Determinación de Frentes de Pareto

Eduardo Tapia
Maestria en Ciencias de la Computación
Cimat
Guanajuato, Mexico
eduardo.tapia@cimat.mx

I. Introducción

El poder determinar el frente de pareto dado un conjunto de soluciones es una parte importante de los problemas de optimizacion multi objetivo, por ello resulta importante tener una metodologia se presentan casos en los que se observa como la implementacion puede afectar el tiempo que toma realizar estas evaluaciones, ya que es importante realizarlo rapidamente dado que se realizara varias veces durante el proceso de optimizacion.

II. Problemas multi objetivo

Un problema de optimizacion multiobjetivo consiste en encontrar un vector de soluciones tal que cumpla con las restricciones del problema a resolver y al mismo tiempo optimice todas las funciones de interés, las cuales en muchos casos, se encuentran en conflicto unas con otras. el vector de solución para un espacio de búsqueda n dimensional tiene la forma $\overline{x}^* = [x_1, x_2, x_3, \cdots, x_n]$ tal que optimiza la solucion la cual se encuentra en el espacio k dimensional de los objetivos.

$$\overline{f}(\overline{x}) = [f_1(x), f_2(x), f_3(x) \cdot \cdot, f_k(x)] \tag{1}$$

A partir de estas 2 ecuaciones, podemos definir el problema de optimizacion como el proceso que busca el vector $\overline{x}$ tal que optimiza el vector $\overline{f}(\overline{x})$

III. Dominancia y Frente de Pareto

A. Dominancia de soluciones

Cuando se resuelve un problema multi objetivo, encontrar una solucion optima no resulta tan sencillo como encontrarla en un problema mono objetivo, en el ultimo caso, simplemente es necesario conocer cual evaluacion en el espacio solucion es menor y eso es suficiente para determinar si una calidad es mejor que otra, sin embargo en el caso de las funciones multiobjetivo resulta complicado determinar si una solucion es mejor que otra únicamente comparando los valores de las fuciones a optimizar, especialmente si se toma en cuenta el hecho de que muchas veces estas soluciones son dependientes entre si. Es por ello que es necesario tener un criterio para seleccionar un conjunto de soluciones sobre las cuales un decisor pueda elegir la que resulte mas conveniente.
De aqui surge el concepto de Dominancia de Pareto, el cual se define como sigue:
Un vector u domina al vector $\overline{v}$ (denotado como $\overline{u} \preceq \overline{v}$) si y solo si u es parcialmente menor que v, es decir $\forall i \in \{1, \cdots, k\}$, $u_i \leq v_i \wedge \exists i \in \{1, ..., k\}$: $u_i < v_i$. Esto significa que para un par de vectores dado se pueden presentar 4 casos:

- $u \preceq$ v donde u domina v
- $v \preceq u$ donde v domina u
- $u = v$
- caso de no dominancia entre u y v

B. Frente de pareto

El frente de pareto o el conjunto de optimos de pareto es el conjunto que para un problema de optimizacion multi objetivo se define como:

$$P^* := \{x \in \Omega | \nexists x' \in \Omega f(x') \preceq f(x)\} \tag{2}$$

De manera que el frente de pareto se presenta como el conjunto de todas las soluciones al problema de optimización no dominadas.

IV. Búsqueda del frente de pareto

Una vez obtenidas una serie de soluciones, es necesario poder identificar cuales de estas soluciones pertenecen al frente de pareto y cuales se encuentran dominadas por este, para ello existen diversas formas de analizar los conjuntos de soluciones, de manera que dado un conjunto de soluciones se pueda encontrar el conjunto de pareto de dichas soluciones. El caso mas sencillo de de este tipo de algoritmo para cualquier numero de dimensiones en el espacio objetivo el algoritmo es el siguiente: Este algoritmo funciona para encontrar el set de pareto para problemas k dimensionales en el espacio de los objetivos, sin embargo tiene un problema muy claro, su complejidad es de $\mathcal{O}(kn^2)$ lo que se vuelve problematico conforme el numero de soluciones a evaluar crece, es por ello que se han propuesto diversas formas de encontrar el frente de pareto con menor complejidad.

Algorithm 1 Pareto ineficiente

```
Se define el vector sobre el que se guardara la solucon de pareto par
for i en soluciones do
    for j en soluciones do
        if i ≠ j then
            dom = soluciones[i] ⪯ soluciones[j]
            if (dom=SECOND_DOMINATES) or (dom==EQ) then
                break;
            end if
        end if
    end for
    if (SECOND_DOMINATES=dom or dom=EQ) then
        continue;
    end if
    par.push_back(soluciones[i]);
end for
Regresar par
```

A. Frente de Pareto eficiente 2D

Utilizando un esquema de divide y venceras el algoritmo para encontrar el frente de pareto para datos con k=2 es el siguiente: En este algoritmo la parte clave se encuentra en

Algorithm 2 Pareto divide&conquer

```
Recibe soluciones
Ordena las soluciones lexicográficamente
pareto=divide(0,soluciones.size()-1,soluciones)
return pareto;
```

la funcion divide, la cual hace llamadas recursivas dividiendo por la mitad el tamaño de las soluciones y encontrando por intervalos separados el frente de pareto, de la siguiente forma:

Donde la funcion check last revisa cuales de los elementos de la mitad derecha se encuentran dominados por la mitad izquierda y los elimina del frente.

Esta implementacion tiene una complejidad de $\mathcal{O}(knlogn)$ lo que significa una reduccion considerable en el numero de operaciones necesarias mejorando asi el tiempo de computo en casos de poblaciones muy numerosas.

Algorithm 3 Función divide

```
Recibe sol,init,end,pareto;
if end-init=1 then
    dm=dominancia(sol[init],sol[end])
    if(dm=FIRST_DOMINATES)
    pareto.push_back(sol[init])
    if(dm=SECOND_DOMINATES)
    pareto.push_back(sol[end])
    if(dm=EQ) break
    if(dm=NON_DOMINATION) break
else if (end-int=0) then
    pareto.push_back(sol[init])
else
    n=init+ceil((end-init)/2)
    divide(int,n,sol,pareto)
    last=pareto.size()
    divide(n,end,sol,pareto)
    check_last(pareto,last)
end if
```

B. Frente de Pareto eficiente 3D

Para encontrar el frente de pareto de manera eficiente para problemas de 3 dimensiones, uno de los metodos consiste en apoyarse sobre una estructura de datos llamada ”segment tree” la cual sirve para realizar busquedas de los valores contenidos en un arreglo, en este caso lo utilizaremos para buscar el valor mas pequeño contenido en un arreglo dado un intervalo, esta busqueda tiene una complejidad en tiempo de $\mathcal{O}(logn)$. El algoritmo utilizando el segment tree seria el siguiente:

Algorithm 4 Función pareto eficiente 3D

```
Recibe sol;
n=solutions.size()
Coord_compresion(solutions)
sort(solutions)
vec_int aux(n,INT_MAX)
SegmentTree tree(aux)
for x in solutions do
        temp=x[1] if (x[2]¡tree.rmq_v(0,temp0) then
        tree.update(temp,x[2])
        pareto.push_back(x)
    end if
end for
```

En el algoritmo 4 se puede observar que depende fuertemente de la funcion rmq y de la funcion update, como se menciono previamente las busquedas en el segment tree tienen una complejidad de $\mathcal{O}(logn)$, la funcion update tiene la misma complejidad $\mathcal{O}(logn)$ por lo que para un solo individuo se tiene una complejidad de $\mathcal{O}(logn)$ para determinar si es dominado o no, dado que eso se tiene que realizar para n elementos, la complejidad de esta parte,resulta en $\mathcal{O}(nlogn)$, si consideramos el ordenamiento lexicografico $\mathcal{O}(knlogn)$ y la coompresion de coordenadas que tambien es de $\mathcal{O}(knlogn)$, podemos

determinar que este algoritmo complejidad de $\mathcal{O}(knlogn)$ en el peor de los casos.

V. Metodologia

Las pruebas realizadas fueron con el numero de individuos definido por el siguiente conjunto $n \in \{2^i | i \in [8, 12]\}$ y se generaron conjuntos de datos con las siguientes caracteristicas:

- Poblacion aleatora
- Poblacion de no dominados
- Poblacion aleatoria limitada por un plano

Se realizaron las pruebas para 2 y 3 dimensiones, con los diversos tamaños de población y las 2 versiones del algoritmo, eficiente e ineficiente, para observar la diferencia en comportamiento de los algoritmos en cada caso.

VI. Resultados

A. Pareto 2D

Para las soluciones en 2D se obtuvieron los tiempos que se pueden observar en la figura 1 Y como se puede ver en la

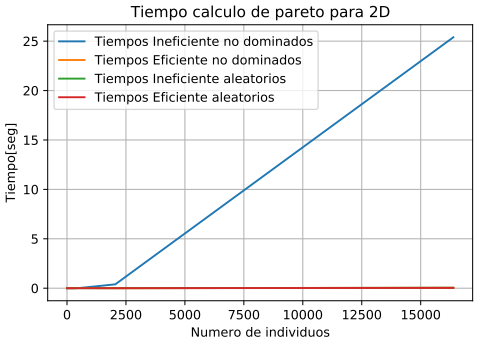

Fig. 1. Tiempos del calculo para problemas 2D

imagen, el tiempo ineficiente en el caso de los no dominados se dispara ya que es el peor de los casos, eso significa que compara cada solucion con todas las demas mientras que en el caso eficiente, el tiempo se mantiene mucho mas bajo, aun en el peor de los casos.

A cotinuacion se presenta una imagen en la que se sobreopone el frente de pareto con el set de datos obtenidos en una de las pruebas 2D, con fines ilustrativos.

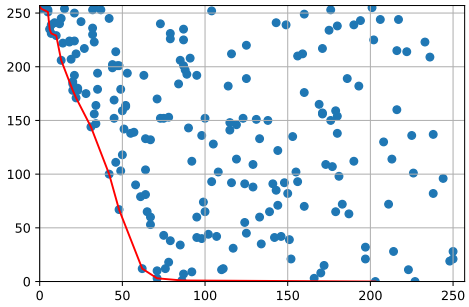

Fig. 2. visualizacion pareto 2D

B. Pareto 3D

Para las soluciones 3D lo que se obtuvo es lo siguiente:

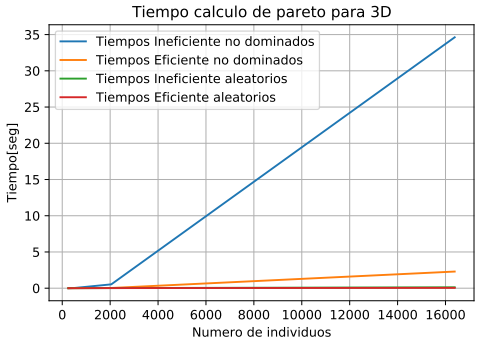

Fig. 3. Tiempos 3D

En la figura 3 se puede ver que el caso de los no dominados resulta un poco mas complicado en 3D que en 2D ya que aun utilizando el segment tree, es necesario realizar una actualizacion en cada uno de los elementos, esto aunado a la dimension extra genera que el tiempo sea ligeramente mayor que el observado en el caso 2D, sin embargo sigue siendo muy inferior al tiempo necesario para el caso ineficiente.