Metaheursticas

Trabajo Práctico

2 de septiembre de 2016

Russo, Christian	679/10

ndice

1. Introduccin Terica

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 metodologa inspirada en el comportamiento colectivo de las hormigas en su bsqueda 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 prcticamente ciegas, y sin embargo, movindose al azar, acaban encontrando el camino ms corto desde su nido hasta la fuente de alimentos (y regresar). Entre sus principales caractersticas 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 seal qumica en el suelo, depositando una sustancia denominada **feromona**, para que las dems puedan seguirla.

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

La metaheurstica 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 solucin).
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 solucin encontrada.
3. Utilizando este rastro de feromonas, las hormigas pueden compartir informacin 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 as, 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 iteracin la solucin se ve influenciada por la feromona.

Figura 1: Ejemplo convergencia a una solucin

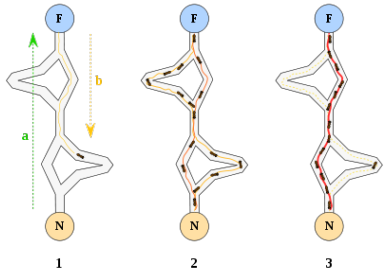
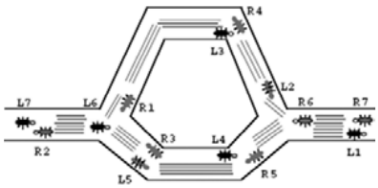


Figura 2: Ejemplo de uso de feromona



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 nmero de lineas punteadas es proporcional a la cantidad de feromona, se puede ver como el camino inferior es ms corto que el superior, por lo cual muchas ms hormigas transitarn por ste durante el mismo período de tiempo. Esto implica, que en el camino ms corto se acumula ms feromona mucho ms rpido. Despus de cierto tiempo, la diferencia en la cantidad de feromona en los dos caminos es lo suficientemente grande para influenciar la decisin de las nuevas hormigas que entren a recorrer estas vas

Se puede ver que una gran ventaja de esta metaheurstica es que puede construir una solucin intercambiando informacin entre las distintas hormigas (soluciones), as generar una solucin mejor de la que podran generar individualmente.

Con el paso del tiempo, el rastro de feromonas comienza a evaporarse, y esto produce que los caminos pierdan su fuerza de atraccin, cuanto ms largo sea el camino, ms tiempo demorara una hormiga en recorrerlo, ms se evaporar la feromona y por ende sern menos frecuentado. Por su parte los caminos mas cortos (o mas ptimos) tendrn 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 metaheurstica:

1. El problema del viajante de comercio (TSP)

2. Optimizacin para el diseo de circuitos lgicos combinatorios
3. Problemas de enrutamiento de vehculos
4. Problema de la asignacin de horarios
5. Aplicaciones a anlisis de ADN y a procesos de produccin
6. Particin de un grafo en rboles:
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

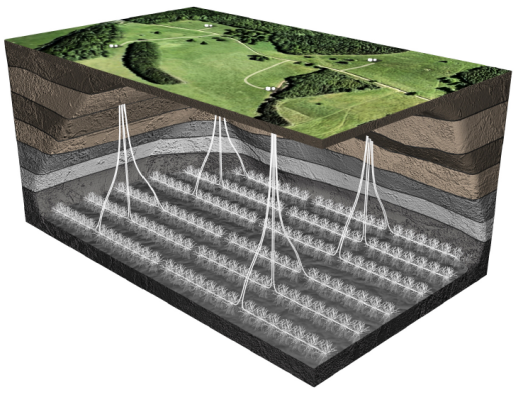
[4] Ant colony optimization: applications and trends. Carlos Algarin

2. El problema

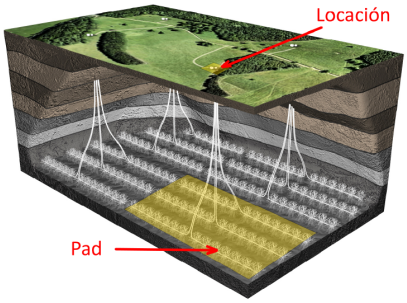
Un yacimiento petrolífero es una acumulación natural de hidrocarburos (gas natural y petróleo, entre otros) en el subsuelo. Debido a la creciente escasez de reservas de hidrocarburos acumulados en yacimientos convencionales, la industria del petróleo y diversos gobiernos nacionales han tornado su atención 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.

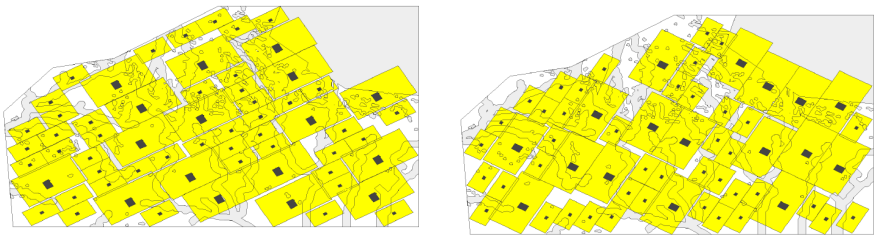
La zona explotada a partir de una locación se denomina pad, y tiene una forma típicamente rectangular.

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 $neto(P) := \rho \, X \, ogip(P) - c_S$.

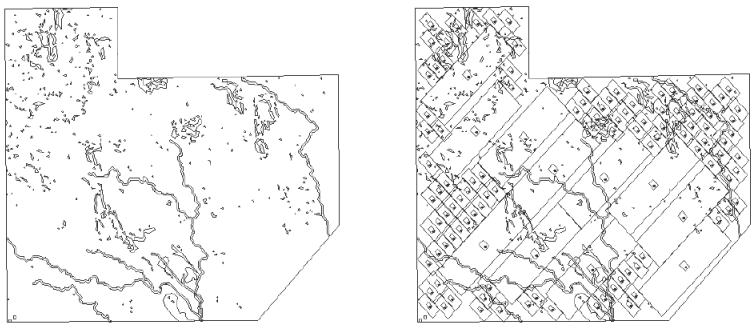
4. Un conjunto de obstáculos (habitualmente de índole geográfica) que las locaciones deben evitar. Consideramos que cada obstáculo esta dado por un polígono en el plano, y ninguna locación se puede superponer con ningún obstáculo.

5. Un ángulo $\alpha \in [0; 2\pi]$ de explotación ideal, denominado ángulo de esfuerzo horizontal mińimo, 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 congura-ción 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, que en un principio esta con todos los valores en 0.

(Ejemplo feromona todo en cero)

La matriz de fermona esta discretizada con un valor configurable por parametro. Por ejemplo, si la region es una region de 100x100, pero la discretizacion de la feromona la seteamos en 10, 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 tamaos 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 un punto 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, 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 correspondientes a partes azules tendriamos un 1 indicando que no esta disponible esa feromona, ya que no esta dentro de la region.

Figura 3: Ejemplo matriz de feromona y matriz de disponibilidad

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.

(2 dibujos de feromonas actualizadas)

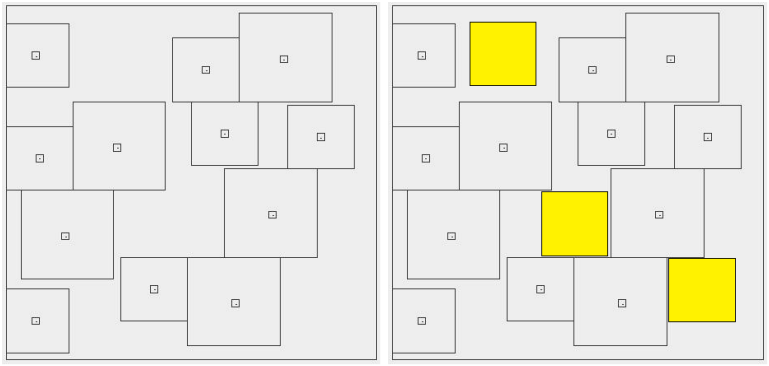
El algoritmo esta dividido en dos partes, la iteracion inicial y el resto de las iteraciones (decision tomado para facilitar la implementacion). A continuacion se explica el trabajo de cada hormiga, es decir la forma de conseguir cada solucion dependiendo si es la iteracion inicial o el resto de las 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 çreyo"que no entraban mas, pero podemos ver en la otra figura, marcado con amarrillo 3 posibles pads que se nota que no se encontraron.

Figura 4: Ejemplo solucion random obtenida



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 .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.

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.

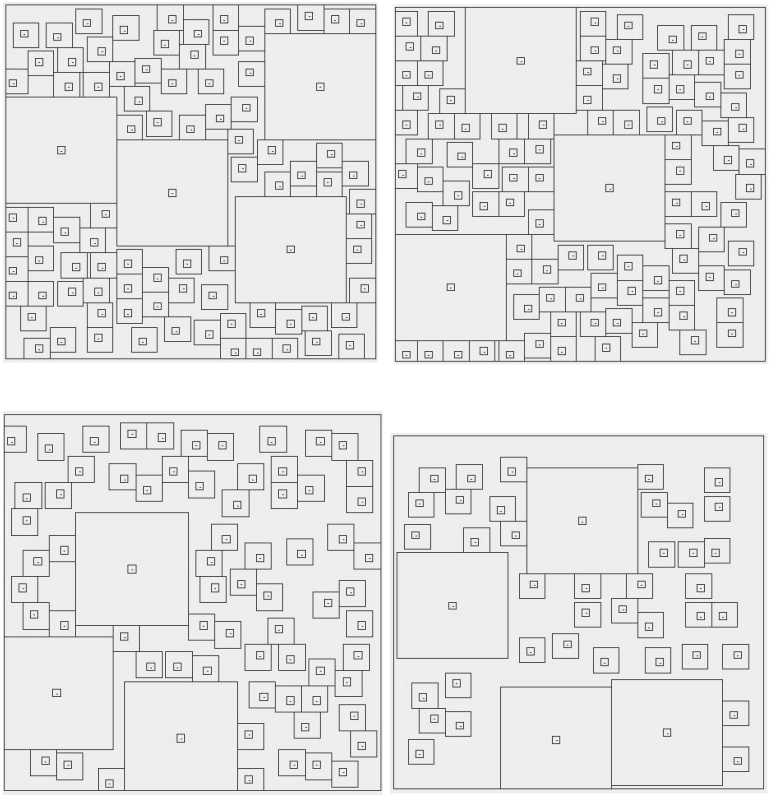
Entonces a la hora de actualizar la matriz de feromona, actualizamos para cada matriz los puntos tapados por los pads que tienen esa semilla. Por ejemplo, si tenemos 2 semillas, una grande y una chica, en una matriz de feromona vamos a actualizar los puntos tapados por los pads chicos y en la otra matriz 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.

(agregar algunos ejemplos de muchas feromonas)

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



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. Experimentacin

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.

5. Resolucion alternativa

6. Programacion lineal entera

7. Conclusin

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.