Tecnológico de Monterrey

Proyecto Final

Entrega Final

Estefania Jimenez Garcia A01635062
Luis Gerardo Reyes Ceja A01635286

Análisis y diseño de algoritmos

Introducción

El problema del arreglo linear mínimo (MinLA, por sus siglas en inglés Minimum Linear Arrangement) es un algoritmo de optimización de errores que fue constatado por primera vez por L.H. Harper en una entrada al Diario “Optimal assignment of numbers to vertices” del SIAM en 1964. El problema consiste en encontrar el mejor conjunto ordenado de los vértices de un determinado grafo que minimice la suma de los valores absolutos de las diferencias de costo entre vértices adyacentes. Este algoritmo produce soluciones de problemas bastante óptimos, con la única desventaja siendo sus largos tiempos de de procesamiento, lo que lo vuelve un blanco activo en el área de investigación debido a que siempre se busca una manera de optimizar, agilizar y reducir el algoritmo. Conforme han avanzado las investigaciones acerca de este algoritmo se ha descubierto varias optimizaciones una de estas siendo, movimientos alternativos, entre otros acercamientos. En este proyecto, se busca encontrar una solución más óptima a la ya establecida por profesionales de la materia, actualmente, existen varios acercamientos a diferentes algoritmos que podrían facilitar el tiempo de procesamiento, el cual es el mayor defecto de este algoritmo y de la mayoría de aproximaciones al mismo. El lenguaje que utilizado dentro del desarrollo del proyecto para esta aproximación sería Java, aunque este lenguaje es inferior en materia de optimización de recursos y versatilidad a c , pero existe una mayor experiencia en implementación de estructuras de datos para este algoritmo en Java. Es importante mencionar que el algoritmo puede llegar a ser excesivamente cargado para ser computado en un lenguaje que no optimiza los recursos en comparación con otros, esto ofrece una capa de dificultad extra dentro del desarrollo del proyecto. Este es un problema que llama bastante la atención debido que posee muchas aplicaciones prácticas y teóricas, las cuales ya han sido fuertemente estudiadas, generando respuestas heurísticas eficientes.

Definición formal del problema combinatorio analizado

Cuando hay grafo no direccionado denominado $G(V,E)$ donde $V(|V|=n)$ (n siendo el número de nodos que contiene el grafo) se dice que puedes obtener los pares de vértices que cumplen con la condición: $E \subset V \times V = \{\{i,j\}: i,j \in V\}$ utilizando la fórmula para cada par: $|\varphi(u) - \varphi(v)|$ (u y v siendo los valores numéricos de ambos nodos que se conectan entre el vértice).

Al final se puede calcular el valor final de los vértices en el grafo con la fórmula:

$$LA(G,\varphi) = \sum_{(u,v) \in E} |\varphi(u) - \varphi(v)|$$

Todo esto tiene como fin de poder encontrar cual es el mejor camino con el menor costo de los vértices en el grafo.

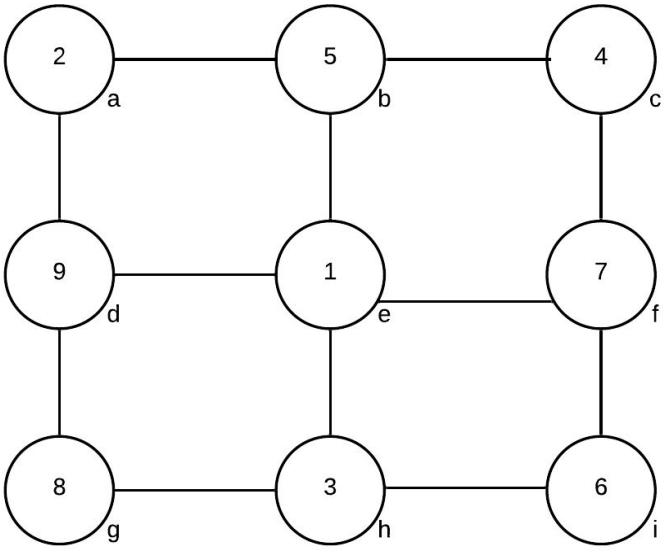
Entre las 17 soluciones optimizadas, resaltan 5 para nuestro gusto particular:

La solución de planteada por McAllister y su "frontal increase minimization" en el cual divide el largo del arreglo en dos sets diferentes el cual, en comparación con un algoritmo de ancho similar, como lo es EB (eigenvalue-based) pero su algoritmo es mucho más competitivo puesto que ofrece una mucho mejor solucion en varios escenarios.

También Koren y Harel presentaron en 2002 otro algoritmo para el problema de MinLA basado en la combinacion de metodos del paradigma de MS transformando un problema de alto nivel iterativo en subproblemas más simples, el problema es resuelto exactamente igual siguiendo ciertos procesos de refinamiento, la solución es proyectada y actualizada apropiadamente en cada escala hasta que el problema es resuelto.

Ejemplo para ilustrar el problema combinatorio analizado

Ejemplo que nos dio el profesor en asesoría:



$$n = 9$$

$$m = 12$$

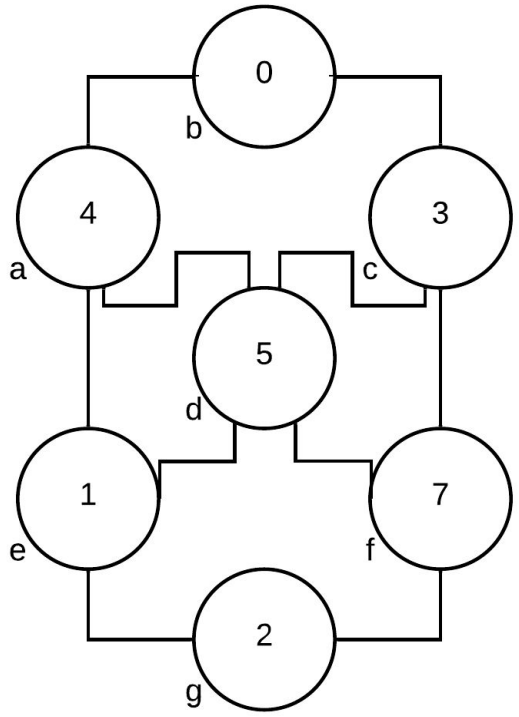
$$LA(G, \varphi) = \sum_{(u,v) \in E} |\varphi(u) - \varphi(v)|$$

$$LA(G, \varphi) = 44$$

a	b	c	d	e	f	g	h	i
2	5	4	9	1	7	8	3	6

$$|\varphi(u) - \varphi(v)|$$

1. $ab = |2 - 5| = 3$
2. $bc = |5 - 4| = 1$
3. $ad = |2 - 9| = 7$
4. $de = |9 - 1| = 8$
5. $ef = |1 - 7| = 6$
6. $dg = |9 - 8| = 1$
7. $gh = |8 - 3| = 5$
8. $hi = |3 - 6| = 3$
9. $be = |5 - 1| = 4$
10. $eh = |1 - 3| = 2$
11. $cf = |4 - 7| = 3$
12. $fi = |7 - 6| = 1$



$n = 7$

$m = 10$

$$LA(G, \varphi) = \sum_{(u,v) \in E} |\varphi(u) - \varphi(v)|$$

$LA(G, \varphi) = 29$

a	b	c	d	e	f	g
4	0	3	5	1	7	2

$|\varphi(u) - \varphi(v)|$

1. $ab = |4 - 0| = 4$
2. $ad = |4 - 5| = 1$
3. $ae = |4 - 1| = 3$
4. $bc = |0 - 3| = 3$
5. $df = |5 - 7| = 2$
6. $de = |5 - 1| = 4$
7. $eg = |1 - 2| = 1$
8. $fg = |7 - 2| = 5$
9. $fc = |7 - 3| = 4$
10. $cd = |3 - 5| = 2$

Descripción detallada del algoritmo desarrollado

Código Metaheurístico:

El algoritmo heurístico elegido para desarrollar en nuestro problema se trata de Simulated annealing (SA), que puede traducirse como recocido simulado, debido a la inspiración del algoritmo, la metalurgia. La inspiración de SA nos ayuda a entender cómo funciona dicho algoritmo y porque es que funciona.

En la metallurgia, para que un metal sea resistente y flexible, se debe de llevar a cabo un proceso llamado recocido, donde el metal es calentado a una altísima temperatura y después de dejar para un proceso de enfriado, solo para calentarse de nuevo y volver a empezar el ciclo hasta obtener el resultado deseado.

En programación es un proceso parecido, excepto que no se "recalienta" a la función más de una vez, solo al inicio. El algoritmo se basa en cambios de estado, en los que hay cambios de "energía", esto es lo que hace posible modificaciones de comportamiento.

El algoritmo se maneja con cambios de probabilidad, mientras más alta esté la variable de temperatura, más drásticos y aleatorios serán los cambios y lo contrario también cierto cuando la temperatura es baja. Con cada iteración la variable temperatura es disminuida en un cierto grado determinado por el programador para cada vez obtener resultados más concisos y lógicos.

Pseudocódigo y su explicación

s se refiere al estado actual

$s0$ se refiere al estado inicial

t es la temperatura, que la probabilidad de un cambio no óptimo en función de las iteraciones

k es la iteración actual

$kMax$ es el número máximo de iteraciones

```
Simulated annealing
Input: Initial state s
Output: Final state s

Procedure
Let s = s0
For k = 0 through kMax (exclusive):
        t ← temperature( (k+1)/kMax )
        Pick a random neighbour, sNew ← neighbour(s)
```

```
    If P(E(s), E(sNew), T) ≥ random(0, 1):
        s ← sNew
```

Lo que hace este algoritmo calcular la temperatura, que se va a reducir con cada iteración, donde a continuación procede a elegir uno de los vecinos de s, es decir, una variación del estado actual. Luego, la evalúa usando un random y la temperatura actual, si este pasa la prueba se guarda como el estado actual, si no, se descarta. De igual manera el ciclo continúa hasta que la temperatura llega a ser 0.

Clase del Grafo:

Para la forma que decidimos implementar el algoritmo fue crear una lista de nodos, cada nodo contiene la etiqueta del nodo y su lista de vecinos. A base de esto de esto obtuvimos las propiedades del grafo como el costo total y cuántos nodos contiene la gráfica aunque también se tiene las posiciones de los últimos nodos que fueron intercambiados en la función neighbors. También se implementó las siguientes funciones:

createNodes: función para crear los nodos del grafo

1. Dependiendo de cuantos nodos contenga el grafo se va creando el nodo con la etiqueta siendo el número de nodo que sea (0 a n)
2. Este nodo se agrega a la lista de nodos

addConections: función para agregar conexiones entre los nodos tiene como input la posición de los 2 nodos que están conectados (a y b)

1. Se consiguen los nodos de la lista
2. Después se agrega a la lista de vecinos del nodo a el nodo b
3. Se hace lo mismo para el nodo b
4. Y se actualiza el costo total del grafo

swapNodes: función para cambiar la posición de 2 nodos entre ellos tiene como input los nodos que se quiera hacer el swap (a y b)

1. Se revisa que los nodos no sean iguales
2. Si no son iguales, se itera entre la lista de los nodos
3. Se crea un current el cual es el nodo en la posición en i
4. Si el current tiene como vecino a entonces se elimina de la lista de vecinos y se agrega b
5. Se hace lo mismo con el nodo b
6. Después se crea 2 listas para guardar temporalmente los vecinos de los nodos a y b
7. Se borran la lista de vecinos de ambos nodos
8. Se intercambia la lista de vecinos entre los nodos

getEnergySupport: función para calcular el costo del grafo total su output es un int con el valor total del grafo.

1. Se crea una variable de la suma y se inicializa en 0
2. Se itera entre la lista de los nodos del grafo y en los vecinos de estos
3. Y mientras el nodo tenga menor costo que su vecino se usa la fórmula para sacar el costo entre esos nodos y se suma a la variable suma
4. Se regresa suma

getEnergy: función para calcular el costo del nodo su input son el nodo que se desea calcular el costo y la lista de los nodos que ya fueron visitados. Su output es el valor del nodo.

1. Se crea una variable de la suma y se inicializa en 0
2. Se revisa si el nodo ya fue visitado, si es así se regresa que el costo del nodo es 0
3. Se itera entre los vecinos del nodo
4. Si el vecino del nodo no fue visitado entonces se usa la formula para sacar su costo y se agrega a suma
5. Si no se marca como revisado
6. Se agrega a visitados el nodo
7. Se regresa la suma

neighbours: función que consigue un vecino del grafo actual tiene como input una posición random de un nodo y su output es el valor total del grafo

1. Se toma el nodo con la posición dada
2. Se crea una variables para el valor de la media de la lista de vecinos de ese nodo y se inicializa en 0
3. Se hace un sorting de su lista de vecinos, según su etiqueta
4. Se hace casos base: *si la lista es de tamaño uno entonces se actualiza la media con el único valor disponible *si la lista es de tamaño par entonces se consiguen los 2 nodos de la mitad y se saca un promedio y por último *Si la lista es de tamaño impar entonces se consigue el dato del medio.
5. Se hace un swap entre el valor de la media y el primer nodo usando la función swapNodes.
6. Y se actualiza la posición de los últimos nodos que fueron intercambiados
7. Con la función de getEnergy se regresa el costo del grafo

revertNeighbours: función para reinvertir los cambios que se hizo al grafo en la función neighbors

1. Se checa las posiciones de los últimos nodos que fueron intercambiados en la función neighbors (atributo de la clase)
2. Se manda a llamar la función swapNodes con estas posiciones

simulatedAnneling: función principal del simulated annealing

1. Se declara un random para la función de aceptación
2. Se declara una temperatura inicial
3. Se consigue la energía actual
4. Se declara un alpha para el cambio de temperatura

5. Se declara la variable de la nueva energía
6. Se declara la temperatura final
7. Mientras que la temperatura inicial sea mayor que la final, se declara el número de loops en dicha temperatura
8. Si el número de loops no se excede entonces se reduce en uno la variable de loops

acceptanceProbability: función de aceptación

1. Si la nueva energía es menor que la mejor energía actual entonces se regresa 1
2. Si la nueva energía es mayor entonces se regresa la probabilidad de esta energía
3. De otra manera se regresa .1.

Y por último se hace lectura del archivo con el número de nodos, y las conexiones entre lo nodos.

Análisis matemático de la complejidad temporal y espacial del algoritmo desarrollado

Operaciones básicas:

- Recorrer el grafo creado siendo el peor caso n^2
- Recorrer la lista de lista de los nodos ya visitados de tamaño $n-1$

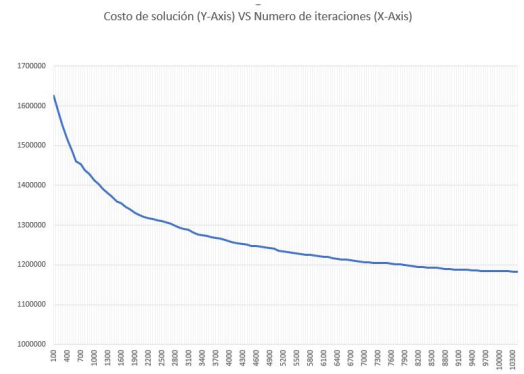
Best case scenario	Worst case scenario
Operación 1: Que no haya conexiones entre los nodos lo cual haría el grafo de tamaño n	Operación 1: Que haya conexiones entre todos los nodos del grafo el cual sería n^2
Operación 2: Que solo haya un nodo sin conexiones	Operación 2: No aplica, el peor de los casos o su caso promedio es: $n-1$

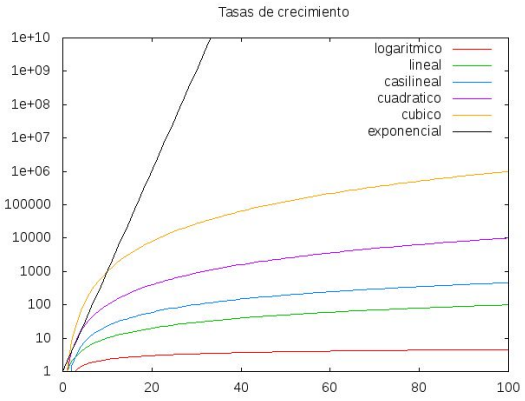
Operación 1:

$$Cworst(n) = \sum_{i=0}^{n-1} \sum_{i=1}^{n-1} 1 = (n-1)^2 \in \theta(n^2)$$

Operacion 2:

$$\sum_{i=1}^{n-1} 1 = n-1 \in \theta(n)$$





Por último si comparamos con la gráfica que tenemos de nuestro crecimiento de costo con su número de iteraciones con la gráfica de al lado podemos ver que nuestro algoritmo crece cuadráticamente.

Experimentación con el algoritmo desarrollado

Archivo	Neighbors	Nueva y vieja energía	Runtime
A-randomA1.rmf	999-Neighbours: 402 824 882 35 126 195 247 825	1216789 1688528	0:03:03
A-randomA1.rmf	999-Neighbours: 333 958 273 332 507 392 742 94	1189659 1688528	0:07:20
A-randomA1.rmf	999-Neighbours: 998 993 528 922 213 112 687 236	1188246 1688528	0:08:55
A-randomA2.rmf	999-Neighbours: 8 29 53 70 97 252 257 266 348 489 620 634 655 680 697 707 755 772 811 903 995 613 557 466 799 550 389 761 946 467 722 834 745	7264558 8278100	0:11:06
A-randomA2.rmf	999-Neighbours: 702 825 707 747 828 942 880 70 965 704 230 761 697 674 28 312 605 914 257 745 433 529 524 693 292 569 553 528 486 755 427 443 489	7277885 8278100	0:12:46
A-randomA2.rmf	999-Neighbours: 40 129 139 143 230 250 257 273 443 457 458 468 502 529 674 692 697 705 755 757 766 781 812 828 854 868 939 963 964 965 969 595 508	7240505 8278100	0:13:23
A-randomA3.rmf	999-Neighbours: 21 41 42 43 74 120 132 142 146 148 158 177 184 259 276 297 318 347 371 576 611 644 664 668 698 710 725 734 736 795 815 900 903 904 962 975 976 977 980 944 958 369 151 642 855 878 4 824 420 666 612 6 784 827 436 538 598 304 652 830 641 229 791 847 821 409 753 568 181 749 461 575 390 562 667 183 898 584 122 536 579 428 919 556 495 211 592 97 373 474 883 802 475 959 503 487 901 535 544 879 889 468 512 290	15254821 16629337	0:14:26
A-randomA3.rmf	999-Neighbours: 803 311 905 876 823 105 625 846 51 95 80 812 236 374 886 108 943 71 766 727 84 640 249 77 129 136 325 949	15234044 16629337	0:15:54

	125 779 806 298 366 252 674 930 87 919 376 861 364 958 503 375 557 608 936 683 402 716 453 741 929 629 443 268 495 783 915 687 851 415 429 154 614 572 530 517 63 481 599 615 555 984 967 560 424 644 570 575 539 461 479 658 428 535 287 512		
A-randomA3.rmf	999-Neighbours: 5 17 53 85 139 175 194 211 235 277 324 325 327 332 355 371 381 396 518 574 579 636 695 723 724 760 770 787 848 872 922 926 952 971 976 988 989 994 132 403 701 556 401 702 624 607 473 821 433 287 410 301 677 152 578 451 629 522 475 489 685 776 500 462 349 138 422 253 47 595 504 580 622	15262496 16629337	0:16:32
A-randomA4.rmf	999-Neighbours: 78 145 247 295 527 667 776 906 912 915 917 955 671 922 819 717 757	2159004 2710738	0:21:48
A-randomA4.rmf	999-Neighbours: 32 109 142 659 938 961 687 70 616 836 931 573 147 910 566 287	2153655 2710738	0:22:26
A-randomA4.rmf	999-Neighbours: 145 329 610 679 759 833 872 906 915 917 955 637 912 549 632 465 92	2134674 2710738	0:23:07
A-randomG4.rmf	999-Neighbours: 590 907 948 958 874 939 321 921 571 841 943 434	1619932 2743658	0:24:06
A-randomG4.rmf	999-Neighbours: 29 35 49 410 413 451 538 700 790 550 524 449 24 458 716	1470794 2743658	0:24:41
A-randomG4.rmf	999-Neighbours: 815 832 848 939 956 979 836 974 820 857 838 885 829 766 834	1550958 2743658	0:25:10
B-bintree10.rmf	1022-Neighbours: 510	262319 262143	0:26:56
B-bintree10.rmf	1022-Neighbours: 510	262277 262143	0:27:20
B-bintree10.rmf	1022-Neighbours: 254 1021 510	262932 262143	0:27:45
B-hc10.rmf	1023-Neighbours: 949 1015 991 761 1009 871 488	526640 523776	0:28:30

	1021 1022 1007		

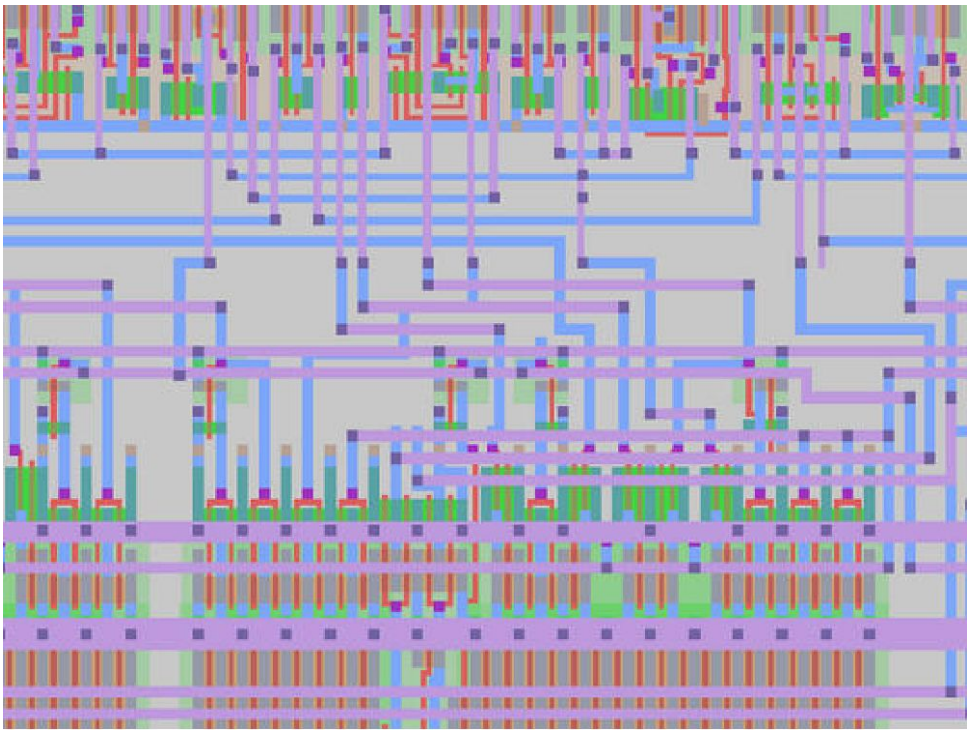
B-hc10.rmf	1023-Neighbours: 763 864 977 993 1014 1022 941 490 1019 1021	526872 523776	0:29:57
B-hc10.rmf	1023-Neighbours: 744 938 971 994 1019 1021 1022 498 895 1015	526690 523776	0:30:41
B-mesh33x33.rmf	1088-Neighbours: 1044 1069 1071	37604 35904	0:31:47
B-mesh33x33.rmf	1088-Neighbours: 1063 1087 1047	37046 35904	0:32:39
B-mesh33x33.rmf	1088-Neighbours: 1046 1087	36965 35904	0:33:11
C-3elt.rmf	4719-Neighbours: 4212 4227 4302 4347 4353 4376 4718	1041221 1060932	0:33:55
C-3elt.rmf	4719-Neighbours: 4212 4227 4302 4347 4353 4376 4718	1036857 1060932	0:34:21
C-3elt.rmf	4719-Neighbours: 4216 4370 4416 4423 4718 4323 4204	1020024 1060932	0:34:51
C-airfoil1.rmf	4252-Neighbours: 4220 4247 4225 4208	405028 407921	0:35:38
C-airfoil1.rmf	4252-Neighbours: 4231 4237 4228 4246 4248	405554 407921	0:36:08
C-airfoil1.rmf	4252-Neighbours: 4248 4250 4235 4242	405274 407921	0:36:46
C-whitaker3.rmf	9799-Neighbours: 6181 6183 6645 9724 9796 9798	8960636 9029276	0:41:29
C-whitaker3.rmf	9799-Neighbours: 6181 6183 6645 9724 9796 9798	8958487 9029276	0:41:59
C-whitaker3.rmf	9799-Neighbours: 6181 6183 6645 9724 9796 9798	8983738 9029276	0:43:08
D-c1y.rmf	827-Neighbours: 767 810 811 486	211284 369905	0:44:10
D-c1y.rmf	827-Neighbours: 421 739 825 514 539	216311 369905	0:44:37
D-c1y.rmf	827-Neighbours: 746 130 498	209842 369905	0:45:14
D-c2y.rmf	979-Neighbours: 489	309228 517737	0:46:16

D-c2y.rmf	979-Neighbours: 977 487 522	326946 517737	0:46:45
D-c2y.rmf	979-Neighbours: 516	313561 517737	0:51:42

D-c3y.rmf	1326-Neighbours: 269 270 946 1090	530206 787049	0:52:57
D-c3y.rmf	1326-Neighbours: 921	517103 787049	0:53:24
D-c3y.rmf	1326-Neighbours: 1148 1214	526721 787049	0:53:48
D-c4y.rmf	1365-Neighbours: 760	595879 919089	0:54:38
D-c4y.rmf	1365-Neighbours: 866	570118 919089	0:55:12
D-c4y.rmf	1365-Neighbours: 982 1242 765 1351	559476 919089	0:55:36
D-c5y.rmf	1201-Neighbours: 619 838 899 1146 624	452188 743485	0:5644
D-c5y.rmf	1201-Neighbours: 750 1084 1130	462342 743485	0:57:36
D-c5y.rmf	1201-Neighbours: 649 675 953 637	455839 743485	0:58:05
gd95c.rmf	61-Neighbours: 54 56 55	736 990	0:59:06
gd95c.rmf	61-Neighbours: 57 58 60 56 55	672 990	0:59:048

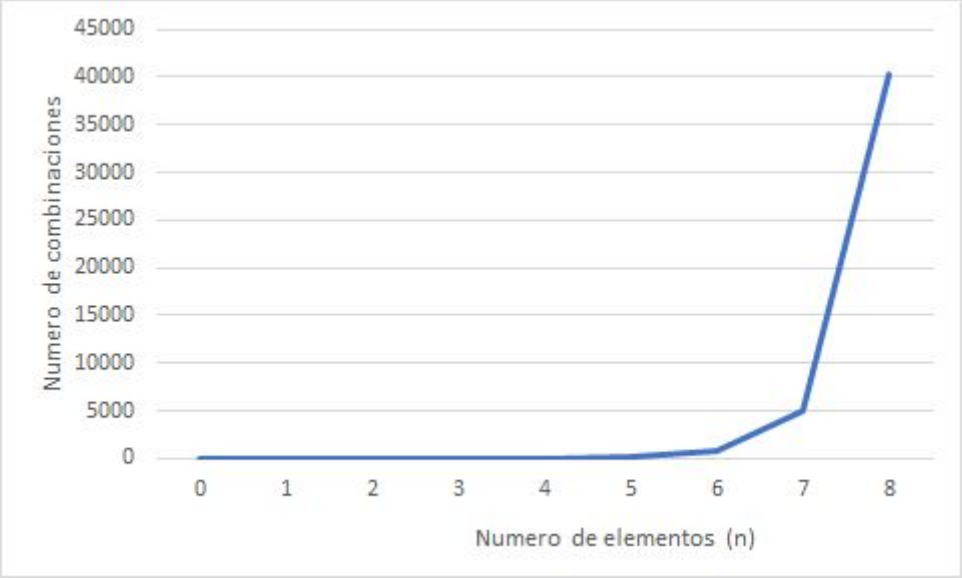
Ejemplo

El problema combinatorio que decidimos abarcar se refiere al VLSI (Very-large-scale integration), que es el proceso de crear un circuito integrado usando miles de transistores dentro de un chip. Esta práctica fue muy usada aproximadamente en los años 70, donde los semiconductores y las tecnologías más complejas aún se estaban desarrollando.



Visualización del proceso de VLSI

Como su nombre lo indica, al haber una gran cantidad de interconexiones entre módulos, cierta cantidad de cable debía de ser usada para llevar a cabo dichas conexiones. Es justo aquí donde podemos ligar el algoritmo MinLA con VLSI, para lograrlo, debemos de representar los módulos del circuito integrado como nodos de nuestro grafo, y por consiguiente las conexiones entre dichos módulos se traduce como vértices del grafo.



Número de elementos (X-Axis) VS Número de combinaciones (Y-Axis)

Como se puede apreciar en la gráfica anterior, el tamaño de busqueda del problema MinLA viene dado con la expresión matemática N! para un grafo con n elementos. Esto se debe a que en cualquiera de las variaciones del grafo, puedes hacer un movimiento y llegar a otro. Debido a esta naturaleza, el problema llega a escalar muy rápido con tan solo unos poco elementos, como se puede apreciar en la anteriormente mencionada gráfica, en la que con tan solo 8 elementos tenemos ya más de 40,000 combinaciones totalmente únicas en las que podemos arreglar nuestro grafo.

¿Esto que causa en nuestro problema? Pues muy sencillo, la respuesta es que es casi imposible generar un mecanismo que sea capaz de crear soluciones óptimas, es decir, puede que existan, pero el tiempo computacional que llevaría ejecutarlas sería infinito, debido a que se deben de explorar todas las posibilidades.

A continuación, nos gustaría presentar la instancia escogida para la resolución del problema. Primero tenemos la instancia original, siendo el primer dígito la etiqueta del nodo que le corresponde a cada módulo , y luego de eso se presentan los vecinos de dicho módulo, que son aquellos a los que está conectado.

Después de eso, se presenta la instancia modificada con el mismo formato, y justo debajo de ella tenemos la sumatoria de energía de la instancia original y la energía de la instancia ya solucionada, para contar con una rápida comparación.

Instancia original:

0-Neighbours: 7 18 31
1-Neighbours: 36 39 49 64
2-Neighbours: 10 25 30 56
3-Neighbours: 16 24 59 61
4-Neighbours: 12 22 30 33 57
5-Neighbours: 16 27
6-Neighbours: 7 15 18 41
7-Neighbours: 0 6 24 61
8-Neighbours: 9 13 16 27 59
9-Neighbours: 8 34 53
10-Neighbours: 2 46 56 63
11-Neighbours: 22 25 42 55
12-Neighbours: 4 22 28 62
13-Neighbours: 8 21 34 45 58
14-Neighbours: 19 33 42 52 60
15-Neighbours: 6 35
16-Neighbours: 3 5 8 21
17-Neighbours: 18 23 31 41 51
18-Neighbours: 0 6 17 35 37
19-Neighbours: 14 46 52 55
20-Neighbours: 34 45
21-Neighbours: 13 16 47 61
22-Neighbours: 4 11 12 42
23-Neighbours: 17 37 40
24-Neighbours: 3 7
25-Neighbours: 2 11 30 55
26-Neighbours: 36 43 60 62
27-Neighbours: 5 8 53
28-Neighbours: 12 32 44 62 64
29-Neighbours: 31 45 51 58
30-Neighbours: 2 4 25 57
31-Neighbours: 0 17 29 47 61
32-Neighbours: 28 36 48
33-Neighbours: 4 14 44 53
34-Neighbours: 9 13 20
35-Neighbours: 15 18 48
36-Neighbours: 1 26 32 43 64
37-Neighbours: 18 23 48
38-Neighbours: 43 49 52 63
39-Neighbours: 1 49 50 56
40-Neighbours: 23 51
41-Neighbours: 6 17 54 61
42-Neighbours: 11 14 22 60
43-Neighbours: 26 36 38 52
44-Neighbours: 28 33 50 57 64
45-Neighbours: 13 20 29 47
46-Neighbours: 10 19 55 63
47-Neighbours: 21 31 45
48-Neighbours: 32 35 37
49-Neighbours: 1 38 39 63
50-Neighbours: 39 44 56 57
51-Neighbours: 17 29 40 54
52-Neighbours: 14 19 38 43
53-Neighbours: 9 27 33
54-Neighbours: 41 51 58
55-Neighbours: 11 19 25 46
56-Neighbours: 2 10 39 50
57-Neighbours: 4 30 44 50

58-Neighbours: 13 29 54 59 61
59-Neighbours: 3 8 58
60-Neighbours: 14 26 42 62
61-Neighbours: 3 7 21 31 41 58
62-Neighbours: 12 26 28 60
63-Neighbours: 10 38 46 49
64-Neighbours: 1 28 36 44

Instancia solucionada:

0-Neighbours: 1 2
1-Neighbours: 28 0 4
2-Neighbours: 3 0 4 5
3-Neighbours: 6 7 2
4-Neighbours: 1 6 20 2 8
5-Neighbours: 7 15 2 8
6-Neighbours: 3 9 17 4
7-Neighbours: 3 9 24 5 19
8-Neighbours: 4 9 10 5 12
9-Neighbours: 6 7 11 14 16 8
10-Neighbours: 11 20 8
11-Neighbours: 9 13 10 17
12-Neighbours: 14 15 8
13-Neighbours: 11 16
14-Neighbours: 9 18 12 19
15-Neighbours: 22 5 12 19
16-Neighbours: 9 13 18 24
17-Neighbours: 6 11 62 20
18-Neighbours: 16 21 14 60
19-Neighbours: 21 14 7 15 27
20-Neighbours: 28 63 10 17 4
21-Neighbours: 18 64 24 23 19
22-Neighbours: 15 27
23-Neighbours: 21 58 27
24-Neighbours: 16 21 7
25-Neighbours: 26 30 29
26-Neighbours: 28 63 25
27-Neighbours: 22 23 19
28-Neighbours: 20 26 1
29-Neighbours: 25 33 53 56 31
30-Neighbours: 25 32 54 55 31
31-Neighbours: 32 33 30 29
32-Neighbours: 30 34 36 57 31
33-Neighbours: 29 35 38 31
34-Neighbours: 37 35 36 32
35-Neighbours: 33 34 37 38
36-Neighbours: 34 42 47 32
37-Neighbours: 34 35 39 40
38-Neighbours: 33 35 41 59
39-Neighbours: 37 40 42 44
40-Neighbours: 37 39 41 43
41-Neighbours: 40 43 38 59
42-Neighbours: 44 47 39 36
43-Neighbours: 40 41 45 50
44-Neighbours: 45 46 42 39
45-Neighbours: 43 44 46 50
46-Neighbours: 45 44 48 49
47-Neighbours: 42 55 36 48 57
48-Neighbours: 46 47 55 49
49-Neighbours: 46 51 52 48
50-Neighbours: 43 45 51 61
51-Neighbours: 49 57 61 50 52
52-Neighbours: 49 51 53 54
53-Neighbours: 29 52 54 56
54-Neighbours: 53 55 52 30
55-Neighbours: 47 54 48 30
56-Neighbours: 53 61 59 29
57-Neighbours: 32 47 51 58
58-Neighbours: 57 64 23
59-Neighbours: 38 41 56 61
60-Neighbours: 18 64
61-Neighbours: 51 56 50 59
62-Neighbours: 63 17
63-Neighbours: 62 20 26
64-Neighbours: 21 60 58

Vieja energía:2665
Nueva energía:862

Como podrá ver, la energía se reduce en más del 70%, y aunque estamos seguros de que no es la solución óptima, si presenta una reducción bastante considerable y con un tiempo de ejecución bastante bajo.

Aportaciones

Gerardo	● Esqueleto base de la metaheurística ● La definición de la metaheurística de las primeras entregas ● Estefanía y yo colaboramos para el sistema de grafos ● Hice el ejemplo práctico el reporte acerca de VLSI ● El análisis del costo de la solución vs el número de iteraciones ● La tabla de la fórmula matemática de búsqueda

	espacial
Estefania	● Definición del problema ● Introducción del proyecto ● Ejemplos ilustrativos del problema ● Colaboración en el sistema de grafos ● Experimentación con el algoritmos desarrollado ● Análisis matemático del algoritmo ● Explicación y comentarios de la clase grafo

Aprendido durante el curso

Gerardo	Creo que si listo todo lo que aprendí en el curso, esta parte del proyecto se haría muy larga, así que mejor voy a decirle acerca de mi tema preferido. El tema que elegiría para esa categoría sería el último tema que vimos, el de cómputo paralelo. Es un tema con muchísimo que cubrir y también era uno que no tenía mucho en cuenta porque carecía de conocimientos para pensar en ello.
Estefania	En lo personal a mi me interesa mucho la área de machine learning y lo que más aprendí del curso fue el hecho de investigar más a fondo muchos temas que el profesor nos dio y buscar usos en áreas que a mi me interesan lo cual hace que descubra sobre más temas. Un ejemplo fue geometric deep learning.

Lo que nos gustó y no nos gustó del curso

	Lo que nos gustó	Lo que no nos gustó
Gerardo	Me gustó muchísimo que usted impartiera la clase, tiene muchísimas cualidades buenas como profesor, pero quizás la mejor es que de verdad se preocupa porque sus alumnos aprendan. Si del curso se trata, creo que se me hizo muy bueno el aprendizaje que tuvimos, con temas que no estaba excesivamente complicados una vez que los entiendas	Solo hay una cosa que no me gustó, si es que le puedo llamar a sí. Creo que la curva inicial del curso es muy alta desde el principio. En esos momentos yo me sentía perdido y no entendía hasta que me puse al corriente con mis compañeros, pero creo que mi queja más bien es personal, porque hay mucha gente que dijo lo contrario.
Estefania	El curso está muy bien explicado y al principio lo que más ayudaba a entender los temas fue las tareas. Aparte aunque considero que es una materia pesada por que hay muchos temas que aprender siento que el hecho de saber que lo que importa es aprender y no la calificación es algo muy importante de saber y el profesor hizo muy buen énfasis en eso.	Al principio sentía que no entendía lo que estaba viendo y que me hacia falta mas conocimiento previo para tomar el curso.

Conclusión

En este proyecto aprendimos a cómo aplicar un algoritmo metaheurístico para la solución del problem MinLa. Usamos los conocimientos previos que adquirimos en el transcurso de la materia poniéndolos en práctica, lo cual nos ayudó a reforzar estos conocimientos.

También aprendimos a buscar siempre la manera que nuestro algoritmo sea más óptimo y como esto incluso si son pocos datos afecta demasiado al momento de hacer testing a tu codigo ya que esto afecta la memoria de tu computadora.
Por último fue interesante poder investigar sobre el uso de MinLa y buscar ejemplos de lo que se puede usar. Este proyecto nos ayudó a adquirir más conocimiento de los grafos y conocimientos que por mas básicos que pensaba que era saber que hay mucho más por aprender.

Referencias

Eduardo Rodriguez-Tello, Jin-Kao Haoa, Jose Torres-Jimenez. (2007). An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. 1 de Junio del 2020, de Science Sitio web: https://www.sciencedirect.com/science/article/abs/pii/S0305054807000676