Análisis de Balance de Flujos Mediante Recocido Simulado

Juan Antonio López Rivera

June 8, 2018

1 Introducción

Una de las maneras más obvias de modelar una red de reacciones químicas es mediante una gráfica dirigida, donde los nodos son reactivos y son adyacentes si la sustancia origen puede usarse para producir la destino. Sin embargo, para muchas aplicaciones éste modelo es insuficiente[1]. Uno de sus problemas principales es la dificultad de operar datos estoiquiometricos (la estoiquiometría es el estudio de las proporciones de cada reactivo que interviene en una reacción; por ejemplo, que 4A + 6B -> 2C).El Análisis de Balance de Flujos (Flux Balance Analysis o FBA) provee un modelo matemático (principalmente la matriz de estoiquiometría) que nos permite analizar el flujo de todos los metabolitos (reactivos involucrados en reacciones metabólicas) en una red metabólica.

Usualmente los problemas de FBA son resueltos mediante programación lineal[2]. Sin embargo, éste método tiene la gran desventaja de que su complejidad en tiempo es de orden exponencial en el peor caso[3]. Conforme profundizamos en el estudio de sistemas biológicos, la cantidad y el tamaño de los datos a operar crece, así que la necesidad de nuevas técnicas para resolver problemas de FBA es inminente.

En éste trabajo aproximamos soluciones a un modelo concreto (una simplificación de la red metabólica de la bacteria *Escherichia Coli*) usando la heurística del recocido simulado[4]. En la sección 2 se explica brevemente cómo funciona el FBA y como se ocupa programación lineal para resolver un modelo. La sección 3 detalla el método alternativo propuesto que ocupa la heurística de aceptación por umbrales o recocido simulado. La sección 4 corresponde a la implementación en el lenguaje Julia de un programa que aproxima soluciones al modelo E Coli Core[5], así como una discusión de los resultados obtenidos. Finalmente la sección 6 enuncia algunas conclusiones e ideas para seguir ésta linea de investigación.

2 El Análisis de Balance de Flujos (FBA)

2.1 La matriz de estoiquiometría

Una matriz de estoiquiometría S es una matriz de M por R, donde M es el total de reactivos en un modelo y R el número de reacciones – cada renglón corresponde a un reactivo y cada columna a una reacción. Si un reactivo no contribuye en una reacción el elemento correspondiente tiene un valor de 0, de lo contrario hay dos casos. Si se consume el reactivo (aparece del lado izquierdo de una reacción) el valor estoiquiométrico es negativo, de lo contrario (si es un producto de la reacción) el valor es positivo.

Por ejemplo consideremos un modelo con 2 reacciones, R1 es 2 A + B -> 3 C, y R2 es B + 3 D -> 4 A. La matriz de estoiquiometría correspondiente es:

	R1	R2
A	-2	4
B	-1	-1
C	3	0
D	0	-3

La matriz de estoiquiometría nos permite determinar fácilmente los reactantes y productos de cualquier reacción dada, incluyendo sus valores estoiquiométricos, así como las reacciones de las cual una sustancia es parte de, y si es reactante o producto en cada reacción. Además, y éste es el punto más importante, podemos hacer operaciones y álgebra lineal con ella.

2.2 Las restricciones

Diversas propiedades físicas, biológicas y químicas de las sustancias involucradas en las reacciones nos indican cotas ínfimas y superiores para las sustancias existentes en nuestro modelo. Por ejemplo por principios de kinética Michaelis-Menten[6] sabemos que el flujo a través de una enzima dada tiene un valor máximo posible finito. Otra es que las reacciones que son irreversibles necesariamente tienen un flujo mayor o igual a cero.

Éstas cotas son cruciales para el modelo de FBA, ya que restringen en gran medida el tamaño del espacio de búsqueda que deberemos explorar para hallar soluciones.

La otra restricción importante que puede tener un modelo de FBA es el estado de equilibrio. En términos matemáticos, ésto quiere decir que un vector de flujos v es factible syss

$$Sv = 0$$

Es posible estudiar sistemas que tengan un estado inestable de flujos mediante Análisis Dinámico de Balance de Flujos (Dynamic Flux Balance Analysis).

2.3 La función objetivo

Dada una matriz S de dimensiones M x R, una función objetivo f es una función que recibe un vector v de longitud R y devuelve un escalar. De ésta manera definimos matemáticamente la meta de un modelo de programación lineal, es

$$\max_v f(v)$$

Un ejemplo de una función objetivo sencilla es una que para cualquier vector $v = (v_1, v_2, ..., v_n)$ devuelva el valor de v_i, donde i es constante. Ésto representa que queremos maximizar el producto de la reacción i, que podría corresponder a algo como la producción de ATP (para hacer que nuestro modelo busque la máxima energía total generada por el sistema).

2.4 Solución usando Programación Lineal

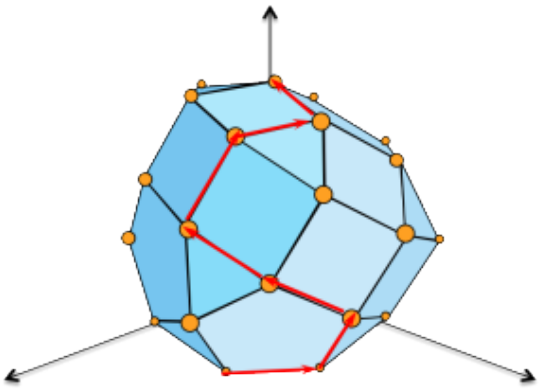

Figure 1: Politopo tridimensional con camino entre vértices.

Consideremos la figura 1. Representa un politopo acotado tridimensional. Supongamos que éste es el espacio de soluciones posibles a un conjunto de igualdades lineales y cotas (que corresponden a ciertas ecuaciones de balance de flujos y restricciones de capacidad dadas). Cada punto dentro y en la frontera del politopo satisface éstas condiciones, pero algunas soluciones son mejores que otras. Más aún, existe una o un conjunto de soluciones "óptimas", bajo algún criterio (dado por la función objetivo).

Existen diversos algoritmos para resolver problemas de programación lineal, pero todos los algoritmos se componen de dos partes: hallar una solución inicial factible, y luego a partir de ella recorrer el espacio de soluciones para hallar la que maximiza la función objetivo.

Uno de los más usados es el *método del simplex*[7], que a grandes rasgos funciona de la siguiente manera. Se inicia a partir de cualquier vértice del politopo, mediante operaciones de 'pivote' se

recorre el politopo de vértice a vértice, moviéndonos siempre hacia uno cuya evaluación en la función objetivo sea mejor que la del actual (la figura 1 en rojo muestra un posible camino escogido por el algoritmo de Simplex). Las matemáticas involucradas para hacer ésto posible son mucho, mucho más complicadas que ésta explicación (y precisamente es una de las motivaciones principales de éste proyecto, proponer un método para resolver estos problemas que sea relativamente más facil de entender e implementar).

2.5 Algunas aplicaciones

Mediante diversas funciones objetivo se puede usar un modelo de FBA para estudiar diversos sistemas; por ejemplo, si elegimos una función que maximice la producción de ATP, podemos predecir la energía máxima que puede fluir a través del sistema[8]. Luego, se pueden agregar o quitar reacciones y metabolitos del modelo (por ejemplo unas que estén presentes en especies similares), con el fin de hallar un sistema con un mayor flujo máximo. Ésto funge como modelo teórico para guiar investigaciones en ingeniería metabólica[9].

Otra aplicación de ésta técnica es en genética. Es posible reflejar en un modelo de FBA los cambios que surgen en una red metabólica al eliminar genes[10], siendo así de utilidad también en la elaboración de transgénicos y modificaciones genéticas de organismos.

3 La heurística del Recocido Simulado para FBA

La heurística de aceptación por umbrales o el recocido simulado[4] (llamada así porque halló su inspiración en metalurgia de la edad media) es en realidad bastante similar en principios a la mayoría de las heurísticas inspiradas por fenómenos naturales que plagan actualmente el área de optimización combinatoria – la idea principal es recorrer un espacio de búsqueda semialeatoriamente, refinando a ratos nuestra búsqueda (escribí una explicación mucho más detallada de la heurística en el reporte del primer proyecto[11]). Para ocupar ésta heurística necesitamos definir dos aspectos muy importantes: ¿cuándo una solución es mejor que otra? y ¿a partir de una solución, como generamos otra diferente?

Pero primeramente, definamos en concreto el problema a resolver. Un modelo de FBA consta de una matriz de estoiquiometría S, para la cual queremos hallar soluciones ie. vectores de flujo v que maximizen una función objetivo f, ie. $\max_v f(v)$. Opcionalmente el modelo incluye restricciones, que pueden ser cotas inferiores y/o superiores para cada elemento v_i de v, o una condición de equilibrio ($Sv = 0$).

La búsqueda de tal solución constará de dos partes, una es hallar una v que cumpla ambas restricciones, y después hay que hallar una que maximice $f(v)$.

3.1 Solución Inicial

La manera más fácil de superar la restricción del estado de equilibrio es darnos cuenta que los vectores v que cumplen que $Sv = 0$ son exactamente el espacio nulo o núcleo de S. Basta con hallar una base de $ker(S)$, digámosle B, y todas las soluciones las generaremos como una combinación lineal de elementos de B. La solución incial es simplemente una combinación lineal aleatoria de vectores de B. Para la factibilidad nada más falta cumplir las cotas inferiores y superiores de elementos de v, pero de ésto se encargará la función de costo.

Alternativamente, para soluciones iniciales podemos generar vectores que cumplan las cotas (simplemente escogemos aleatoriamente un número entre las cotas definidas, para cada elemento de v) y tratar de cumplir la condición de equilibrio usando la función de costo. Sin embargo éste enfoque resultó ser mucho más difícil que el otro; no es claro como definir una función de costo que ayude a las soluciones a acercarse a cumplir $Sv = 0$, y los experimentos realizados tratando de incorporar la distancia absoluta al cero de los elementos de Sv en la función de costo fueron infructuosos.

Me parece que aquí yace una de las mayores oportunidades para mejorar éste metodo que se propone: podrían usarse fragmentos de algoritmos de programación lineal para hallar mejores soluciones iniciales. Por ejemplo, utilizar los métodos ocupados por el algoritmo del simplex para hallar una solución inicial dentro del politopo de soluciones factibles, en lugar de hallar una dentro del espacio nulo y encontrar el politopo mediante recocido simulado como hacemos aquí. Sin embargo, incluso los algoritmos más sencillos para hallar el politopo inicial requieren un conocimiento de

algebra lineal mucho mayor del que tiene el autor. Además, la implementación de tal programa cae fuera del alcance del curso para el cual se desarrolló éste método (requeriría principalmente conocimientos fuertes de programación lineal, cuando éste curso es sobre heurísticas).

3.2 Función de Costos

La función de costo que escogamos definirá cómo se explorará el espacio de búsqueda. Inicialmente se intentó una función de costo que solamente busque maximizar la función objetivo dada. Sin embargo, las soluciones que genera ésta elección son muy lejanas a ser factibles, ya que la heurística solo buscaba maximizar el peso dado sin considerar en lo absoluto las restricciones. Irónicamente, las restricciones que fueron incorporadas a los modelos de FBA para acotar el espacio de búsqueda y facilitar la labor de los algoritmos de programación lineal, resultaron ser el obstáculo más difícil de superar usando Aceptación por Umbrales.

El programa final usa una función de costos por casos, si el número de cotas que viola v es mayor a 0 entonces se ocupa un costo para minimizar éste número, de lo contrario entonces se intenta maximizar la función objetivo dada (teniendo en cuenta que todas las soluciones que violen 0 cotas siempre deben evaluarse mejor que cualquiera que viole 1 o más cotas).

Concretamente, definimos el costo de una solución v como

$$costo(v) = \begin{cases} 0 - (p_1 * (\sum_{i=1}^{k} |v_i|) + p_2 * c, & si\ c > 0 \\ p_3 + f(v), & si\ c = 0 \end{cases}$$

donde p_1, p_2 y p_3 son constantes que sirven como castigos, y c es el número de elementos de v que violan alguna cota. k es la longitud del vector v.

Es importante escoger los castigos cuidadosamente, depende de cada modelo, pero hay que preservar siempre que las soluciones que caigan en el segundo caso siempre se evalúen mejor que las primeras. En el caso de que la función objetivo sea simplemente el valor de algún elemento v_i de v (como es el caso del ejemplo trabajado en la sección 4) ésto se logra simplemente escogiendo un $p_2 > a_i$, donde a_i es la cota inferior para valores de v_i.

En la figura 2 podemos ver un ejemplo de un modelo de programación lineal en 2 dimensiones con 3 restricciones. Los puntos A, B, C, y D son soluciones dadas. Como el punto B se encuentra debajo de la recta verde, cumple la desigualdad que representa y se evalúa mucho mejor que el punto A y el C. Sin embargo, como el C está más cerca que el A de cruzar las rectas verde y azul, se evalúa mejor por la función de costo. El D, estándo completamente dentro del polígono de factibilidad, se evalúa mejor que todos los demás.

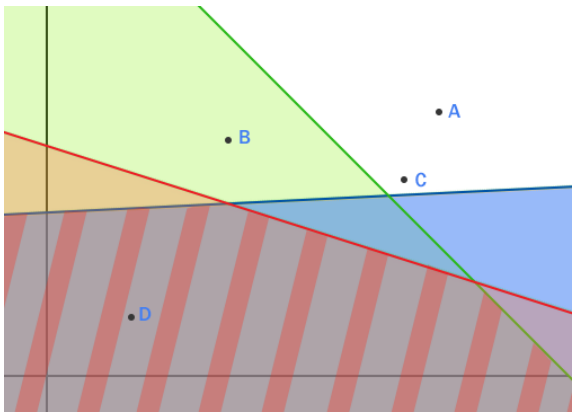

Figure 2: Un modelo bidimensional con 3 restricciones. Las rectas de colores representan las restricciones marcadas por desigualdades, y la región sombreada es el polígono de soluciones factibles.

3.3 Generación de Vecinos

Recordemos que la solución inicial la generamos a partir de una combinación lineal de vectores de una base B del espacio nulo de S. Un vecino de una solución $v = s_1\ b_1 + s_2\ b_2 + ... + s_n\ b_n$ consistirá

en modificar ligeramente el valor de alguna s_i (agregaremos o restaremos un valor predefinido a algún s_i escogido aleatoriamente).

Aquí recae la otra posible mejora grande al método propuesto en éste trabajo. Halladas soluciones iniciales dentro del politopo de factibilidad, mínimo habría que encontrar vecinos que no se salgan del politopo, pero podría incluso hacerse uso de herramientas de programación lineal – por ejemplo las del método del simplex, empezar desde una pared y moverse solamente sobre la frontera del espacio de soluciones. Nuevamente, me parece que tal programa cae fuera de los objetivos del curso.

3.4 Ejemplo de funcionamiento

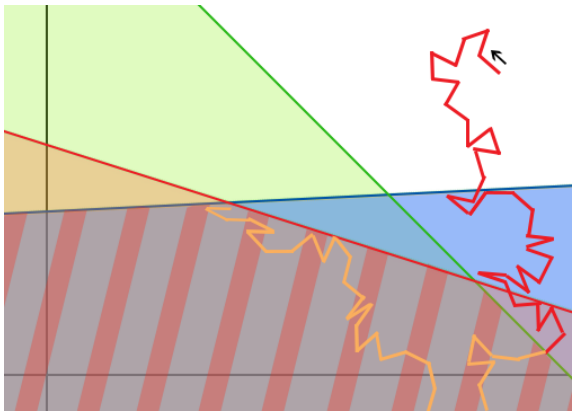

Figure 3: Un ejemplo de un posible camino que puede tomar una ejecución de la heurística. Nótese como una vez que se cruza una recta hacia una región "mejor", no vuelve a salir de ésta. Ésto se logra teniendo un castigo p_2 que es ordenes de magnitud más alto que p_1. Así, el recocido logra dos cosas simultáneamente, prohibe la salida de una región a menos que sea mejor que la actual (de ésto se encarga el p_2 alto), y (gracias a la suma asociada a p_1) nos acerca a las fronteras de regiones mejores. Una vez se entra al polígono sombreado, la función de costo empieza a tratar de maximizar la función objetivo (en este caso es maximizar el valor sobre el eje y).

4 Implementación

Se trabajó sobre el modelo Ecoli Core iAF1260[2], el cual es una simplificación de la red metabólica de la bacteria *Escherichia Coli*. Contiene una matriz de estoiquiometría de 73 renglones (metabolitos) por 96 columnas (reacciones), todas con meta información acerca del tipo de reacción y las sustancias y demás. Además, para cada metabolito incluye cotas superiores e inferiores. Finalmente, propone una función objetivo simbolizada por un vector c que representa la biomasa total del sistema.

El sistema fue escrito para Julia, un lenguaje de programación diseñado para cómputo científico. La elección de lenguaje fue por dos motivos, primeramente Julia cuenta con una cantidad respetable de funciones de álgebra lineal (por ejemplo, el espacio nulo de S lo hallamos sencillamente usando la función null(S) del lenguaje). Por otro lado, Julia ya habia probado su utilidad (y velocidad) en el primer proyecto[11], que también fue sobre recocido simulado, para el recocido se pudo reutilizar secciones de código y usar una estructura general bastante similar.

Los insumos fueron descargados en un sólo archivo de formato xls con varias páginas, para facilitar su lectura programáticamente fueron separados en archivos csv, uno para la matriz estoiquiométrica ("ecoli_core_model.csv"), otro para las restricciones ("constraints.csv") y finalmente el vector que representa la función objetivo ("objective.csv"). Éstos archivos se encuentran dentro de la carpeta "sources" del proyecto. Las funciones que leen los archivos de insumos y guardan los datos en objetos de Julia se encuentran en el archivo "metabolic_network.jl".

Los métodos de generación de soluciones fueron implementados como descritos en la sección 3.1, las funciones correspondientes van en el archivo "solutions.jl". Asimismo, la función de costo y algunas otras más se encuentran en "costs.jl".

Cabe recalcar que cada palabra del programa la escribí yo; el proyecto está en inglés porque ~~soy un niñato pretencioso~~ quería algo de variedad al escribir el proyecto, y se va a quedar en mi github hasta el final de los siglos y probablemente a la mayoría de la gente que lo podría leer le convenga el inglés.

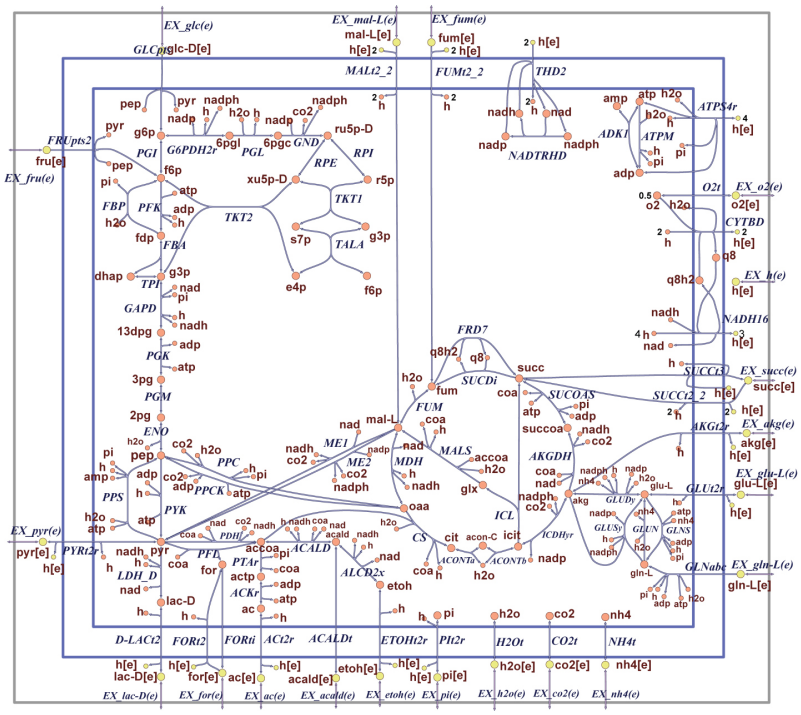

Figure 4: Diagrama de reacciones del modelo EColi Core usado.

El recocido simulado ("simulated_annealing.jl") corre ocupando los objetos de insumos, la función de costo y de vecinos, así como la configuración dada para temperatura inicial/final, velocidad de enfriamiento, tamaño de lote, etc. especificado en un archivo de configuración.

El ciclo principal del programa se halla en "run_heuristic.jl". Éste parsea todos los archivos necesarios (insumos así como la configuración especificada), y corre el número de veces que se pidió, generando una semilla aleatoria cada vez a partir de una semilla maestra dada. Para cada corrida, va imprimiendo en pantalla la temperatura para monitorear el progreso; al terminar cada una, imprime en pantalla y guarda en un archivo (dentro de la carpeta "results", con el nombre de la semilla maestra usada) los resultados la corrida. Al finalizar todas las corridas, imprime en pantalla y guarda en el archivo el mejor resultado obtenido.

Para ejecutar el programa con un archivo de configuración "settings.txt", desde la linea de comandos hay que escribir

```
> julia main.jl settings.txt
```

El archivo "main.jl" simplemente manda a llamar al módulo Run definido en "run_heuristic.jl" con el nombre de archivo de configuración especificado como parámetro. Un ejemplo de archivo de configuración válido se encuentra dentro de la carpeta "settings".

Los resultados obtenidos con éste sistema se compararon con el resultado óptimo hallado mediante programación lineal. Para ésto se utilizó el paquete COBRA (COnstraint Based Reconstruction and Analysis Toolbox)[12], escrito para Matlab pero muy afortunadamente está disponible como paquete de Julia. Ocupa un archivo fuente distinto (necesita un .mat nativo de Matlab) pero contiene exactamente la misma información que el .xls. El código correspondiente está en el archivo "lp.jl". El resultado 'óptimo' hallado con éste archivo se guardó en "results/lp_solution.txt".

4.1 Resultados

Se obtuvieron soluciones bastante cercanas a la óptima predecida con programación lineal. La mejor solución hallada (con semilla 4453425) no viola ninguna restricción, cumple la condición de

equilibrio ($Sv = 0$), y tiene un valor de biomasa (su función objetivo) de -0.1353957, mientras que la óptima que hallamos con programación lineal es de 0.873921. Considerando que puede tomar valores entre -1000 y 1000, y la imprecisión inherente al ocupar una heurística, diría que es bastante bueno. Se puede replicar la corrida que lo halló usando el archivo de configuración "settings_best.txt" que se encuentra en la carpeta "settings" del proyecto:

```
> julia main.jl settings/settings_best.txt
```

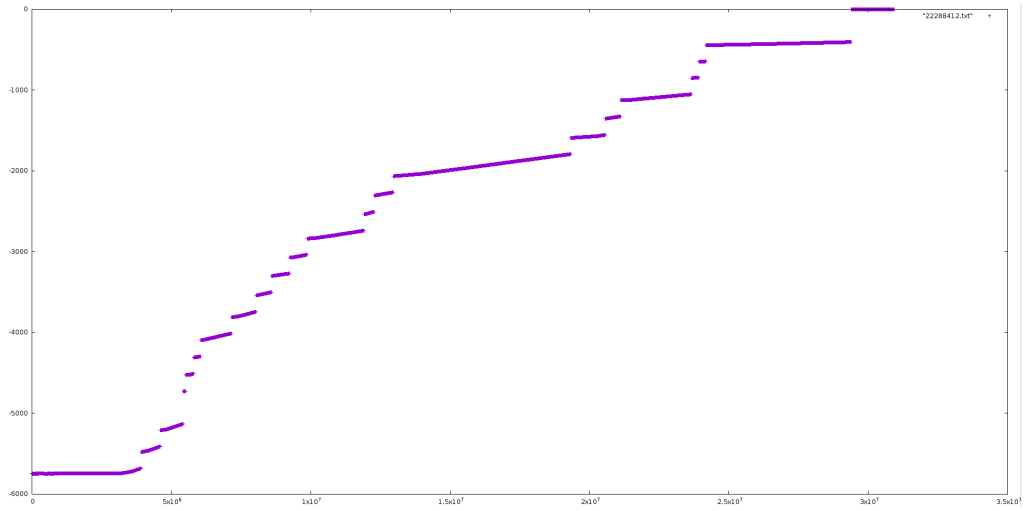

Figure 5: Serie de tiempo del costo de las soluciones aceptadas a lo largo de la una corrida.

Es interesante la forma de la gráfica de costos aceptados (Figura 5) de una corrida. Las discontinuidades ocurren cuando se cruza a una región de factibilidad, es decir se deja de violar alguna cota inferior o superior. Ésta gráfica corresponde a una corrida que quedó a 1 restricción violada de ser factible. La gráfica de la mejor semilla es similar pero al final da un salto enorme, que hace difícil ver los detalles. Éste salto ocurre por el valor de p_3 ocupado (1000), necesario porque debemos garantizar que las soluciones que violan 0 restricciones siempre se evaluen mucho mejor que las que no.

El vector de la mejor solución hallada se guardó en "results/best_solution.txt"

5 Conclusiones y Trabajo Futuro

Éste método tiene la gran desventaja de su tiempo de ejecución. Para hallar soluciones decentes es necesario tener valores de temperatura inicial, final, y factor de enfriamiento algo extremos, que alentecen bastante la ejecución de la heurística.

Otro problema es la imprecisión de los resultados; depende del tamaño de paso entre una solución y otra, conforme mas pequeño sea, más finamente recorremos el espacio de búsqueda y mejoran las soluciones posibles, pero aumenta en ordenes de magnitud el tamaño del espacio de búsqueda, así que – como suele pasar al usar ésta heurística – se deben hallar valores que nos den un balance entre buenas soluciones y tiempo de ejecución. Una consideración empleada es dar un rango de imprecisión aceptable – un elemento v_i de la solución v no violará la cota inferior si $v_i > cota\_inf_i - precision$, y no violará la superior si $v_i < cota\_sup_i + precision$. Se lograron resultados con precisión de hasta 4 decimales; una precisión mayor probablemente requeriría un tamaño de paso entre vecinos mucho menor (que de por sí es pequeño, para las soluciones halladas se ocupó un tamaño de paso de 0.00005).

Nuevamente, me parece que las mayores mejoras posibles sobre ésta linea de investigación serían mediante un algoritmo híbrido que ocupe métodos de programación lineal más avanzados. Para fines de éste curso me parece que los resultados obtenidos, así como el sistema programado y el reporte escrito son más que suficientes.

References

[1] Sandefur, C. I., Mincheva, M., & Schnell, S. (2013). Network representations and methods for the analysis of chemical and biochemical pathways. Molecular bioSystems, 9(9), 2189–2200. http://doi.org/10.1039/c3mb70052f

[2] Orth, J. D., Thiele, I., & Palsson, B. Ø. (2010). What is flux balance analysis? Nature Biotechnology, 28(3), 245–248. http://doi.org/10.1038/nbt.1614

[3] Raman, K., & Chandra, N. (2009). Flux balance analysis of biological systems: applications and challenges. Briefings in bioinformatics, 10(4), 435-449.

[4] Van Laarhoven, P. J., & Aarts, E. H. (1987). Simulated annealing. In Simulated annealing: Theory and applications (pp. 7-15). Springer, Dordrecht.

[5] EcoSal Chapter 10.2.1 - Reconstruction and Use of Microbial Metabolic Networks: the Core Escherichia coli Metabolic Model as an Educational Guide by Orth, Fleming, and Palsson (2010) http://gcrg.ucsd.edu/Downloads/EcoliCore

[6] Ainsworth, S. (1977). Michaelis-menten kinetics. In Steady-state enzyme kinetics (pp. 43-73). Palgrave, London.

[7] Dantzig, George (May 1987). "Origins of the simplex method". A history of scientific computing. doi:10.1145/87252.88081. ISBN 0-201-50814-1.

[8] Harcombe, W. R., Delaney, N. F., Leiby, N., Klitgord, N., & Marx, C. J. (2013). The ability of flux balance analysis to predict evolution of central metabolism scales with the initial distance to the optimum. PLoS computational biology, 9(6), e1003091.

[9] Schwender, J. (2008). Metabolic flux analysis as a tool in metabolic engineering of plants. Current Opinion in Biotechnology, 19(2), 131-137.

[10] Edwards, J. S., & Palsson, B. O. (2000). Metabolic flux balance analysis and the in silico analysis of Escherichia coli K-12 gene deletions. BMC bioinformatics, 1(1), 1.

[11] López Rivera Juan Antonio, Repositorio de Código del Primer Proyecto ("Aproximando Soluciones Al Problema del Agente Viajero"). https://github.com/stunnedbud/tsp-sa

[12] Laurent Heirendt & Sylvain Arreckx, Thomas Pfau, Sebastian N. Mendoza, Anne Richelle, Almut Heinken, Hulda S. Haraldsdottir, Jacek Wachowiak, Sarah M. Keating, Vanja Vlasov, Stefania Magnusdottir, Chiam Yu Ng, German Preciat, Alise Zagare, Siu H.J. Chan, Maike K. Aurich, Catherine M. Clancy, Jennifer Modamio, John T. Sauls, Alberto Noronha, Aarash Bordbar, Benjamin Cousins, Diana C. El Assal, Luis V. Valcarcel, Inigo Apaolaza, Susan Ghaderi, Masoud Ahookhosh, Marouen Ben Guebila, Andrejs Kostromins, Nicolas Sompairac, Hoai M. Le, Ding Ma, Yuekai Sun, Lin Wang, James T. Yurkovich, Miguel A.P. Oliveira, Phan T. Vuong, Lemmer P. El Assal, Inna Kuperstein, Andrei Zinovyev, H. Scott Hinton, William A. Bryant, Francisco J. Aragon Artacho, Francisco J. Planes, Egils Stalidzans, Alejandro Maass, Santosh Vempala, Michael Hucka, Michael A. Saunders, Costas D. Maranas, Nathan E. Lewis, Thomas Sauter, Bernhard Ø. Palsson, Ines Thiele, Ronan M.T. Fleming, Creation and analysis of biochemical constraint-based models: the COBRA Toolbox v3.0 (submitted), 2017, arXiv:1710.04038.