Detecção de comunidades em grafos com ACO

Cleverson de Souza Carneiro

***Abstract*—A necessidade de detecção de comunidades em grafos é um problema muito comum em diversas áreas de pesquisas que utilizam grafos como modelos, sendo este um problema cuja solução exata necessita de algoritmos exponênciais, a busca por algoritmos heuristicos com tempos e precisões cada vez melhores torna-se um desafio para os cientistas da computação. Este trabalho buscou uma solução para o problema, no campo da computação evolutiva e encontrou uma proposta de algoritmo de colônia de formiga. O algoritmo foi implementado e executado em diversos grafos, mostrando ser ACO uma boa técnica para detecção de comunidades, principalmente podendo ser aplicado em computação paralela.**

***Keywords*—Comunidades, Grafo, ACO, Computação Evolucionária**

I. Introdução

A detecção de comunidades em grafo é uma tarefa importante para os campos da biologia, sociologia e ciência da computação. Ela permite por exemplo detectar comunidades nas iterações de proteinas no campo da biologia, ou nas redes sociais detectar padrões de comportamentos de um grupo de pessoa [Romdhane et al., 2013].

Um grafo é um modelo matemático que representa um conjunto de vértices e um conjunto de arestas ligando os vertices uns aos outros, a aresta representa o relacionamento entre dois vérticies. Por exemplo em uma rede social poderiamos representar cada usuário como um vértice e o vinculo de amizade entre eles como uma aresta.

Em um grafo pode existir conjuntos de vértices mais conectados entre si do que os restantes dos vérticies. Este conjunto de vérticies são geralmente chamados de grupos (clusters), comunidades ou módulos [Palla et al., 2005]. O objetivo dos algoritmos de detecção de comunidades é encontrar esses grupos de vértices de maneira rápida e com qualidade.

A detecção de comunidade é um problema NP-hard, pois não existe nenhum algoritmo que encontre a solução ótima em tempo polinomial. Sendo assim, para muitos casos reais, torna-se necessária a aplicação de algoritmos heurísticos para encontrar uma solução viável em um tempo computacional factível.

Desta forma, foi estudado neste trabalho a aplicação de algoritmo de ACO (Otimização por Colônia de Formigas) na detecção de comunidades. Foram utilizados como referência os trabalhos de [Mandala et al., 2013] e [Romdhane et al., 2013], sendo implementado na linguagem Python o algoritmo proposto por [Mandala et al., 2013].

II. Definições e Conceitos

A. Comunidade

Uma comunidade pode ser definida comparando o número de arestas entre seus vértices com o número de arestas de um grafo aleatório G_{null} com a mesma distribuição de graus que o gráfico original G. O número esperado de arestas entre dois verticies em G_{null} é dado por $E[A_{ij}] = 2mp_ip_j = \frac{k_ik_j}{2m}$. [Newman and Girvan, 2004]

Desta forma a qualidade da comunidade é definida pelo excesso de números de arestas dentro da comunidade comparado com o número de arestas no grafo aleatório.

Podemos definir a contribuição de dois verticies i e j dentro da mesma comunidade por b_{ij}:

$$b_{ij} = a_{ij} - \frac{k_ik_j}{2m} \tag{1}$$

Por meio da modularidade, o problema de agrupamento em grafos se resume a um problema de otimização, cujo o objetivo é encontrar partições que maximizem a modularidade.

B. Otimização por Colônia de Formigas - ACO

1) Busca Local: Os algoritmos de pesquisa local (LS) são heurísticas de melhoramento que pesquisam uma melhor solução na vizinhança da solução atual, até que não se possa fazer qualquer melhoria adicional. Algoritmos de busca locais podem ser categorizados pelas vizinhanças que eles consideram. Por exemplo, a vizinhança de uma solução representada por um vetor binário pode ser definida pelas soluções que podem ser obtidas alterando um bit ou múltiplos bits no vetor binário simultaneamente [Merz and Freisleben, 2002].

A forma mais simples de busca local para o BQP é a pesquisa local 1-opt: Em cada etapa, uma nova solução com um maior fitness na vizinhança da solução atual é pesquisada. A vizinhança da solução atual é definida pelo conjunto de soluções que podem ser alcançadas movendo um único bit. Assim, a vizinhança 1-opt contém todas as soluções com uma distância de hamming de 1 para a solução atual [Merz and Freisleben, 2002].

III. Construção da Solução

A. Grau do Vértice

Considerar um grafo binário uniderecional G = (V,E) representado por uma matrix de adjacencia A_{nxn}. Onde $a_{ij} = 1$ se $(i, j) \in E$ e $a_{ij} = 0$ se $(i, j) \notin E$

Grau do node i:

$$k_i = \sum_{j=1}^{n} a_{ij} \tag{2}$$

B. Modularidade

Podemos definir a contribuição de dois verticies i e j dentro da mesma comunidade por b_{ij}:

$$b_{ij} = a_{ij} - \frac{k_i k_j}{2m} \tag{3}$$

C. Qualidade da Partição

$$Q_G(x) = \frac{1}{2m} \sum_{i=1}^{n} \sum_{j=1}^{n} b_{ij} \delta(x_i, x_j) \tag{4}$$

A qualidade da partição irá variar entre -0.5 e 1.0.

D. Busca Global

Em cada iteração t, cada formiga k constrói uma solução potencial, usando o depósito de feromônio, como probabilidade de atribuir o grupo 1 a um vertice. Portanto, a probabilidade de construir uma solução é dada como:

$$P(s_i^{kt} = 1) = \tau_i \qquad \text{e} \qquad P(s_i^{kt} = 0) = 1 - \tau_i \qquad \forall i \in 1, ..., n \tag{5}$$

E. Busca Local

A solução construída por cada formiga k é ainda melhorada por um procedimento de pesquisa local de 1-opt [Merz and Freisleben, 2002]. Essa busca local melhora a solução executando recursivamente uma busca de descida dentro de uma vizinhança de 1 hamming de s onde s' é um vizinho de 1-hamming de s se eles diferem exatamente em um componente. O funcionamento deste procedimento de busca pode ser descrito definindo o ganho, $g_i(s)$, que é o aumento no valor da função objetivo ao mudar o valor do i-ésimo componente da solução Eq. 6.

$$g_i(s) = b_{ii}(1 - 2s_i) + 2 \sum_{j=1, j \neq i}^{n} b_{ji} s_j (1 - 2s_i) \tag{6}$$

F. Atualização do Feromônio

O feromônio é atualizado através da média ponderada das três soluções encontradas. Sendo s_{tb} melhor solução encontrada na iteração, s_{rb} melhor solução encontrada antes da convergência e s_{bs} melhor solução global:

$$s^* = \sum_{s \in S} \frac{Q(s)}{\sum_{s \in S} Q(s)} s \qquad \text{where} \qquad S = \{s_{tb}, s_{rb}, s_{bs}\} \tag{7}$$

Sendo τ o vetor de feromônio e ρ o fator de evaporação. Temos a sua atualização da seguinte forma:

$$\tau = \tau + \rho(s^* - \tau) \tag{8}$$

IV. Implementação

Pseudo código do fluxo principal:

```
comunities = []
cnj_nodes = []
for i in range(total_nodes):
    cnj_nodes.append(i)

sub_problems = []
sub_problem = SubProblem(cnj_nodes)
sub_problems.append(sub_problem)

while len(sub_problems) > 0:
    sub_problem = sub_problems.pop(0)

    ant = AntSystem(sub_problem)

    for i in range(MAXIMO_ITERACOES):
        ant.iteration()

    if ant.best_value > (0.01 * total_edges):
        modularity += ant.best_value / total_edges
        cnj_nodes_tmp = ant.get_sub_cnj_nodes(True)
        sub_problem = SubProblem(cnj_nodes_tmp)
        sub_problems.append(sub_problem)
        cnj_nodes_tmp = ant.get_sub_cnj_nodes(False)
        sub_problem = SubProblem(cnj_nodes_tmp)
        sub_problems.append(sub_problem)
    else:
        comunities.append(sub_problem.cnj_nodes)
```

Código completo disponível em: https://github.com/clvrsn/acocluster/

V. Resultados

A. Comparação com outros algoritmos

Comparação do algoritmo ACO implementação original, com a implementação deste trabalho ACO*, com outros EO [Duch and Arenas, 2005] e NM [Newman and Girvan, 2004].

A tabela apresenta a comparação da modularidade e o número de comunidades entre parenteses.

	Nodes	EO	NM	ACO	ACO *
Karate club	34	0.419 (4)	0.419 (2)	0.419 (4)	0.392 (3)
Jazz musicians	198	0.445 (5)	0.445 (4)	0.595 (4)	0.505 (8)
Metabolic	453	0.434 (12)	0.434 (10)	0.669 (7)	0.525 (8)
E-mail	1333	0.574 (15)	0.574 (13)	0.716 (8)	—

Esta comparação mostrada no artigo [Mandala et al., 2013] mostrou que o algoritmo proposto apresentou uma melhor qualidade quando comparado aos outros dois algoritmos.

B. Testes do Algoritmo Implementado

1) Grafo Simples: Grafo {(1, 2), (1, 10), (2, 3), (2, 10), (3, 7), (4, 5), (4, 10), (5, 9), (5, 10), (6, 7)}. O resultado foi modularidade: 0.325, tempo de execução: 0.929 segundos e 3 comunidades {(3, 4, 7, 8, 9), (2, 5, 6), (0, 1)}.

2) Grafo Karate: O resultado foi modularidade: 0.419, tempo de execução: 10.057 segundos e 4 comunidades. Resultado em: https://github.com/clvrsn/acocluster/blob/master/output_karate.txt

3) Grafo Jazz: O resultado foi modularidade: 0.563, tempo de execução: 326.332 segundos e 6 comunidades. Resultado em: https://github.com/clvrsn/acocluster/blob/master/output_jazz.txt

4) Grafo Metabolic: O resultado foi modularidade: 0.602, tempo de execução: 1843.806 segundos e 8 comunidades. Resultado em: https://github.com/clvrsn/acocluster/blob/master/output_metabolic.txt

C. Alteração de Parametros ACO

Em alguns teste o código implementado neste trabalho teve uma performance inferior aos testes descritos no artigo.

Utilizando o grafo Karate club como referência. Foram feitos alguns ajustes nos parametros do ACO para buscar um melhor resolutado.

N. Formigas	Tx. Eva. Feromonio	N. Iterações	Modularidade
5	0.01	40	0.1997 (5)
10	0.05	45	0.3330 (3)
15	0.10	50	0.3909 (4)
20	0.15	55	0.4188 (4)
20	0.15	60	0.4132 (4)

Os testes alterando as parametrizações do algoritmo ACO, mostrou uma estabilidade com 20 formigas e 55 iterações e 0.15 de taxa de evaporação de feromonio.

VI. Conclusion

O estudo sobre ACO para clusterização mostrou que esta abordagem é bem promissora e inovadora. O artigo utilizado como referência deixou lacunas que dificultaram bastante a implementação e não foi possível garantir que está 100% implementado conforme descrito pelos autores. Principalmente quanto ao tempo de execução nos grafos Metabolic e E-mail.

Uma melhoria substancial no tempo de execução seria a paralelização do algoritmo. O que é totalmente possível de ser feito no processo de busca local e também na solução dos sub-problemas.

References

[Duch and Arenas, 2005] Duch, J. and Arenas, A. (2005). Community detection in complex networks using extremal optimization. *Physical review E*, 72(2):027104.

[Mandala et al., 2013] Mandala, S. R., Kumara, S. R., Rao, C. R., and Albert, R. (2013). Clustering social networks using ant colony optimization. *Operational Research*, 13(1):47–65.

[Merz and Freisleben, 2002] Merz, P. and Freisleben, B. (2002). Greedy and local search heuristics for unconstrained binary quadratic programming. *Journal of Heuristics*, 8(2):197–213.

[Newman and Girvan, 2004] Newman, M. E. J. and Girvan, M. (2004). Finding and evaluating community structure in networks. *Physical Review*, E 69(026113).

[Palla et al., 2005] Palla, G., Derényi, I., Farkas, I., and Vicsek, T. (2005). Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818.

[Romdhane et al., 2013] Romdhane, L. B., Chaabani, Y., and Zardi, H. (2013). A robust ant colony optimization-based algorithm for community mining in large scale oriented social graphs. *Expert Syst. Appl.*, 40(14):5709–5718.