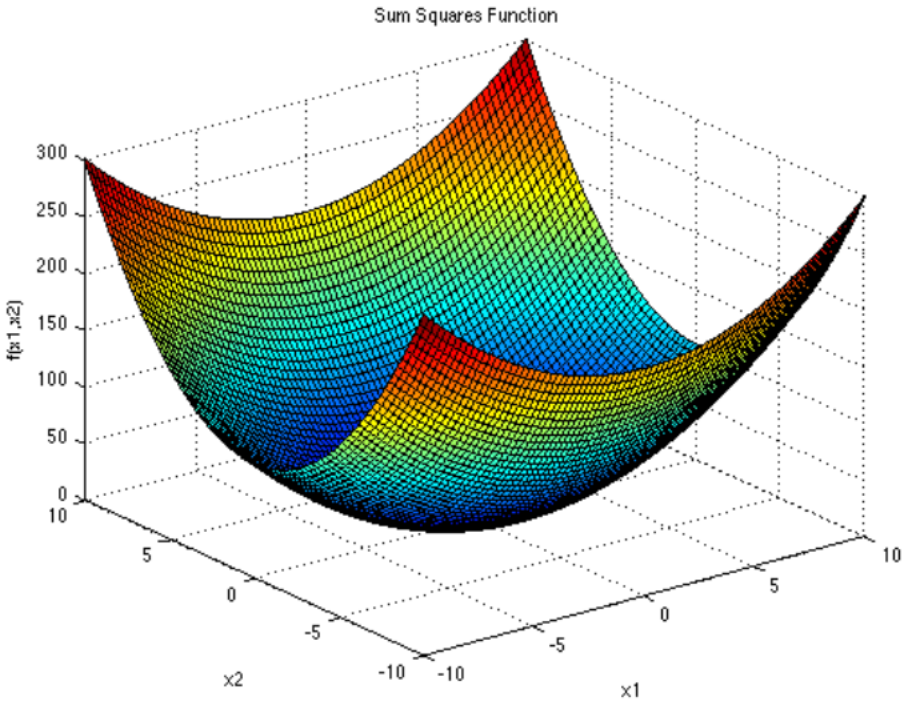
Tarea 1 - Problemas de Optimización

1. Funciones que se utilizan para probar métodos de optimización continua.

   (a) **Sum Squares Function**[SB13b]



$$f(x) = \sum_{i=1}^{d} i x_i^2$$

Descripción

Podemos notar que la función Sum squares posee las siguientes características:

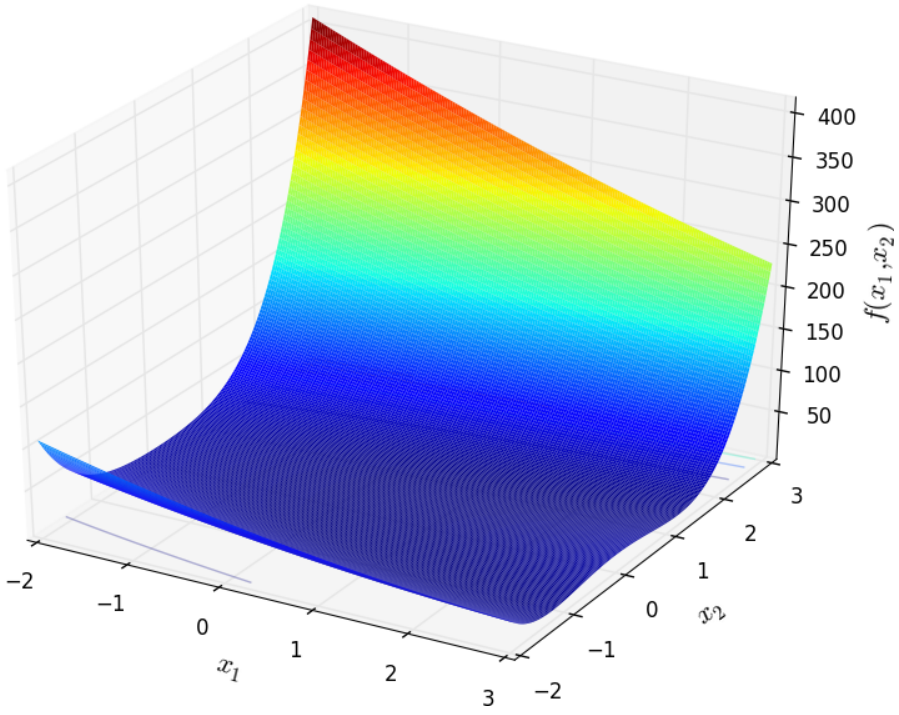
i. *Continua*
   Observando la gráfica, sabemos que ésta no tiene interrupciones, lo que significa que en cualquier punto de la función existe el límite y éste coincide con el valor que toma la función en tal punto.

ii. *Convexa*
   Dado que la gráfica tiene forma de cuenco, para cualesquiera dos puntos, la línea que los une queda por encima de la función.

iii. *Unimodal*
   Notamos que la gráfica únicamente tiene mínimo global, no cuenta con mínimos locales.

Dimensión
La dificultad es independiente a la dimensión, puesto que aunque aumente la dimensión, la función sigue conservando su único mínimo global.

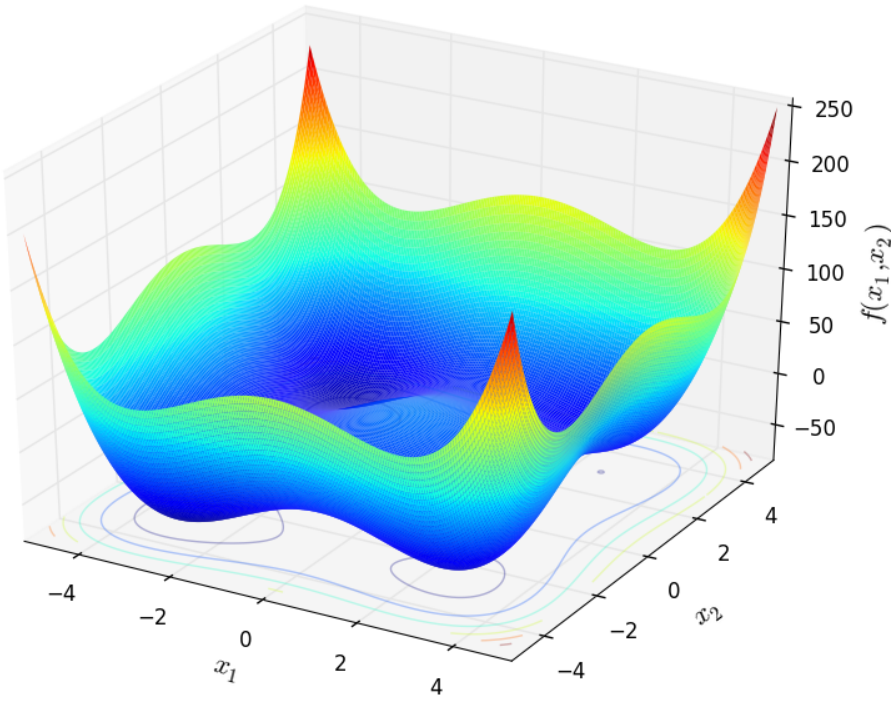
DIXON-PRICE FUNCTION [SB13a]



$$f(x) = (x_1 - 1)^2 + \sum_{i=2}^{d} i(2x_i^2 - x_{i-1})^2$$

Descripción
La función Dixon-Price en su forma bidimensional, se asemeja a una hoja cayendo sobre una superficie, por lo que ésta es continua, no es convexa puesto que al tener bajadas y subidas, la unión de algunos puntos específicos se encuentran por debajo de la función.
Dimensión
La dificultad se mantiene conforme aumentan las dimensiones, ésto porque la función no cuenta con bajadas tan profundas y es posible guiarse de más información de las demás dimensiones para encontrar el mínimo global.

Styblinski-Tang Function[ST13]



$$f(x) = \frac{1}{2}\sum_{i=1}^{d}(x_i^4 - 16x_i^2 + 5x_i)$$

Descripción

La función Styblinski-Tang en su forma bidimensional, cuenta con tres mínimos locales y un mínimo global, usualmente es evaluado utilizando hipercubos $x_i \in [-5, 5]$ para $i = 1, 2, ..., d$.

Notamos que la función es continua, puesto que no hay saltos en su gráfica, de igual forma sabemos que no es convexa, dado que para cualesquiera dos puntos, no se puede asegurar que la unión de éstos queda por encima de la función.

Dimensión

La dificultad aumenta conforme más dimensiones se agregan, dado que el espacio de búsqueda aumenta y sabemos que ésta función tiene subidas y bajadas profundas que hacen más difícil obtener el mínimo global.

¿Cuál podría ser más fácil y difícil de optimizar?

Al analizar las tres distintas funciones, notamos que el orden de dificultad para optimizarlas es el siguiente:

1. Sum square
Es la función más fácil de optimizar, puesto que cuenta únicamente con un mínimo global.

2. Dixon-Price
Tiene una dificultad intermedia, pues cuenta con varios mínimos locales, sin embargo no hay una gran diferencia entre éstos y es relativamente sencillo salir de éstos para encontrar el mínimo local.

3.Styblinski-Tang
Es la función más difícil de optimizar, puesto que cuenta con varios mínimos locales y se debería implementar un método robusto y perseverante que permita empeorar el resultado para acercarnos a un valor más preciso, para finalmente hallar el mínimo local.

(b) **Implementación de las funciones**

i. Sum Squares Function
Pseudocódigo:

Algorithm 1 Suma de Cuadrados

Require: x es un arreglo de números
Ensure: *resultado* al aplicar la función a los elementos en x.

1: $resultado \leftarrow 0$
2: **for** $i \leftarrow 1$ to número de elementos $x_i \in x$ **do**
3: $\quad resultado \leftarrow resultado + i \cdot x_i^2$
4: **end for**
5: **return** *resultado*

Implementación:

```java
public double sumSquare(double[] valores) {
    double res = 0;
    for (int i = 0; i < valores.length; i++) {
        res += (i + 1) * Math.pow(valores[i], b:2);
    }
    return res;
}
```

ii. Dixon-Price Function
Pseudocódigo:

Algorithm 2 Dixon-Price

Require: x es un arreglo de números
Ensure: *resultado* al aplicar la función a los elementos en x.

1: $resultado \leftarrow x_1 - 1 \quad (x_1 \in x)$
2: **for** $i \leftarrow 2$ to número de elementos $x_i \in x$ **do**
3: $\quad resultado \leftarrow resultado + i(2x_i^2 - x_{i-1})^2$
4: **end for**
5: **return** $resultado/2$

Implementación:

```java
public double dixonPrice(double[] x) {
    double res = Math.pow(x[0] - 1, b:2);
    for (int i = 1; i < x.length; i++) {
        res += (i + 1) * Math.pow(2 * Math.pow(x[i], b:2) - x[i - 1], b:2);
    }
    return res;
}
```

iii. Styblinski-Tang Function

Presudocódigo:

Algorithm 3 Styblinski-Tang

Require: x es un arreglo de números

Ensure: *resultado* al aplicar la función a los elementos en x.

1: $resultado \leftarrow 0$
2: **for** $i \leftarrow 1$ to número de elementos $x_i \in x$ **do**
3: $\quad resultado \leftarrow resultado + resultado + x_i^4 - 16x_i^2 + 5x_i$
4: **end for**
5: **return** *resultado*

Implementación:

```java
public double StyblinskiTang(double[] x) {
    double res = 0;
    for (int i = 0; i < x.length; i++) {
        res += Math.pow(x[i], b:4) - 16 * Math.pow(x[i], b:2) + 5 * x[i];
    }
    return res / 2;
}
```

(c) **Evaluación en las funciones**
Para la selección de las funciones que implementamos se requiere:

i. Un índice k que inidica en qué función se desea evaluar.
ii. Un entero d que tiene el valor de la dimensión deseada
iii. x_i valores para evaluar ($i \in \{1, \cdots, d\}$)

Ejemplos de ejecución:

Ej. 1
$k = 1$ para función SumSquares
$d = 2$ para dimensión 2
$x_1 = 6.32, x_2 = -0.46$

```
PS C:\Users\ginso\Documents\semestre 4\ComputoEvolutivo\CE_Tarea01> java -jar target/ejecuta.jar 1 2 6.32 -0.46
Dimensión: 2
Valores de x_i:
x_1 = 6.32
x_2 = -0.46
Resultado:
40.36560000000001
```

Ej. 2
$k = 2$ para función Dixon-Price
$d = 3$ para dimensión 3
$x_1 = 5.4, x_2 = 6.32, x_3 = 0.35$
java -jar target/ejecuta.jar 2 3 5.4 6.32 0.35

```
PS C:\Users\ginso\Documents\semestre 4\ComputoEvolutivo\CE_Tarea01> java -jar target/ejecuta.jar 2 3 5.4 6.32 0.35
Dimensión: 3
Valores de x_i:
x_1 = 5.4
x_2 = 6.32
x_3 = 0.35
Resultado:
11226.047737080004
```

Ej. 3
$k = 3$ para función Styblinski Tang
$d = 4$ para dimensión 4
$x_1 = 5.4, x_2 = 6.32, x_3 = 0.35, x_4 = 3.83$
java -jar target/ejecuta.jar 3 4 5.4 6.32 0.35 3.83

```
PS C:\Users\ginso\Documents\semestre 4\ComputoEvolutivo\CE_Tarea01> java -jar target/ejecuta.jar 3 4 5.4 6.32 0.35 3.83
Dimensión: 4
Valores de x_i:
x_1 = 5.4
x_2 = 6.32
x_3 = 0.35
x_4 = 3.83
Resultado:
699.0458756100002
```

2. Implementar una búsqueda aleatoria para problemas de optimización continua.

   (a) **Función**

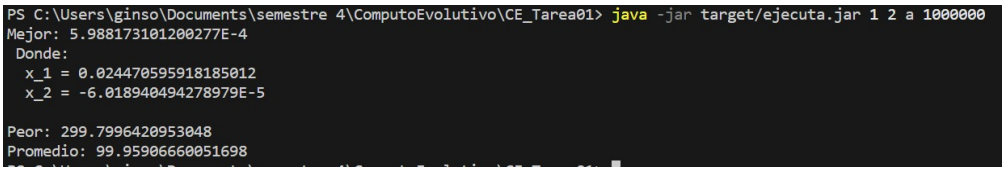
   Búsqueda aleatoria
   Para la búsqueda aleatoria se requieren 4 argumentos

   i. Un índice k que inidica en qué función se desea evaluar.
   ii. Un entero d que tiene el valor de la dimensión deseada
   iii. El caracter $'a'$ que indica búsqueda aleatoria
   iv. Un entero positivo *iter* que es el número de iteraciones que deseamos que se hagan

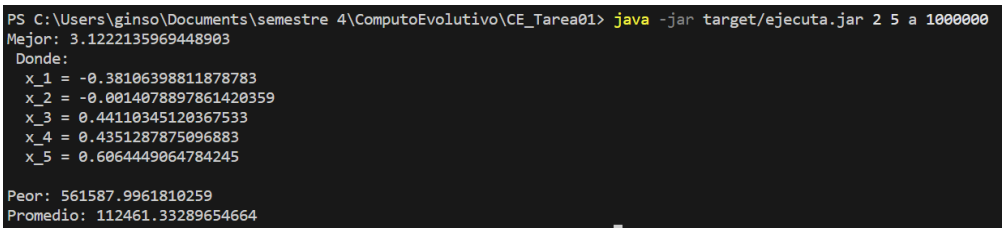
   java -jar target/ejecuta.jar k d a iter

   Ejemplos de ejecución:

   Ej. 1
   $k = 1$ para función SumSquares
   $d = 2$ para dimensión 2
   $iter = 1,000,000$ para un millon de ejecuciones
   java -jar target/ejecuta.jar 1 2 a 1000000



   Ej. 2
   $k = 2$ para función Dixon-Price
   $d = 5$ para dimensión 5
   $iter = 1,000,000$ para un millon de ejecuciones
   java -jar target/ejecuta.jar 2 5 a 1000000



Ej. 3
$k = 2$ para función Styblinski Tang
$d = 10$ para dimensión 5
$iter = 1,000,000$ para un millon de ejecuciones
java -jar target/ejecuta.jar 3 10 a 1000000

```
PS C:\Users\ginso\Documents\semestre 4\ComputoEvolutivo\CE_Tarea01> java -jar target/ejecuta.jar 3 10 a 1000000
Mejor: -348.4238687866076
 Donde:
  x_1 = -3.101033234082254
  x_2 = -2.764472852194957
  x_3 = -2.7991685290372836
  x_4 = -2.464471499417671
  x_5 = 1.6503114765416989
  x_6 = -3.1608634942310774
  x_7 = -2.7460154340181573
  x_8 = -3.627859733510724
  x_9 = -3.0411947189232444
  x_10 = -3.0740368582764455

Peor: 595.0297672003187
Promedio: -41.53591884217943
```

(b) **Tabla**
Análisis de los datos:

Función	Dimensión	Mejor valor f(x)	Valor Promedio f(x)	Peor Valor f(x)
Sum Squares	2	3.57E-04	100.1055544	299.7961592
Sum Squares	5	2.905597709	500.2460415	1430.813709
Sum Squares	10	100.6714711	1833.309827	4880.211803
Dixon-Price	2	2.01E-04	16099.15999	88180.01013
Dixon-Price	5	3.779827101	112472.2916	557533.9459
Dixon-Price	10	1026.846389	1691550.771	433722.0258
Styblinski-Tang	2	-78.33193706	-8.404385532	246.8290384
Styblinski-Tang	5	-191.2365544	-20.90218031	479.7691703
Styblinski-Tang	10	-336.3947867	-41.6168216	646.1896918

En las fuentes ya se nos dice cuál es el óptimo global de cada una de las funciones, usaremos eso para analizar cada una de las funciones con su resultado de búsqueda aleatoria.

Sum Squares Function
Para *SumSquares* su mínimo global se encuentra en $f(x^*) = 0$, donde $x^* = (0, \cdots, 0)$. [SB13b]
En la búsqueda aleatoria de dimensión dos, el mejor resultado fue 0.00357, lo cual comparado con la dimensión 5 y 10, es una minimización mucho mejor (aunque

realmente no encontró el mínimo global).
Por otro lado, se nota que la dimensión afecta mucho (para esta función) si se desea hacer una búsqueda aleatoria, ya que para tener una solución que se acerque al mínimo global, cada una de sus variables tendría que acercarse también al cero. En la dimensión 10, encontrar de manera aleatoria una combinación en la que todas se encuentren decentemente cercanas al 0 se vuelve mucho más difícil que en la dimensión 2, ya que requieren 10 y 2 variables respectivamente.

Como la computadora no tiene a todos los reales, si n es la cantidad de números representables, la cantidad de combinaciones sería de n^d, lo que significaría que por cada dimensión extra, el número de combinaciones se multiplicaría por n.
Además, de la tabla nos podemos dar cuenta que el promedio está bastante alejado del mejor valor, lo que nos podría indicar que muchas de las búsquedas se hacen muy alejadas de donde va encontrando los mejores valores.

Dixon-Price Function
Para *Dixon-price* su mínimo global se encuentra en $f(x^*) = 0$, donde $x^* = (2^{-\frac{2^1-2}{2^1}}, \cdots, 2^{-\frac{2^d-2}{2^d}})$. [SB13a]
EN esta función se nota mucho más como la dimensión afecta la búsqueda aleatoria, ya que mejor valor para dimensión 2 es aproximadamente 0.00029, pero el mejor valor para la dimensión 10 es aproximadamente 1026.84, el óptimo global de ambos es el cero, pero la búsqueda aleatoria en la dimensión 10 es más de mil veces mayor.

Styblinski-Tang function
Para *Styblinski-Tang* su mínimo global se encuentra en $f(x^*) = -39.16 * d$, donde $x^* = -2.9035, \cdots, -2.9035)$. [SB13a]
Esta función me sorprendió, ya que aunque concluimos que optimizarla sería difícil porque tiene varios mínimos locales, al parecer la búsqueda aleatoria no se estanca en esos mínimos locales, porque ni siquiera los reconoce. Eso ayudó a que en todas las dimensiones el mejor valor no esté tan alejado del mínimo global.

3. Mencionar algún ejemplo de problema de optimización combinatoria, diferente a los mencionados en clase.
Indicar claramente:
- Espacio de búsqueda
- Función objetivo
- Tamaño del espacio de búsqueda
- Ejemplar concreto del problema
- Ejemplo de una solución

Problema del bombero [Val19]

El problema del bombero plantea un conjunto de vértices (casas), las cuales pueden

estar en tres estados distintos:
1. Estado inicial - Éstas casas no son modificadas
2. Ardiendo - Éstas casas pueden propagar el fuego a sus vértices vecinos, alguna casa vecina les exparció el fuego o es la casa que inicia el incendio.
3. Protegida - Éstas casas no se pueden incendiar y no dejan pasar el fuego, puesto que han llegado los bomberos.

En el tiempo inicial una casa comienza a arder y es posible colocar bomberos (limitados) para detener el incendio, en el tiempo siguiente, los vecinos de la casa inicial en llamas que no tengan bomberos comienzan a arder, entonces es posible colocar una nueva oleada de bomberos, continúa hasta que el fuego no se pueda propagar más, ya sea que todas las casas se quemen o los bomberos hayan detenido el incendio.

En este problema es posible minimizar diferentes recursos, ya sea la cantidad de bomberos colocados o el tiempo mínimo para detener el incendio.
En esta ocasión, nos enfocaremos en minimizar la cantidad de bomberos a colocar para detener el incendio, por lo tanto, tenemos lo siguiente:

(a) Espacio de búsqueda
El espacio de búsqueda son todas las posibles combinaciones de bomberos colocados en las casas, exceptuando la que empieza a arder, puesto que si todas las casas están protegidas, nunca se origina ningún incendio.

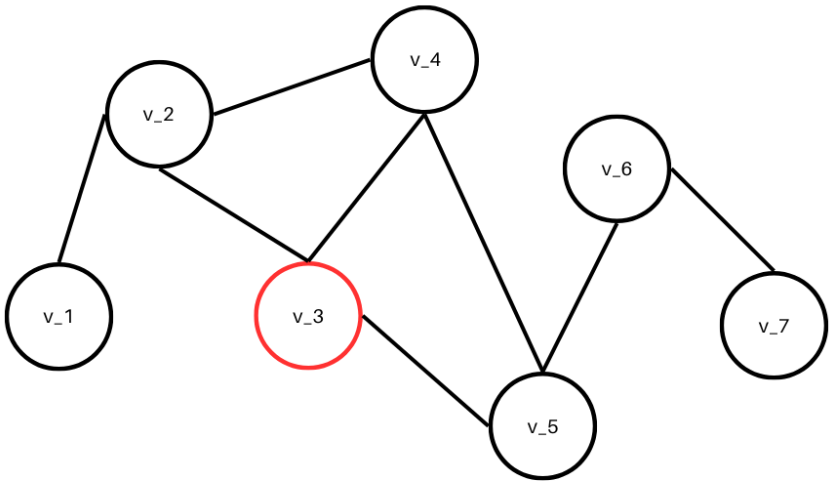
(b) Función objetivo
La función objetivo, en este caso, es encontrar una organización óptima de bomberos, es decir, asociar cada vértice de la gráfica con un valor que represente si está protegida o no, como pueden están en tres distintos estados, podemos representar a cada uno con un valor numérico, 0 para el estado inicial, 1 para el estado ardiendo y 3 para el estado protegido.

(c) Tamaño del espacio de búsqueda
El tamaño del espacio de búsqueda, definido anteriormente, toma en cuenta la cantidad de vértices n de la gráfica y la cantidad de bomberos m disponibles.
Por lo tanto, podemos expresar el tamaño de la siguiente forma:

$$(n-1)^m$$

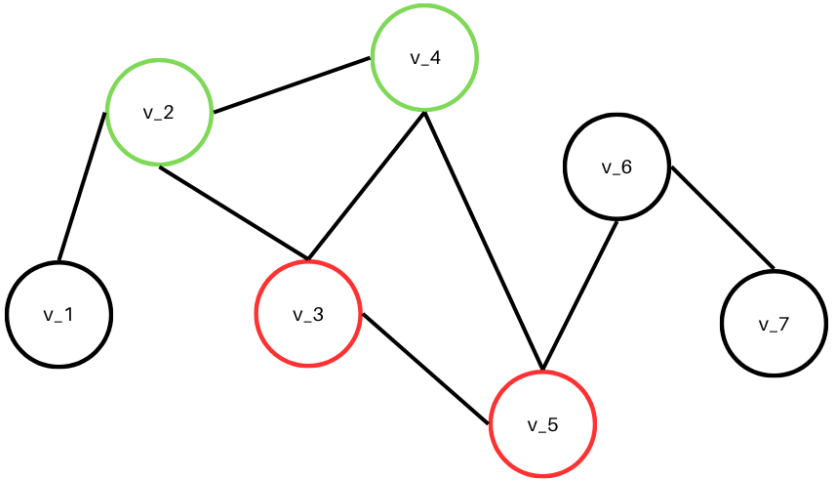
(d) Ejemplar concreto del problema y ejemplo de una solución

Consideramos un vecindario con siete casas enumeradas del 1 al 7. En el tiempo 0, todas las casas comienzan en un estado inicial, sin embargo, la casa número tres cambia su estado a “ardiendo”.



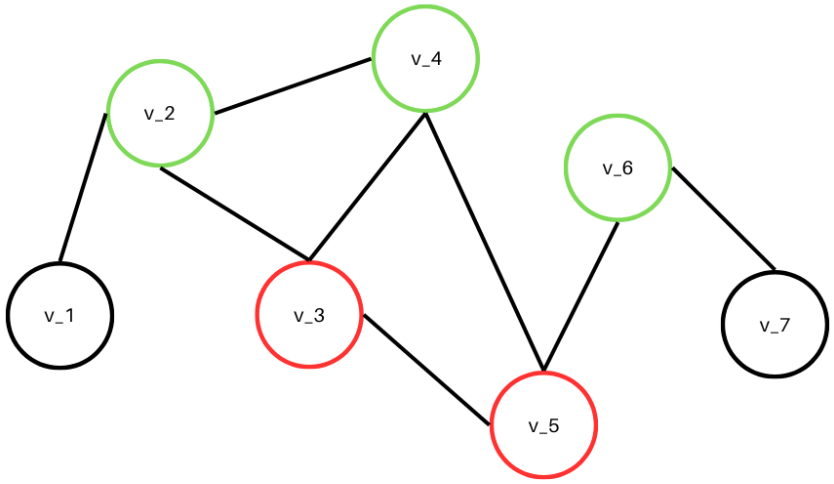
t=0

En el tiempo 1, colocamos dos bomberos en las casas 2 y 4, éstas cambian su estado a "protegida". La casa número 5, al no estar protegida y es vecina de la casa en llamas, entonces a ésta se le propaga el fuego y cambia su estado a "ardiendo".



t=1

En el tiempo 2, colocamos un bombero en la casa 6, lo que provoca que el fuego no pueda propagarse más y terminamos el problema.



t=2

Por lo tanto, los valores de nuestros vértices en la gráfica una vez resuelto el problema, recordando los valores numéricos, son:

$$G = ((v_1, 0), (v_2, 3), (v_3, 2), (v_4, 3), (v_5, 2), (v_6, 3), (v_7, 0))$$

Notamos que fueron necesarios 3 bomberos y el tiempo mínimo fue de 2.

En el ejemplo anterior, el número de bomberos no estaba definido, normalmente es un número limitado, puesto que sería sencillo rodear la primera casa en llamas con bomberos para frenar el incendio, como ésto no siempre es posible en las aplicaciones reales, como incendios, infestaciones, plagas, etc., entonces minimizar el número de bomberos y su configuración óptima es realmente importante.

References

[SB13a] Sonja Surjanovic and Derek Bingham. Dixon-price function optimization test problems. *Virtual Library of Simulation Experiments: Test Functions and Datasets*, 2013.

[SB13b] Sonja Surjanovic and Derek Bingham. Sum squares function optimization test problems. *Virtual Library of Simulation Experiments: Test Functions and Datasets*, 2013.

[ST13] M.A. Styblinski and S. Tang. Problem 10: The styblinski-tang function. *Global Optimization Test Functions Index*, 2013.

[Val19] Canek Peláez Valdés. Tres problemas a resolver con heurísticas de optimización combinatoria. *Youtube*, 2019.