromonas

A parte de tener un algoritmo de colonias de hormigas que ejecuta con una unica feromona, se programo una version alternativa donde contamos con mas de una feromona. Es decir para cada solucion, en lugar de tener una unica feromona, **tenemos una matriz de feromona para cada semilla**.

Para esto se modifica leventente el codigo teniendo en cuenta que ahora manejamos arrays de feromonas, y cambian levemente los algoritmos de actualizar la feromona. En particular, el algoritmo para obtener cual es la maxima feromona es el siguiente:

```
getMaximaFeromona() {
    result = null;
    for (f in todas las feromonas) {
        if(si la feromona f tiene valores dissponibles)
            if(maximo de f > result)
                result = f1;
    }
    return result;
}
```

El algoritmo es bastante sencillo, la idea es recorrer todas las feromonas buscando el maximo valor.

Tener en cuenta que a la hora de actualizar la feromona, cada feromona solo se actualiza donde corresponde. Por ejemplo, la feromona correspondiente a la semilla 0 se calienta o enfria solo en los puntos donde la solucion puso semillas 0.

4. Parametros

En esta seccion se explicaran los parametros utilizados a la hora de hacer la experimentacion de todos los algoritmos.

Notar que los algoritmos utilizados para comparar resultados son:

1. Scip (S)
2. Goloso (G)
3. Goloso Maximos Locales (GML)

todos estos seran descriptos con mas detalles en la seccion **experimentacion**

Los parametros usados para estos algoritmos son:

1. Nx: Valor de la discretizacion del eje X.
2. Ny: Valor de la discretizacion del eje Y.
3. PMD: **Paso Mejora Discretizacion**, en el algoritmo GML se utiliza para **rediscretizar**, leer la explicacion en la seccion **experimentacion**
4. PMP: **Paso Movimiento Pad**, en el algoritmo GML se utiliza para el paso en que se mueven los Pads, a la hora de buscar el maximo local, leer la explicacion en la seccion **experimentacion**

Por otro lado, para el algoritmo de colonia de hormgias se utilizaron los siguientes parametros:

1. IMPSR: **Intentos Meter Pad Solucion Random**, para ver cuando se termina de intentar meter pad en las soluciones tipo random (recordar que trabajamos en el continuo, y tenemos que decididr cuando ya creemos que se lleno la region).
2. CITF: **Cantidad Intentos de Tapar una Feromona**, cantidad de pads que tapan a una feromona, para cada uno pruebo si es valido. Si ninguno es, se descarta esa feromona.
3. CSRI: **Cantidad Soluciones Random Iniciales**, cantidad de soluciones de la primera iteracion. (Soluciones random)
4. CI: **Cantidad Iteraciones**, cantidad iteraciones luego de la inicial.
5. CSNRPI: **Cantidad Soluciones No Random Por Iteracion**, cantidad de soluciones por iteracion (Luego de la inicial).

6. MCBS: **Modo Chequeo Buena Solucion**, el modo para chequear cuando una solucion es buena o mala (para enfriar o calentar la feromona, explicado mejor en la seccion **Algoritmo Propuesto**).

7. DF: **Discretizacion Feromona**, la discretizacion de la matrix de feromonas.

8. FCF: **Factor Cambio Feromona**, factor que se multiplica al actualizar la feromona (tambien se lo multiplica por el ogip normalizado)

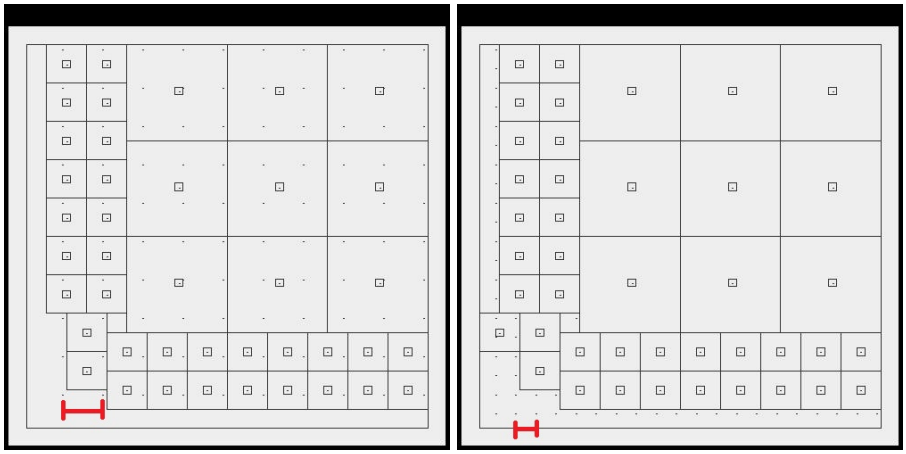
5. Experimentación

En esta seccion se presentaran los resultados de la experimentacion realizada. Pero previamente es necesario conocer los algoritmos con los cuales se comparo nuestra metaheuristca.

1. Scip: Este algoritmo esta realizado con programacion lineal entera y se explica en detalle en la seccion **Programacion Lineal Entera**.

2. Goloso: Este algoritmo es un simple Goloso, es decir, en cada iteracion agrega a la solucion en pad que mas ogip tape.

3. Goloso Maximos Locales: Este algoritmo, en cada iteracion busca el pad de mayor ogip (al igual que el goloso), pero como esto esta ligado a la discretizacion, a este pad se lo mueve tratando de ubicarlo en algun lugar cercano donde tenga mayor ogip, es decir, no importa al 100 % la discretizacion, dado que se consigue un pad centrado en la discretizacion (este pad es el pad con mayor ogip de todos los pads centrados en la discretizacion) y luego busco un maximo local en los alrededores y una vez encontrado lo agrega a la solucion. Para esto se utiliza un parametro **Paso Movimiento Pad** que indica el paso que se tiene en cuenta a la hora de mover el pad buscando el maximo local. Una vez que no tengo mas pads disponibles puede haber pasado que al mover los pads, no tenga mas pads disponibles de los centrados en la discretizacion, pero si existen huecos donde entran otros, por lo tanto, se **re-discretiza** el area no tapada hasta el momento, se hace una discretizacion mas fina, y para esto se usa el parametro **Paso mejora Discretizacion** que indica en cuanto se achica la discretizacion. Luego se vuelve a calcular los posibles pads para esta nueva discretizacion. Notar que solo se discretiza mas fino los sectores no tapados por los pads provenientes de la discretizacion mas gruesa. Por ejemplo, en las siguientes figuras podemos ver como en la primera el paso de la discretizacion es mas grueso que en la segunda (marcado en rojo). Y tambien podemos ver como la discretizacion en la segunda solo se hace en los luegares no tapados. Notar que la discretizacion esta marcada con puntos en la region.



Antes de continuar se explican que son cada item de los resultados obtenidos:

1. Tiempo:
2. Cant. Pads
3. Area (%)
4. Ogip
5. Area Superpuesta (del total cubierto) (%)
6. Cant. Ar. Cubierta
7. Cant. Ar. Superpuesta
8. Ar. Region
9. Iter. Sol.

Lo primero que se hizo fue un analisis de la variacion de los parametros. Entonces analizando los resultados en la seccion **anexo** se concluyo que los parametros CITF (Cantidad Intentos de Tapar una Feromona), FCF (Factor Cambio Feromona), IMPSR (Intentos Meter Pad Solucion Random) se van a dejar fijos. Por otro lado, se decidio hacer que CSRI = CSNRPI dadol que no importa que iteracion sea, siempre dajamos seteada la cantidad de soluciones en un mismo valor.

Entonces los parametros que nos quedaron para variar son CSRI (Cantidad Soluciones Random Iniciales = Cantidad Soluciones No Random Por Iteracion) , CI (Cantidad de Iteraciones), MCBS (Modo Chequeo Buena Solucion), DF (Discretizacion Feromona)

A la hora de analizar el tiempo vemos que el parametro MCBS no influye, por lo tanto no lo tenemos en cuenta para esta experiemtancion (Nota, se puede ver que el tiempo cambia levemente si se cambia el MCBC, pero nada significativo).

Lo primero que se analizo fue el tiempo de ejecucion si se varia la cantidad de soluciones por iteracion, y para eso observemos la siguiente tabla:

Corrida										Resultados
Algoritmo	Input	IMPSR	CITF	CSRI	CI	CSNRPI	MCBS	DF	FCF	Tiempo
CH	45G110X90Y8E7AR	10	4	5	10	5	0	6000	10	6,408
CH	45G110X90Y8E7AR	10	4	10	10	10	0	6000	10	14,518
CH	45G110X90Y8E7AR	10	4	5	20	5	0	6000	10	8,831
CH	45G110X90Y8E7AR	10	4	10	20	10	0	6000	10	10,59
CH	inst2	100	4	5	10	5	0	60	10	4,327
CH	inst2	100	4	10	10	10	0	60	10	7,51
CH	inst2	100	4	5	20	5	0	60	10	2,806
CH	inst2	100	4	10	20	10	0	60	10	4,56
CH	0G400x400_pocos	30	4	5	10	5	0	200	10	12,014
CH	0G400x400_pocos	30	4	10	10	10	0	200	10	18,793
CH	0G400x400_pocos	30	4	5	20	5	0	200	10	12,14
CH	0G400x400_pocos	30	4	10	20	10	0	200	10	15,667

En esta tabla podemos observar 6 subconjuntos de resultados agrupados entre si, donde para cada subconjunto, todos los parametros se mantienen igual excepto la cantidad de iteraciones que varia entre y 5 y 10 (parametros CSRI y CSNRPI). Se puede ver que en todos los casos el tiempo aumenta considerablemente al aumentar la cantidad de soluciones por iteracion, en particular, se puede ver en el primer caso (filas rojas) que al duplicar la cantidad de soluciones por iteracion, el tiempo aumenta a mas del doble (pasa de 6 a 14 segundos).

Observemos que pasa con el algoritmo de colonia de hormigas version 2:

Corrida										Resultados
Algoritmo	Input	IMPSR	CITF	CSRI	CI	CSNRPI	MCBS	DF	FCF	Tiempo
CH	45G110X90Y8E7AR	10	4	5	10	5	0	6000	10	6,408
CH	45G110X90Y8E7AR	10	4	10	10	10	0	6000	10	14,518
CH	45G110X90Y8E7AR	10	4	5	20	5	0	6000	10	8,831
CH	45G110X90Y8E7AR	10	4	10	20	10	0	6000	10	10,59
CHV2	inst2	100	4	5	10	5	0	60	10	8,066
CHV2	inst2	100	4	10	10	10	0	60	10	11,151
CHV2	inst2	100	4	5	20	5	0	60	10	10,562
CHV2	inst2	100	4	10	20	10	0	60	10	11,536
CHV2	45G100x100_muchos	100	4	5	10	5	0	60	10	9,819
CHV2	45G100x100_muchos	100	4	10	10	10	0	60	10	12,348
CHV2	45G100x100_muchos	100	4	5	20	5	0	60	10	7,37
CHV2	45G100x100_muchos	100	4	10	20	10	0	60	10	12,13

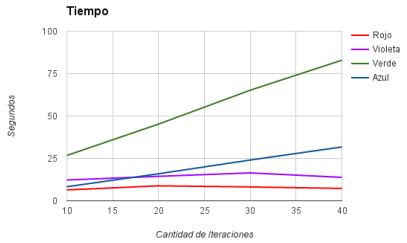
En esta tabla podemos observar lo mismo que en el caso anterior, es decir, al aumentar la cantidad de soluciones el tiempo aumenta. Pero vale destacar que en instancias como **inst2**, que es una instancia pequenia, el tiempo aumenta poco.

Todos estos resultados tienen sentido, dado que, sin importar la version que se utilice del algoritmo, para cada solucion encontrada tenemos que aumentar la feromona, y la logica de actualizacion de feromona demora bastante, mas alla del tiempo que demora en conseguir la propia solucion.

A continuacion vamos a analizar que pasa con el tiempo al variar el otro parametro, que es el CI (Cantidad de Iteraciones).

Corrida										Resultados	
Algoritmo	Input	IMPSR	CITF	CSRI	CI	CSNRPI	MCBS	DF	FCF	Tiempo	Iter. Sol.
CH	45G110X90Y8E7AR	10	4	5	10	5	0	6000	10	6,408	9
CH	45G110X90Y8E7AR	10	4	5	20	5	0	6000	10	8,831	14
CH	45G110X90Y8E7AR	10	4	5	30	5	0	6000	10	8,163	16
CH	45G110X90Y8E7AR	10	4	5	40	5	0	6000	10	7,298	11
CH	45G110X90Y8E7AR	10	4	10	10	10	1	5000	10	12,27	10
CH	45G110X90Y8E7AR	10	4	10	20	10	1	5000	10	14,43	16
CH	45G110X90Y8E7AR	10	4	10	30	10	1	5000	10	16,455	25
CH	45G110X90Y8E7AR	10	4	10	40	10	1	5000	10	13,829	12
CH	45G110X90Y8E7AR	10	4	5	10	5	0	1000	10	26,733	10
CH	45G110X90Y8E7AR	10	4	5	20	5	0	1000	10	45,216	20
CH	45G110X90Y8E7AR	10	4	5	30	5	0	1000	10	65,206	30
CH	45G110X90Y8E7AR	10	4	5	40	5	0	1000	10	82,893	40
CH	45G110X90Y8E7AR	10	4	5	10	5	1	500	10	83,028	10
CH	45G110X90Y8E7AR	10	4	5	20	5	1	500	10	159,376	20
CH	45G110X90Y8E7AR	10	4	5	30	5	1	500	10	240,406	30
CH	45G110X90Y8E7AR	10	4	5	40	5	1	500	10	317,481	39

En esta tabla se pueden ver 4 subconjunto de corridas (rojo, violeta, verde y azul). Veamos a continuacion, como se refleja esto en una grafico de lineas.



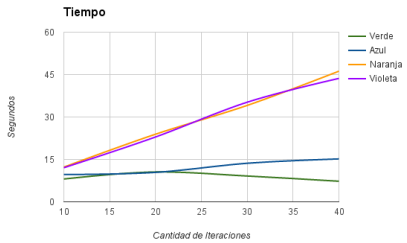
Podemos ver tanto en la tabla como en el grafico, que a medida que aumentamos la cantidad de iteraciones, aumenta el tiempo, excepto en el caso del rojo y violeta. Esto es a causa del que el tiempo, es el tiempo hasta encontrar la mejor solucion y si miramos con detalle la columna Iter. Sol., vamos a observar que en los dos casos que el tiempo disminuye es porque la mejor solucion se encontro antes que los casos anteriores. Este resultado es un resultado correcto, dado que puede pasar que la solucion se encuentre antes ya que cada solucion es distinta de las otras y ademas se comienza con soluciones randoms. Por lo tanto vemos que no siempre aumentar la canitad de iteraciones es bueno para mejorar el tiempo ya que encontramos casos donde aumentamos las iteraciones y se tardo mas, pero por otro lado encontramos casos donde aumentamos las iteraciones y se tardo menos y vale aclarar que existen casos donde se aumento la cantidad de iteraciones y la solucion se encontro en un numero de iteracion que era posible encontrar en casos anteriores.

Nota: en el grafico los valores muy elevados se normalizaron (se lo dividio por 10) para que el grafico quede prolijo.

Veamos ahora que pasa usando colonia de hormigas version 2:

Corrida										Resultados	
Algoritmo	Input	IMPSR	CITF	CSRI	CI	CSNRPI	MCBS	DF	FCF	Tiempo	Iter. Sol.
CHV2	inst2	100	4	5	10	5	0	60	10	8,066	10
CHV2	inst2	100	4	5	20	5	0	60	10	10,562	12
CHV2	inst2	100	4	5	30	5	0	60	10	9,124	11
CHV2	inst2	100	4	5	40	5	0	60	10	7,279	8
CHV2	inst2	100	4	5	10	5	0	30	10	9,648	10
CHV2	inst2	100	4	5	20	5	0	30	10	10,409	11
CHV2	inst2	100	4	5	30	5	0	30	10	13,633	15
CHV2	inst2	100	4	5	40	5	0	30	10	15,176	21
CHV2	inst2	100	4	10	10	10	0	1	10	1227,867	8
CHV2	inst2	100	4	10	20	10	0	1	10	2391,535	20
CHV2	inst2	100	4	10	30	10	0	1	10	3409,414	30
CHV2	inst2	100	4	10	40	10	0	1	10	4620,145	40
CHV2	inst2	100	4	10	10	10	2	1	10	1205,3	10
CHV2	inst2	100	4	10	20	10	2	1	10	2293,054	20
CHV2	inst2	100	4	10	30	10	2	1	10	3523,47	30
CHV2	inst2	100	4	10	40	10	2	1	10	4365,503	40

En esta tabla se pueden ver 4 subconjunto de corridas (verde, azul, naranja y violeta). Veamos a continuacion, como se refleja esto en una grafico de lineas.



Se puede ver, al igual que en la version 1, que la version 2 se comparta igual, es decir al aumentar la cantidad de iteraciones aumenta el tiempo, pero en algunos casos el tiempo disminiye debido a que la solucion se encontro en una interacion mucho antes.

Agregar comparacion de tiempos para las distintas discretizaciones, es decir, comprar los tiempos de todas las discretizaciones.

Agregar comparacion de tiempos entre los algortimos, agarrar para cada instancia la mejor soluicion (ogip) y ponerlo todo en una tabla y graficar. Aca se puede explicar mucho. Aca se puede charlar que quiza tenemos una solucion no tan buiena pero que tarda mucho mucho menos.

6. Resolucion alternativa

La idea es usar un GRASP (Goloso Randomizado) y en general es encontrar el punto de la discretizacion donde el ogip valga mas y luego, dentro de un rango valido (un cuadrado) agarrar una lista de posibles pads, es decir, agregar a la lista todos los pads que pueden tapar a ese punto y luego agarrar un pad random para meter en la solucion y asi iterar sucesivamente.

Una vez conseguida la solucion, hacer una busqueda local para tratar de mejorar la solucion. Y la forma de hacer la busqueda local es.

1- Para cada pad tratar de acercarlo a otro pads y al finalizar esto chequear si es posible meter otro nuevo pad. 2- Sacar algun pad con algun criterio y tratar de acomodar los pad restantes con algun algoritmo de programacion lineal entera. 3 - Otro

7. Programacion lineal entera

En este trabajo se optó por reducir el problema a un problema de conjunto independiente de peso máximo en un grafo dado por la discretización del área del yacimiento, que proporcionó buenos resultados en la práctica. Dado un grafo $G = (V, E)$, un **conjunto independiente** es un conjunto $I \subseteq V$ de vértices tal que $ij \notin E$ para todo $i, j \in I$. Si además tenemos una función de peso $W : V \to \mathbb{R}$, el **peso** del conjunto independiente I es $w(I) := \sum_{i \in I} w_i$. La motivación para este enfoque viene dada por el hecho de que el conjunto de pads de la solución conforma un conjunto de elementos no conflictivos entre sí, situación que es modelada adecuadamente por medio de conjuntos independientes en un grafo. Sin embargo, esta reducción trae aparejado un *costo de discretización*, que será mayor cuanto mayor sea el paso de discretización seleccionado.

A grandes rasgos, el algoritmo propuesto está compuesto por los siguientes puntos:

1. Discretización $D \subseteq Y$ del área geográfica del yacimiento.
2. Generación de un conjunto T de pads posibles sobre la base de la discretización D.
3. Planteo de un grafo $G = (T, E)$, de modo tal que cada conjunto independiente de G corresponde a una solución factible del problema de optimización del área de drenaje. Los vértices del grafo reciben pesos adecuadamente definidos, de modo tal que el peso de cada conjunto independiente corresponde a la función objetivo de la solución factible.
4. Búsqueda de un conjunto independiente de peso máximo sobre G por medio de un modelo de programación lineal entera, para obtener una solución P al problema.

Describimos a continuación cada punto del algoritmo. Para esto, sean Δ_x, $\Delta_y \in \mathbb{R}_+$ los **pasos de discretización** y sea $A = \{\alpha_1, \ldots, \alpha_p\}$ un conjunto de ángulos posibles, de modo tal que $\alpha_i \in [\alpha - \beta, \alpha + \beta]$ para $i = 1, \ldots, p$. En nuestra implementación computacional, tomamos $A = \{\alpha - \beta, \alpha, \alpha + \beta\}$.

Discretización. El primer paso del algoritmo consiste en generar una discretización $D = \{(x_i, y_i)\}_{i=1}^{m}$ por filas y columnas del área del yacimiento, de modo tal que dos puntos consecutivos de una misma fila estén a distancia Δ_x y dos puntos consecutivos de una misma columna estén a distancia Δ_y. Para esto, se genera un reticulado de puntos en el plano con ángulo α.

Generación de pads. Para cada punto $(x, y) \in D$ de la discretización, cada configuración $S \in \S$ y cada ángulo $i \in \{1, \ldots, p\}$, se incluye en el conjunto T un pad P con configuración S, centrado en (x, y) y rotado en ángulo α_i, siempre que el pad P (i) esté incluido completamente dentro de Y y (ii) su locación L no interseque con ningún obstáculo. Para determinar este último punto, se consideran como centros posibles de

la locación el punto (x, y) y ocho puntos equiangulados sobre la circunferencia de centro (x, y) y radio tol_S, y se considera que se cumple la condición (ii) si para alguno de estos puntos, la locación centrada en ese punto no interseca a ningún obstáculo. Este enfoque es arbitrario e incurre en un nuevo error de discretización, pero se observó que genera resultados aceptables en la práctica.

Grafo de conflictos. Se genera un grafo $G = (T, E)$ cuyos vértices están dados por todos los pads generados en el punto anterior, y cuyas aristas unen pares de pads con intersección no vacía. El conjunto E está compuesto por los pares (P_1, P_2) tales que existe algún punto $(x, y) \in D$ con $(x, y) \in P_1$ y $(x, y) \in P_2$. Esta definición de E permite que existan pares de pads con pequeñas superposiciones pero sin una arista que los una en G. Esto sucede cuando la intersección no contiene ningún punto de la discretización D, lo cual puede ocurrir sólo cuando la superposición es pequeña. De este modo, se maneja adecuadamente la restricción elástica de no superposición de pads.

Obtención de una solución. Se plantea y se resuelve la siguiente formulación de conjunto independiente con peso máximo sobre G, usando las restricciones clique sobre todos los puntos de la discretización. En este modelo, se tiene una variable binaria x_P por cada pad, de modo tal que $x_P = 1$ si y sólo si el pad P se incluye en la solución.

$$\begin{aligned} \text{máx} \quad & \sum_{P \in \mathcal{T}} g(P) x_P \\ \sum_{P:(x,y)\in\mathcal{T}} x_P \quad & \leq \quad 1 \qquad \forall\ (x, y) \in D \\ x_P \quad & \in \quad \{0, 1\} \qquad \forall\ P \in \mathcal{T} \end{aligned}$$

El coeficiente $g(P)$ asociado con la variable x_P en la función objetivo es $g(P) = neto(P)$ si se optimiza el beneficio total (dado por la venta de la producción total esperada menos los costos de construcción), o bien $g(P) = area(P)$ si se optimiza el área total cubierta. Dado que los puntos de la discretización D generan todas las cliques maximales de G (aunque no todo punto de D genera necesariamente una clique maximal), esta formulación incluye todas las restricciones de la formulación por cliques del problema de conjunto independiente de peso máximo, y se espera que sea más fuerte que una formulación con una restricción por arista. Dadas las características aproximadas del procedimiento, no resulta imprescindible en la práctica resolver en forma óptima el modelo de programación entera planteado, aunque la próxima sección muestra que en general este modelo se resuelve en forma exacta para tamaños de instancia razonables.

La generación de la discretización D es un paso clave dentro del algoritmo. Si los pasos de discretización Δ_x y Δ_y son demasiado grandes, entonces no se generará un número suficientemente grande y variado de pads en T y la solución será de peor cali-

dad, además de incluir potencialmente superposiciones entre los pads seleccionados, dado el modo en el que se generan las aristas de G. Sin embargo, a medida que Δ_x y Δ_y disminuyen se espera que estos efectos se vean minimizados, y que caigan por debajo de los errores de mediciones y de los parámetros de seguridad habituales en la industria hidrocarburífera. A medida que tienden a cero, la solución generada por este procedimiento tiende a la solución óptima. Los experimentos computacionales presentados en la próxima sección muestran que eligiendo adecuadamente los valores de Δ_x y Δ_y se obtienen buenos resultados.

8. Conclusión

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.