Interações entre proteínas e entre domínios

Lucas Machado Moschen

17 de abril de 2021

Professor: Luciano Guimarães de Castro

Resumo

Um estudo sobre como redes de interação proteína-proteína podem inferir interações domínio-domínio utilizando uma abordagem de programação linear e inteira. Essa abordagem é baseada no conceito de parcimônia, isto é, dados modelos diferentes que representem um fenômeno, se não haver evidência, escolhemos o modelo mais simples.

Sumário

1 Introdução

Proteínas são macromoléculas formadas por cadeias de aminoácidos. Elas são construídas em formato modular, compostas por domínios, que são unidades da proteína funcionais ou estruturais independentes do resto. Muitas são construídas por dois ou mais domínios dentro de um repertório relativamente pequeno, como na Figura 1. O que variam são as combinações e a ordem de disposição.

Figura 1: Representação da proteína citoplasmática Nck com seus três domínios SH3 e outro domínio SH2. Cada um desses domínios possui um parceiro de ligação em outras proteínas. Disponível em EMBL-EBI.

Frequentemente, os domínios individuais têm funções específicas, como catalisar uma reação ou ligação de uma molécula particular [1, 2].

Proteínas interagem umas com as outras nas células e o conjunto dessas interações é chamado de Rede de Interações Proteína-Proteína (Rede PPI), como na Figura 2. Matematicamente, representamo elas como um grafo (direcionado ou não direcionado) cujos nós são as proteínas e cujas arestas são as interações entre as proteínas, verificadas experimentalmente. Uma rede PPI é a base molecular do desenvolvimento e do funcionamento dos organismos. Seu entendimento é essencial, tanto para a compreensão de estados normais e doentes de células, quanto para o desenvolvimento de drogas. Ela pode ser construída para descrever uma série de informações, como, por exemplo, proteínas que estão na mesma regiões ao mesmo tempo; proteínas que são correguladas; proteínas que estão envolvidas com alguma função biológica; entre outras [2]. Em particular, o tópico de minha possível dissertação vai nesse sentido e, por esse motivo, foi o tema escolhido.

Como as interações entre proteínas geralmente ocorrem via domínios ao invés de toda a molécula, entender como os domínios interagem entre si pode facilitar a obtenção de redes PPI mais completas, com menor custo e tempo. Por esse motivo, predizer interações entre domínios (DDI) é um passo importante para a predição de PPI. Chamamos esse problema de problema de interação domínio-domínio (DDIP).

2 O problema de interação domínio-domínio (DDIP)

Denotamos uma rede PPI conhecida por um grafo não direcionado $\mathcal{N} = (\mathcal{P}, \mathcal{E})$, em que $\mathcal{P}$ é o conjunto de proteínas e $\mathcal{E}$ é o conjunto de pares de proteínas que interagem, dado algum experimento. Para cada proteína $P \in \mathcal{P}$, supomos conhecidos seus domínios D_P e definimos $\mathcal{D} = \{\{d_1, d_2\} : d_1 \in D_{P_1}, d_2 \in D_{P_2},\ \text{e}\ \{P_1, P_2\} \in \mathcal{E}\}$, isto é, o conjunto dos pares (não ordenados) de possíveis interações domínio domínio. Então, queremos inferir, para cada $I = \{P_1, P_2\} \in \mathcal{E}$, quais pares de domínios distintos em D_{P_1} e D_{P_2} explicam a interação entre as proteínas [1].

Vamos assumir que as interações domínio-domínio são conservadas entre as várias interações entre proteínas, isto é, elas tendem a repetir em diversos contextos. Também supomos que a interação entre proteínas evolui de forma parcimoniosa e que o conjunto DDI é bem aproximado pelo menor conjunto de interações necessárias para explicar a rede PPI, ou seja, sabemos que as relações domínio-domínio influenciam fortemente as relações proteína-proteína e utilizamos um método baseado em parcimônia para explicar essa inferência. O objetivo será, portanto, minimizar o número de interações domínio-domínio que justifiquem a interação entre proteínas observada na rede [3].

Seja $\mathcal{B}$ um grafo bipartido não direcionado com dois conjuntos independentes: $\mathcal{P}^2 = \mathcal{E}$ (conjunto dos pares de interações proteína-proteína) e $\mathcal{D}$ (pares de domínios), em que um elemento $\{P_1, P_2\} \in$

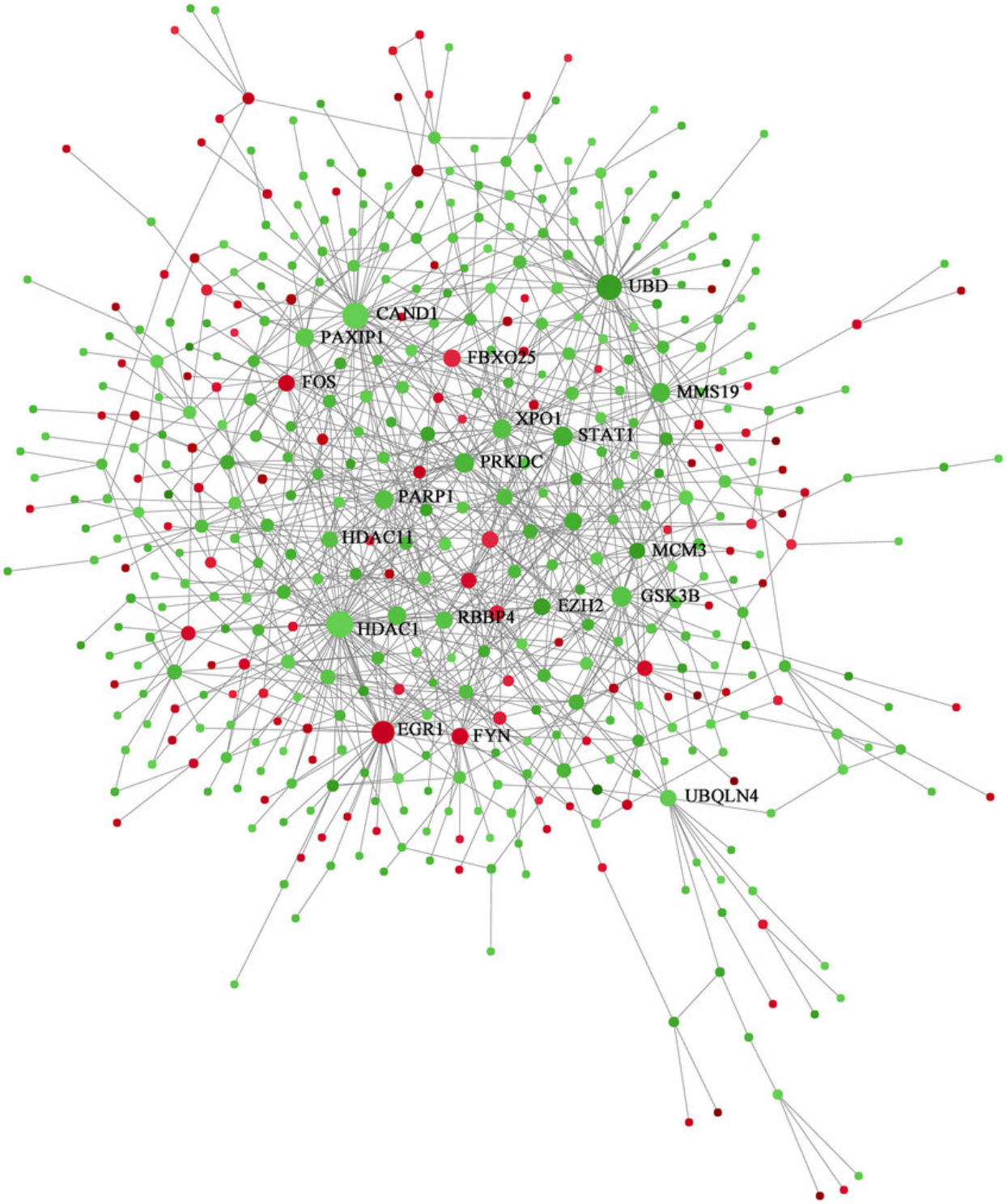

Figura 2: Exemplo de rede PPI usando NetworkAnalysist. Desenvolvido por Wei Zhong, et al.

$\mathcal{P}^2$ é ligado a $\{d_1, d_2\} \in \mathcal{D}$ se $d_1 \in D_{P_1}$ e $d_2 \in D_{P_2}$.

Definição (Cobertura): Um conjunto $S \subset \mathcal{D}$ é cobertura DDI se cada nó em $\mathcal{P}^2$ é adjacente a pelo menos um nó de S. Trivialmente $\mathcal{D}$ é uma cobertura por definição, portanto, sabemos que ela existe.

Para cada nó $x \in \mathcal{P}^2$, denotamos por $\mathcal{V}(x)$ o conjunto de nós em $\mathcal{D}$ que são vizinhos de x. Podemos resumir o problema, portanto, com a seguinte a proposição:

> Dado um grafo bipartido $\mathcal{B}$ derivado de $\mathcal{P}^2$ e $\mathcal{D}$, encontre uma cobertura DDI de tamanho mínimo em $\mathcal{B}$.

Nesse caso, o conjunto solução S^* é o conjunto mínimo que aproxima a explicação da interação entre proteínas.

3 Formulação com programação linear

Vamos definir as variáveis binárias x_{ij} que indicam se o nó $\{i, j\} \in \mathcal{D}$ pertence à cobertura S. Queremos minimizar o tamanho dessa cobertura, portanto, a função objetivo a ser minimizada é

$$\sum_{\{i,j\} \in \mathcal{D}} x_{ij}.$$

Para cada nó $v \in \mathcal{P}^2$, ou seja, para cada interação entre duas proteínas, queremos que pelo uma dupla de seus domínios interajam entre si. Assim, pelo menos um par de domínios vizinhos de v deve existir para explicar a PPI. Definimos a restrição, para cada v,

$$\sum_{\{i,j\} \in \mathcal{V}(v)} x_{ij} \geq 1$$

Além disso, temos, é claro, $0 \leq x_{ij} \leq 1, x_{ij} \in \mathbb{Z}$. Sabemos que espaço viável é não vazio, pois colocando todas as variáveis em 1, todas as restrições são verdadeiras.

4 Experimentação com dados hipotéticos

Nessa seção, vamos usar o ferramental de otimização na linguagem de programação *Julia* para fazer experimentos do problema. Vamos usar um dataset hipotético desenvolvido em [4].

[1]:
```
using Pkg
#pkg"add JuMP GLPK CSV DataFrames Cbc"
pkg"activate ../."
pkg"instantiate"
using JuMP, GLPK, CSV, LinearAlgebra, DataFrames, Cbc
```

```
  Activating environment at `~/Documents/linear-
programming/Project.toml`
```

4.1 Visualização do dado hipotético

Podemos visualizar os dados e transcrever para uma estrutura de dicionário.

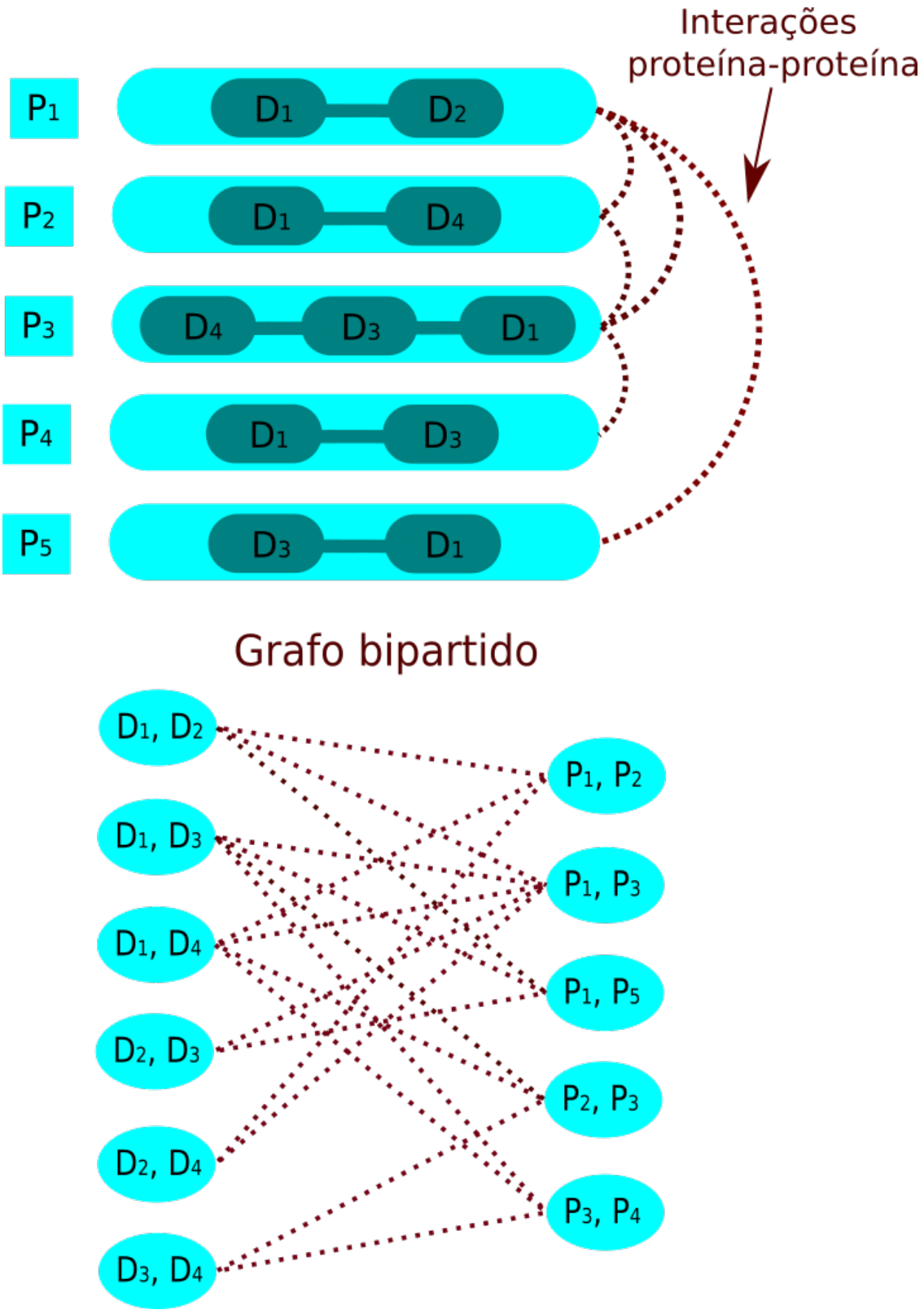

Figura 3: Exemplo simplificado de rede PPI com respectivos domínios e a construção do grafo bipartido correspondente.

[2]:
```
io = open("data/hypothetical_network.txt", "r")
print(read(io, String))
```

```
#######
# NODES
#######
0       P0      0       NULL    NULL    NULL    Hypothetical protein    Yeast
NULL
1       P1      0       NULL    NULL    NULL    Hypothetical protein    Yeast
NULL
2       P2      0       NULL    NULL    NULL    Hypothetical protein    Yeast
NULL
3       P3      0       NULL    NULL    NULL    Hypothetical protein    Yeast
NULL
4       P4      0       NULL    NULL    NULL    Hypothetical protein    Yeast
NULL
5       P5      0       NULL    NULL    NULL    Hypothetical protein    Worm
NULL
6       P6      0       NULL    NULL    NULL    Hypothetical protein    Worm
NULL
7       P7      0       NULL    NULL    NULL    Hypothetical protein    Worm
NULL
8       P8      0       NULL    NULL    NULL    Hypothetical protein    Worm
NULL
9       P9      0       NULL    NULL    NULL    Hypothetical protein    Human
NULL
10      P10     0       NULL    NULL    NULL    Hypothetical protein    Human
NULL
11      P11     0       NULL    NULL    NULL    Hypothetical protein    Human
NULL
#######
# EDGES
#######
0       0       1       h
1       0       2       h
2       0       3       h
3       0       4       h
4       1       2       h
5       1       3       h
6       1       4       h
7       5       6       h
8       5       7       h
9       6       8       h
10      7       8       h
11      9       10      h
12      10      11      h
#########################
```

```
# NODE -> DOMAIN MAPPINGS
#########################
0       Yellow
0       Blue
1       Azure
1       Orange
1       Green
1       Red
1       Blue
1       Red
2       Red
3       Red
3       Orange
4       Violet
4       Green
4       Red
4       Blue
4       Red
4       Yellow
4       Red
5       Orange
5       Red
6       Blue
7       Blue
8       Red
8       Green
9       Red
10      Blue
10      Orange
10      Violet
10      Azure
10      Green
11      Red
```

[3]:
```
protein_domain = Dict("P₁"=>["Yellow", "Blue"],
                      "P₂"=>["Azure", "Orange", "Green", "Red", "Blue", "Red"],
                      "P₃"=>["Green"],
                      "P₄"=>["Red", "Orange"] ,
                      "P₅"=>["Violet", "Green", "Red", "Blue", "Red", "Yellow",␣
 ↪"Red"] ,
                      "P₆"=>["Orange","Red"] ,
                      "P₇"=>["Blue"] ,
                      "P₈"=>["Blue"] ,
                      "P₉"=>["Red", "Green"] ,
                      "P₁₀"=>["Red"] ,
                      "P₁₁"=>["Blue", "Orange", "Violet", "Azure", "Green"],
                      "P₁₂"=>["Red"]);
```

```
ppi_network = [
    ("P1", "P2"), ("P1", "P3"), ("P1", "P4"), ("P1", "P5"),
    ("P2", "P3"), ("P2", "P4"), ("P2", "P5"),
    ("P6", "P7"), ("P6", "P8"),
    ("P7", "P9"), ("P8", "P9"), ("P10", "P11"), ("P11", "P12")
];

B_P = Dict([
    ("P1", "P2")=>[], ("P1", "P3")=>[], ("P1", "P4")=>[], ("P1", "P5")=>[],
    ("P2", "P3")=>[], ("P2", "P4")=>[], ("P2", "P5")=>[],
    ("P6", "P7")=>[], ("P6", "P8")=>[],
    ("P7", "P9")=>[], ("P8", "P9")=>[], ("P10", "P11")=>[], ("P11", "P12")=>[]
]);
```

4.2 Construindo o conjunto $\mathcal{D}$ e do grafo $\mathcal{B}$

Com a rede PPI definida, podemos construir o conjunto $\mathcal{D}$ com as relações entre os domínios. Com isso, teremos o grafo bipartido $\mathcal{B}$. Vamos separar o grafo apenas com as informações que precisamos: B_P indica, para cada interação PP, os seus vizinhos em $\mathcal{B}$ e B_D indica, para cada interação DD, a quantidade de vizinhos em $\mathcal{B}$.

[4]:
```
D = Dict()
for i in ppi_network
    for d1 in protein_domain[i[1]]
        for d2 in protein_domain[i[2]]
            if d1 != d2
                D[Set([d1, d2])] = get(D, Set([d1, d2]), length(D)+1)
                B_P[i] = union(B_P[i], D[Set([d1, d2])])
            end
        end
    end
end
B_D = zeros(length(D))
for v in B_P
    B_D[v[2]] .+= 1
end
```

4.3 Programa Linear e Inteiro

Vamos, enfim, construir o programa. Vamos chamar $ij = u$. Denote que os vizinhos de cada interação entre proteínas é dado por $B[i]$.

[5]:
```
ddip_model = Model(GLPK.Optimizer);

@variable(ddip_model,x[u=1:length(D)], Bin);

@objective(ddip_model, Min, sum(x));
```

```
@constraint(ddip_model, [sum(x[B_P[i]]) for i in ppi_network] .>= 1);
```

Com o problema definido, podemos otimizar! Felizmente, o algoritmo terminou tudo bem! Ele encontrou duas variáveis iguais a 1 (como podemos ver abaixo), sendo que uma delas satisfaz quase todas as restrições (exceto as que envolvem P_3).

[6]:
```
optimize!(ddip_model)

termination_status(ddip_model)
```

[6]:
```
OPTIMAL::TerminationStatusCode = 1
```

[7]:
```
objective_value(ddip_model)
```

[7]:
```
2.0
```

Podemos observar que as variáveis 8 (interação entre Green e Blue) e 9 (interação entre Red e Blue) foram as únicas que marcadas.

[8]:
```
value.(x)[[8,9]]
```

[8]:
```
2-element Vector{Float64}:
 1.0
 1.0
```

5 Adicionando mais realidade

Obtemos um resultado que diz quais interações domínio-domínio são necessárias para explicar as interações entre proteínas que observamos com um método concebido pela ideia de parcimônia. Nesse caso, obtemos um resultado 0 ou 1. Podemos, todavia, indicar a probabilidade de uma interação domínio a domínio existir. Observe que a restrição fica inalterada. Como observamos uma interação proteína-proteína $(P_1, P_2) \in \mathcal{P}^2$, podemos dizer que

$$\begin{aligned} 1 &= \mathbb{P}\left((P_1, P_2) \in \mathcal{P}^2\right) \\ &= \mathbb{P}\left((d_1, d_2) \in \mathcal{D}, \text{ para alguns } d_1 \in D_{P_1}, d_2 \in D_{P_2}\right) \\ &\leq \sum_{d_1 \in D_{P_1}, d_2 \in D_{P_2}} \mathbb{P}\left((d_1, d_2) \in \mathcal{D}\right) \end{aligned}$$

Suponha agora, que existem duas soluções de mesmo tamanho. Quando isso acontece, vamos dar um peso maior para aquela com maior sentido biológico, isto é, se um no $(d_1, d_2) \in \mathcal{D}$ tem mais vizinhos do que $(d_3, d_4) \in \mathcal{D}$, queremos que ele tenha mais chance de estar no conjunto ótimo. Isto é, vamos adicionar pesos que beneficiam nós com mais vizinhos! Assim, a função objetivo fica

$$\sum_{\{i,j\} \in \mathcal{D}} w_{ij} x_{ij}.$$

em que $w_{ij} = 1/\mathcal{V}(ij)$. No nosso exemplo, a resposta ficará inalterada, mas é interessante ter um critério de desempate biológico.

Observação: Colocar a variável entre 0 e 1, mas não binária, também simplifica as contas quando tivermos mais variáveis na Seção 6.

[9]:
```
ddip_model2 = Model(GLPK.Optimizer);

@variable(ddip_model2, 1 >= x[u=1:length(D)] >= 0.0);

@objective(ddip_model2, Min, sum(x./B_D));

@constraint(ddip_model2, [sum(x[B_P[i]]) for i in ppi_network] .>= 1);


optimize!(ddip_model2)
objective_value(ddip_model2)
```

[9]:
```
0.23376623376623376
```

[10]:
```
print(value.(x))
```

```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0]
```

As redes de PPI não são sempre inteiramente corretas. Elas são sujeitas ao ruído na experimentação. Esse ruído pode ser lidado de formas diferentes: adicionar pesos às interações PP indicando a probabilidade de elas serem verdadeiras, ou remover algumas restrições de forma "aleatória". A primeira, no nosso caso, representaria pesos nos nós em $\mathcal{P}^2$ e é mais difícil de representar em um modelo determinístico. A segunda tem uma formulação interessante: permitimos a escolha de uma porcentagem r de restrições, enquanto as outras são desconsideradas, isso é, consideraremos apenas uma porcentagem r das interações PP. Vamos fazer isso do seguinte modo:

Defina y_v uma variável binária para cada restrição. Queremos que

$$\sum_{\{i,j\}\in\mathcal{V}(v)} x_{ij} \geq y_v, \text{ para cada } v \in \mathcal{P}^2$$

E, além disso, restringimos

$$\sum_{v\in\mathcal{P}^2} y_v \geq r|\mathcal{P}^2|$$

Essa restrição pode fazer tirar uma interação PP quando ela tem pouca influência ou advém de um ruído.

[11]:
```
ddip_model3 = function(r)

    model = Model(GLPK.Optimizer);

    @variable(model, 1 >= x[u=1:length(D)] >= 0.0);
    @variable(model, y[v=1:length(B_P)], Bin);

    @objective(model, Min, sum(x./B_D));
```

```
    @constraint(model, [sum(x[B_P[i]]) for i in ppi_network] .>= y);
    @constraint(model, sum(y) >= r*length(B_P));

    optimize!(model)
    return objective_value(model), value.(x), value.(y)
    end;
```

Vamos experimentar alguns valores de r

[12]:
```
obj1, xx1, yy1 = ddip_model3(0);
obj2, xx2, yy2 = ddip_model3(0.5);
obj3, xx3, yy3 = ddip_model3(0.9);
```

Quando $r = 0$, permitimos que todas as restrições sejam deixadas de lado. Certamente, estamos colocando ruído a mais. Com $r = 0.5$, para satisfazer metade das restrições, apenas um nó é suficiente. Para $r = 0.9$, apesar de ter considerado um pouco de ruído, ainda mantivemos a mesma escolha de quando $r = 1$.

[13]:
```
(obj1, obj2, obj3)
```

[13]:
```
(0.0, 0.09090909090909091, 0.23376623376623376)
```

[14]:
```
print(xx2)
print("\n")
print(xx3)
```

```
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0]
[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0]
```

6 Experimentação com dados reais

Vamos utilizar essa modelagem em um problema real, com dados de [4].

[15]:
```
pd_table = CSV.File("data/protein_domain.txt", delim='\t') |> DataFrame;
ppi_table = CSV.File("data/ppi_network.txt", delim='\t') |> DataFrame;
select!(ppi_table, Not(:"exp_class"));
```

O número de interações é 26.032, enquanto o número de mapas entre nós e domínios é 30.986.

[16]:
```
print(size(pd_table))
print(size(ppi_table))
```

```
(30986, 2)(26032, 3)
```

6.1 Proteínas e seus domínios

Primeiro montamos o dicionário com cada proteína e seus respectivos domínios:

[17]:
```
proteins = Dict()
for p in eachrow(pd_table)
    proteins[p[1]] = union(get(proteins, p[1], []), [p[2]])
end
proteins[100]
```

[17]:
```
12-element Vector{Any}:
 "C2"
 "Pfam-B_11112"
 "Pfam-B_34975"
 "Pfam-B_44417"
 "Pfam-B_44419"
 "Pfam-B_5217"
 "Pfam-B_68720"
 "PH"
 "PI-PLC-X"
 "PI-PLC-Y"
 "SH2"
 "SH3_1"
```

6.2 Rede PPI

Montamos a rede PPI:

[18]:
```
ppi_real = []
for i in eachrow(ppi_table)
    if i[2] != i[3]
        ppi_real = append!(ppi_real, [(i[2], i[3])])
    end
end
```

6.3 Grafo bipartido $\mathcal{B}$

Montamos $\mathcal{B}$:

[19]:
```
B_p = Dict(i => [] for i in ppi_real);
```

[20]:
```
D = Dict()
for i in ppi_real
    for d1 in proteins[i[1]]
        for d2 in proteins[i[2]]
            D[Set([d1, d2])] = get(D, Set([d1, d2]), length(D)+1)
            B_p[i] = union(B_p[i], D[Set([d1, d2])])
        end
    end
end
B_d = zeros(length(D))
for v in B_p
    B_d[v[2]] .+= 1
```

```
end
```

6.4 Modelo Final

Agora podemos usar o modelo! Lembramos que ele tem muitas, mas muitas mais variáveis que nosso teste, então deve demorar bastante mais tempo para encerrar o programa, pelo menos na primeira rodagem! Observe que o número de variáveis é de quase 200mil.

[23]:
```
length(D) + length(B_p)
```

[23]:
```
198935
```

[21]:
```
ddip_model_real = function(r, D, BP, BD, PPI)

    model = Model(Cbc.Optimizer);

    @variable(model, 1.0 >= x[u=1:length(D)] >= 0.0);
    @variable(model, y[v=1:length(B_P)], Bin);

    @objective(model, Min, sum(x./BD));

    @constraint(model, [sum(x[BP[i]]) for i in PPI] .>= 1);
    @constraint(model, sum(y) >= r*length(B_P));

    optimize!(model)
    return objective_value(model), value.(x), value.(y)
end;
```

[22]:
```
@time obj_real1, xx_real1, yy_real1 = ddip_model_real(0.9, D, B_p, B_d,␣
↪ppi_real);
```

```
167.731943 seconds (29.02 M allocations: 1.692 GiB, 0.46% gc time, 0.56%
compilation time)
Welcome to the CBC MILP Solver
Version: 2.10.5
Build Date: Mar 11 2021

command line - Cbc_C_Interface -solve -quit (default strategy 1)
Continuous objective value is 14023.8 - 0.18 seconds
Cgl0004I processed model has 6304 rows, 10233 columns (0 integer (0 of which
binary)) and 19312 elements
Cbc3007W No integer variables - nothing to do
Cuts at root node changed objective from 14023.8 to -1.79769e+308
Probing was tried 0 times and created 0 cuts of which 0 were active after adding
rounds of cuts (0.000 seconds)
Gomory was tried 0 times and created 0 cuts of which 0 were active after adding
rounds of cuts (0.000 seconds)
```

```
Knapsack was tried 0 times and created 0 cuts of which 0 were active after
adding rounds of cuts (0.000 seconds)
Clique was tried 0 times and created 0 cuts of which 0 were active after adding
rounds of cuts (0.000 seconds)
MixedIntegerRounding2 was tried 0 times and created 0 cuts of which 0 were
active after adding rounds of cuts (0.000 seconds)
FlowCover was tried 0 times and created 0 cuts of which 0 were active after
adding rounds of cuts (0.000 seconds)
TwoMirCuts was tried 0 times and created 0 cuts of which 0 were active after
adding rounds of cuts (0.000 seconds)
ZeroHalf was tried 0 times and created 0 cuts of which 0 were active after
adding rounds of cuts (0.000 seconds)

Result - Optimal solution found

Objective value:                14023.79644197
Enumerated nodes:               0
Total iterations:               0
Time (CPU seconds):             1.80
Time (Wallclock seconds):       2.00

Total time (CPU seconds):       1.80   (Wallclock seconds):       2.00
```

O programa teve sucesso e levou quase 3min para concluir o processo. Podemos ver qual a porcentagem (mínima) necessária para justificar as interações entre proteínas. Em torno de 9% por cento das interações domínio-domínio foram necessárias para justificar todo o aparato.

[24]:
```
sum(xx_real1)/length(xx_real1)
```

[24]:
```
0.09398963730569948
```

Note que um pouco mais de 90% das restrições foram usadas, o que diminui um pouquinho do possível ruído.

[25]:
```
sum(yy_real1)/length(yy_real1)
```

[25]:
```
0.9230769230769231
```

7 Conclusão

Podemos observar uma modelagem com otimização linear em um problema complexo de biologia. Apesar das diversas simplificações, o método simplex é simples de interpretar e é rápido para funcionar, o que, em muitos casos, é essencial para um conjunto de dados longos. No exemplo real, percebemos uma dificuldade no programa para encerrar a otimização, que levou quase 3min.

Referências

[1] Althaus, Ernst & Klau, Gunnar & Kohlbacher, Oliver & Lenhof, Hans-Peter & Knut, Reinert. (2009). *Integer Linear Programming In Computational Biology.* J Proteome Res, Volume 5760 of Lecture Notes in Computer Science. 5760. 199-218. 10.1007/978-3-642-03456-5_14.

[2] EMBL-EBI. Protein classification: An introduction to EMBL-EBI resources. Disponível https://www.ebi.ac.uk/training/online/courses/protein-classification-intro-ebi-resources/.

[3] Guimarães, K.S., Jothi, R., Zotenko, E. et al. Predicting domain-domain interactions using a parsimony approach. Genome Biol 7, R104 (2006). https://doi.org/10.1186/gb-2006-7-11-r104

[4] Riley, R., Lee, C., Sabatti, C. et al. Inferring protein domain interactions from databases of interacting proteins. Genome Biol 6, R89 (2005). https://doi.org/10.1186/gb-2005-6-10-r89