Título: GRASP para o problema *Weighted Target Set Selection* .

Estudante: Jorge Reynaldo Moreno Ramírez

Orientador(es): Simone de Lima Martins

Alexandre Plastino de Carvalho

Resumo:

Dado um grafo com pesos não negativos atribuídos aos vértices, o problema Weighted Target Set Selection (WTSS) consiste em encontrar um subconjunto de vértices *S* do grafo, tal que a soma dos custos dos vértices pertencentes a *S* seja mínima. Este problema é NP-difícil e tem muitas aplicações em análise de redes sociais e fenómenos naturais. Este trabalho propõe o uso da metaheurística GRASP para resolver o problema WTSS. Foram realizados experimentos em instâncias correspondentes com redes reais e os resultados obtidos mostram a efetividade do método proposto.

Abstract:

Given a graph with non negative weight assigned to the vertices, the problem Weighted Target Set Selection (WTSS) is to find a subset S of the graph, such that the sum of the costs of vertices belonging to *S* is minimal. This problem has been proven to be NP-hard and has many applications in social network analysis and natural phenomena. This paper proposes the use of GRASP heuristic to solve the problem WTSS. Experiments were performed in real networks and the results show the effectiveness of the proposed method.

1 Introdução

Processos em grafos tem recibido recentemente grande importância no estudo das redes sociais e numerosos artigos relacionados como isto aparecem na literatura (Chen 2009). Os principais estudos estão centrado na análise da difusão de uma noticia viral na rede. Um maneira de analisar este fenômeno é usando o modelo de processos irreversíveis, que modela uma situação na qual quando um nó da rede é "infectado" com a noticia viral, mantém este estado "infectado". Outro aspecto importante neste modelo é que os nós "não infectados" adquirem este estado se o número de vizinhos infectados é maior que um certo limite próprio de cada nó.

Associado a este modelo vários existem problemas de otimização combinatória. Um destes problemas é determinar qual é o número mínimo de nós que devem ser infectados inicialmente para infectar a rede toda após um certo período de tempo (Chen et al. 2009). Outro problema associado é quando os vértices tem um custo ou peso. Neste problema o objetivo é determinar quais nós devem ser selecionados inicialmente para infectar a rede toda sendo que a soma de seus custos seja mínima. Este problema é conhecido na literatura como Weighted Target Set Selection (WTSS). Foi proposto em (Raghavan e Zhang 2013) e resulta do interesse em problemas reais, onde se objetiva proliferar uma informação em toda a rede, procurando fazer o menor investimento.

Em (Zhanh e Sahin 2014) é apresentado um modelo de inteiro misto para este problema. Um algoritmo polinomial para determinar WTSS em árvores é apresentado em (Raghavan e Zhang 2015), assim como um algoritmo branch and cut para redes gerais. Em (Cordasco et al. 2015) é proposto um algoritmo guloso para resolver este problema obtendo melhores resultados que trabalhos prévios.

O problema WTSS é um problema de otimização combinatória e metaheurísticas provaram ser uma poderosa ferramenta para resolver este tipo de problemas. A metaheurística Greedy Randomized Adaptive Search Procedures (GRASP), desde seu surgimento (Feo e Resende 1989), tem sido aplicada com sucesso em numeroso problemas. Neste trabalho é desenvolvida a heurística GRASP para resolver o problema WTSS.

O documento está organizado como é explicado a seguir. Na seção 2 é definido o problema e as notações usadas no trabalho. Na seção 3 é apresentado o esquema geral da metaheurística GRASP e as estratégias desenvolvidas para gerar a solução inicial e busca local. A seleção dos parâmetros e os resultados alcançados nos experimentos computacionais são apresentados na seção 4. Finalmente, conclusões do trabalho são expostas na seção 5.

2 Definição do problema

Seja $G=(V_G,E_G)$ um grafo não dirigido, onde V_G representa o conjunto de vértices do grafo e E_G denota o conjunto de arestas. A vizinhança de um vértice v será definida como $N(v)=\{u\in V_G|(u,v)\in E_G\}$ e o grau de um vértice v será $d_G(v)=|N(v)|$.

Uma rótulo sobre G se define como uma função $c:V_G\rightarrow\{0,1\}$ e um processo sobre um grafo G se define como um sequência infinita de rótulos, de forma que para todo $k\geq 0$, c_{k+1} é obtido a partir de c_k a partir de uma regra inequívoca. Quando sobre G se define uma função $t:V_G\rightarrow\mathbb{N}$ (denominada função limiar) e se usa a seguinte regra para obter os rótulos,

$$\forall\, v\in V_G, \forall\, k\geq 0,\ \ c_{k+1}(v)=1 \ \Leftrightarrow\ c_k(v)=1 \ \vee\ |\{u\in N(v)|c_k(u)=1\}|\geq t(v)$$

então o processo é chamado irreversível. Em essência, em um processo irreversível, um vértice terá atribuído o valor 1 para o rótulo, se ele tem um rótulo com esse valor ou o número de

vizinhos com rótulo igual a 1 supera ou iguala o limite para o vértice definido pela função limiar.

Um conjunto de conversão $S \subseteq V_G$ define o conjunto de vértices que inicialmente tem associado o valor 1 no rótulo inicial, que assegura a que a partir de um rótulo, todos os vértices tenham valor associado 1 nos rótulos seguintes. Formalmente

$$S = \{ v \in V_G \mid c_0(v) = 1 \ \wedge \ \exists i \geq 0 (\forall v \in V_G, \forall k \geq i, \ c_k(v) = 1) \}$$

Se a cada vértice se associa um peso mediante uma função $w : V_G \rightarrow \mathbb{N}$, o custo de um conjunto de vértices $S \subseteq V_G$ se define como $C(S) = \sum_{v \in S} w(v)$. O problema Weighted Target Set Selection (WTSS) consiste em encontrar o conjunto de conversão de menor custo, ou seja, encontrar o conjunto de conversão $S \subseteq V_G$ tal que:

$$C(S) = min \{ C(S') \mid S' \ é \ um \ \ conjunto \ \ de \ \ conversão \}$$

Finalmente, um vértice estará *infectado* se e só se está rotulado com 1.

3 Algoritmo proposto

3.1 Metaheurística GRASP

A metaheurística *Greedy Randomized Adaptive Search Procedures* (GRASP) é um método para obtenção de soluções de boa qualidade para problemas de otimização combinatória (Resende et al. 2010). O GRASP pode ser definido como um processo iterativo, onde em cada iteração se realizam uma construção e uma busca local. Na fase de construção, a ideia é combinar um algoritmo guloso com uma componente aleatória. Algoritmos gulosos são geralmente iterativos e, a cada iteração, geralmente incluem na solução em construção o elemento que traz o melhor incremento possível na sua qualidade. Porém este procedimento nem sempre obtém o valor ótimo. Por isso, inclui-se uma componente aleatória no procedimento de construção para trazer diversificação nas soluções criadas, o que pode conduzir a bons resultados.

Uma vez gerada uma solução, é percorrida uma vizinhança em busca de melhores soluções. Este processo é repetido até atingir o critério de parada. A melhor solução encontrada dentre todas as iterações é retornada como resultado. A Figura 3.1 contém o pseudocódigo do GRASP para um problema de minimização.

```
procedimento GRASP(α)
1.  bestSol := ∅ ;
2.  repita
3.      sol := Construcao_Inicial(α) ;
4.      sol := Busca_Local(sol) ;
5.      se Aval(sol) < Aval(bestSol) então
6.          betsSol := sol ;
7.      fim-se
8.  até Criterio_de_Parada() ;
9.  retorne bestSol ;
```

Figura 3.1: Pseudocódigo da metaheurística GRASP

3.2 Solução inicial

Para criar a solução inicial foi utilizado o algoritmo WTSS proposto em (Cordasco et al. 2015), com uma modificação na escolha gulosa. Este é um algoritmo guloso que em cada passo atribui o valor 0 (“não infectado”) ou 1 (“infectado”) a um vértice. Como entrada se recebe um grafo, e os limites e custos associados a cada vértice.

Num primeiro caso, o algoritmo procura aqueles vértice que tem um número de vértices adjacentes “infectados” igual ou superior a seu limite. Neste caso, esses vértices serão rotulados com 0. Se não tiver sucesso esta busca, então são procurados aqueles vértices que não tem (nem podem ter) suficientes vizinhos “infectados” como para eles se infectar, e como consequência, estes vértices serão rotulados com 1. Se nenhum dos casos anteriores tiveram sucesso, então o algoritmo faz uma escolha gulosa para determinar que vértice rotular com 0. Na figura 3.2 é apresentado o pseudocódigo do algoritmo modificado denominado como WTSS2.

A modificação foi feita no critério para escolha gulosa (linha 17), que vai selecionar aquele vértice que seria um bom candidato a ser um vértice não infectado. O critério utilizado em (Cordasco et al. 2015) foi:

$$v := argmax_{u \in U} \left\{ \frac{c(u) \cdot k(u)}{d(u) \cdot (d(u)+1)} \right\}$$

Ao analisar este critério, pode se perceber que vértices de alto custo e pouco grau são considerados bons candidatos para serem definidos como não infectados. Além disso, o termo $\frac{k(u)}{(d(u)+1)}$ beneficia a vértices com limite próximo ao grau. A ideia parece razoável, pois ao selecionar um vértice no caso 3 com limite próximo ao grau, aqueles vértices como muita diferencia entre o limite e o grau tem mais oportunidade de ser analisados nos casos 1 e 2 do algoritmo que não se correspondem com escolhas gulosas. De esta forma se estaria diminuindo o número de vezes que é desenvolvida a escolha gulosa. Tomando isto em consideração foi modifica a escolha gulosa mediante o seguinte critério:

$$v := argmax_{u \in U} \left\{ \frac{c(u) \cdot |V_G|}{d(u) \cdot (d(u)+1-k(u))} \right\}$$

Este critério também vai escolher vértices de alto custo, menor grau, e limite perto de grau. Só que aumenta a chance de ser escolhidos vértices com pequena diferença entre o limite e o grau. Isto é feito com o propósito de diminuir o número de escolhas gulosas.

Nos experimentos realizados na seção 4, pode-se notar como esta variação proporciona melhores soluções e com um número menor de vértices infectados, pelo qual foi escolhida para gerar as soluções iniciais.

O procedimento utilizado para a construção inicial é mostrado na figura 3.3. Basicamente consiste em usar uma lista restrita de candidatos. Neste caso consiste em avaliar os vértices e

escolher um vértice $u \in U$ com valor para $f(u) = \frac{c(u) \cdot |V_G|}{d(u) \cdot (d(u) + 1 - k(u))}$ no intervalo $[f_{min} + (f_{max} - f_{min}) \cdot \alpha, f_{max}]$, sendo f_{min} e f_{max} os valores mínimos e máximos de f no conjunto de vértices avaliados.

```
procedimento WTSS2 (grafo G , array t , array c )
1.  S := ∅ ;
2.  U := V_G ;
3.  para cada v ∈ V_G
4.      d(v) := d_G(v) ;
5.      k(v) := t(v) ;
6.      N(v) := N_G(v) ;
7.  enquanto U ≠ ∅
8.      se existe v ∈ U tal que k(v) = 0 então                    //caso 1
9.          para cada u ∈ N(v)
10.             k(u) := max(k(u) − 1, 0) ;
11.     senão
12.         se existe v ∈ U tal que d(v) < k(v) então             //caso 2
13.             S := S ∪ {v} ;
14.             para cada u ∈ N(v)
15.                 k(u) := max(k(u) − 1, 0) ;
16.         senão                                                //caso 3
17.             v := argmax_{u∈U} { c(u)·|V_G| / (d(u)·(d(u) + 1 − k(u))) } ;
18.  para cada u ∈ N(v)
19.      d(u) := d(u) − 1 ;
20.      N(u) := N(u) − {v} ;
21.  U := U − {v} ;
22.  fim-enquanto
23.  retorne S ;
```

Figura 3.2: Pseudocódigo do procedimento WTSS2

procedimento Construcao_Inicial (G , t , c , α)

1. $S := \emptyset$;
2. $U := V_G$;
3. **para cada** $v \in V_G$
4. $d(v) := d_G(v)$;
5. $k(v) := t(v)$;
6. $N(v) := N_G(v)$;
7. **enquanto** $U \neq \emptyset$
8. **se existe** $v \in U$ **tal que** $k(v) = 0$ **então**
9. **para cada** $u \in N(v)$
10. $k(u) := max(k(u) - 1, 0)$;
11. **senão**
12. **se existe** $v \in U$ **tal que** $d(v) < k(v)$ **então**
13. $S := S \cup \{v\}$;
14. **para cada** $u \in N(v)$
15. $k(u) := max(k(u) - 1, 0)$;
16. **senão**
17. $f_{max} := max\left\{ \dfrac{c(u) \cdot |V_G|}{d(u) \cdot (d(u) + 1 - k(u))} \mid u \in U \right\}$;
18. $f_{min} := min\left\{ \dfrac{c(u) \cdot |V_G|}{d(u) \cdot (d(u) + 1 - k(u))} \mid u \in U \right\}$;
19. $v := arg_{random}\{ f(u) \in [f_{min} + (f_{max} - f_{min}) \cdot \alpha, f_{max}] \}$
20. **para cada** $u \in N(v)$
21. $d(u) := d(u) - 1$;
22. $N(u) := N(u) - \{v\}$;
23. $U := U - \{v\}$;
24. **fim-enquanto**
25. **retorne** S ;

Figura 3.3: Pseudocódigo do procedimento Construcao_Inicial

3.2 Busca Local

Uma vez construída uma solução S mediante o procedimento *Construcao_Inicial*, é realizada a busca local.

procedimento Buca_Local (G , t , c , Sol , β)

1. $Sbest := Sol$;
2. $numV = 0$;
3. **repetir**
4. $S := Sbest$
4. $numV := numV + 1$;
5. $U := V_G$;
6. Remover $\beta \cdot |S|$ elementos aleatoriamente de S ;
7. **para cada** $v \in V_G$
8. $d(v) := d_G(v)$;
9. $k(v) := t(v)$;
10. $N(v) := N_G(v)$;
11. **para cada** $v \in S$
12. **para cada** $u \in N(v)$
13. $k(u) := max(k(u) - 1, 0)$;
14. $d(u) := d(u) - 1$;
15. $N(u) := N(u) - \{v\}$;
16. $U := U - \{v\}$
17. Fazer os passos entre as linhas 7 e 22 de WTSS2
34. **se** $C(S) < C(Sbest)$
35. $Sbest := S$;
36. $numV := 0$;
37. **até** $numV < K$
38. **retorne** $Sbest$;

Figura 3.4: Pseudocódigo do procedimento para a Busca Local.

Esta etapa tem como objetivo conseguir uma melhora na solução atual, analisando soluções "próximas" à solução encontrada. Para isto, são analisados K vizinhos que são obtidos da seguinte forma: são removidos $\beta \cdot |S|$ elementos aleatórios $\beta \in (0,1)$ da solução S e é aplicado um algoritmo composto das linhas 7 e 22 do algoritmo WTSS2 para para reconstruir a solução; e seleciona-se o primeiro vizinho com menor custo que S . Esta ação é feita cada vez

que a nova solução gerada represente uma melhora com relação à solução a partir da qual foi obtida. Na figura 3.4 é apresentado o pseudocódigo do procedimento para a busca local.

4 Experimentos Computacionais

4.1 Comparação entre WTSS e WTSS2.

Como foi comentado na seção 3.1, neste trabalho foi utilizado um algoritmo para gerar soluções iniciais baseado no algoritmo proposto em (Cordasco et al. 2015). Para analisar a influencia de do critério proposto na seção 3.1 foram desenvolvidos um conjunto de experimentos. Os mesmos consistiram em gerar valores para os custos e limites sobre varias das instâncias propostas em (Cordasco et al. 2015) e aplicar os algoritmos WTSS e WTSS2 (variante modificada).

Na tabela 4.1, as colunas 1,2 e 3 representam o nome da base de dados escolhida, número de vértices e número de arestas da base respectivamente. Sobre cada base de dados foram repetidos os experimentos 5 vezes. Neste caso se geraram valores aleatórios para os limites seguindo a regra $t(v) \in [1, d(v)]$ e para os custos $c(v) \in [1,10]$ para todo $v \in V_G$. As colunas 4 e 6 representam o número de nodos infetados que aparecem no conjunto de conversão obtido pelos algoritmos WTSS e WTSS2 respetivamente. Finalmente, as colunas 5 e 7 representam o custo do conjunto de conversão obtido pelos algoritmos WTSS e WTSS2 respetivamente.

	n	**m**	**|S|**	**W(S)**	**|S2|**	**W(S2)**
Power grid	4941	6594	1121	4241	**1068**	**4231**
			1123	4170	**1070**	**4168**
			1098	4279	**1048**	4298
			1110	4303	**1038**	4336
			1112	4203	**1057**	4240
ca-GrQc	5242	14484	1083	4168	**1059**	**4126**
			1069	4035	**1032**	**4003**
			1098	4257	**1059**	**4255**
			1093	4072	**1036**	**3998**
			1099	3965	**1055**	**3926**
ca-HepPh	12008	118489	1695	6616	**1602**	**6367**
			1682	6379	**1575**	**6080**
			1572	5981	**1485**	**5769**
			1655	6266	**1549**	**6029**
			1659	6328	**1540**	**5991**
Facebook	4039	88234	313	1228	**262**	**908**
			333	1302	**294**	**1028**
			306	1235	**273**	**962**
			307	1282	**269**	**1013**
			342	1304	**287**	**942**
BlogCatalog3	10312	333983	91	422	**82**	**378**
			138	623	**111**	**509**
			94	440	**88**	**408**
			129	591	**102**	**490**
			103	480	**91**	**425**

Tabela 4.1. Comparação entre WTSS e WTSS2. $c(v) \in [1,10]$

No caso da tabela 4.2, a regra para gerar os custos foi $c(v)=t(v)$. Em ambas tabelas, em negrito estão os valores para os quais o algoritmo WTSS2 obtém menor valor.

	n	m	|S|	W(S)	|S2|	W(S2)
Power grid	4941	6594	1694	2611	**1292**	**2584**
			1724	2736	**1342**	**2691**
			1699	2664	**1313**	**2614**
			1747	2751	**1354**	**2705**
			1699	2681	**1293**	**2667**
ca-GrQc	5242	14484	1746	4134	**1261**	**3992**
			1667	3816	**1263**	**3659**
			1695	4265	**1318**	**4098**
			1700	3878	**1289**	**3723**
			1718	3904	**1229**	**3664**
ca-HepPh	12008	118489	2134	11813	**1788**	**10690**
			2588	13706	**1783**	**11879**
			2507	17082	**1837**	**14471**
			2807	17664	**1755**	**15463**
			2464	13986	**1920**	**12220**
Facebook	4039	88234	432	9740	**329**	**8168**
			478	8559	**318**	**7381**
			351	7301	**251**	**6401**
			506	12328	**319**	**11656**
			572	7884	**309**	**6913**
BlogCatalog3	10312	333983	1214	22676	**244**	**18144**
			862	18831	**254**	**18631**
			921	22908	**340**	**22327**
			422	9574	**323**	**6973**
			618	14278	**199**	14306

Tabela 4.2. Comparação entre WTSS e WTSS2. $c(v)=t(v)$

Claramente, a variante WTSS2 obtém um melhor desempenho, alcançando menor custo em 88% das vezes na tabela 4.1 e em 96% das vezes na tabela 4.2. Oberva-se que em todos os casos, os conjuntos conseguidos por WTSS2 tem um menor número de nos infectados que os conjuntos obtidos por WTSS. Devido aos resultados alcançados, o algoritmo WTSS2 foi escolhido para gerar as soluções iniciais.

4.2 Seleção dos parâmetros.

Para determinar o valor do parâmetro α utilizado para gerar as soluções iniciais, foi desenvolvido o seguinte experimento. Para cada uma das bases de dados selecionadas, se geraram valores para os limites e custos mediante a regra $t(v)\in[1,d(v)]$ e $c(v)\in[1,10]$. Então foram geradas 50 soluções seguindo o procedimento descrito no algoritmo da figura 3.3. Isto foi repetido 5 vezes para cada base de dados. A tabela 4.3 apresenta os resultados do teste elaborado. A primeira coluna contém o nome da base de dados utilizada. A segunda coluna representa o valor obtido pelo algoritmo WTSS2 sobre a a instância criada na base de dados. As colunas 3-7 representam os valores mínimos que foram obtidos nas 50 soluções geradas com o α que aparece na coluna correspondente. Finalmente, os valores nas colunas 8-12 mostram quantas das 50 soluções são menores que o valor obtido pelo algoritmo WTSS2.

	WTSS2	MIN					<WTSS2				
		1	**0.95**	**0.9**	**0.85**	**0.8**	**1**	**0.95**	**0.9**	**0.85**	**0.8**
Power grid	4180	4159	4167	4158	**4155**	4169	12	**14**	**14**	2	2
	4300	4295	4301	4297	4300	**4290**	**3**	0	1	0	1
	4469	4432	4432	4432	4443	**4430**	**44**	38	42	22	15
	4394	4373	4379	4390	4377	**4361**	**16**	14	4	9	1
	4301	**4256**	4268	4258	4258	4269	26	**36**	26	17	19
Ca-GrQc	4103	4088	4084	4073	4090	**4080**	21	24	**35**	7	18
	4108	4096	4102	4102	**4091**	4104	**4**	**4**	3	2	1
	4064	**4041**	4052	4042	4051	4051	9	6	**13**	6	1
	3957	3925	3924	3928	3925	**3918**	**49**	47	47	13	26
	3986	3977	3979	**3973**	4009	3998	4	**6**	4	0	0
ca-HepPh	6192	6157	6168	6160	**6140**	6175	9	**14**	7	8	1
	5903	5880	5889	**5864**	5893	5880	4	5	**14**	1	2
	5990	5970	5987	**5949**	5999	5983	7	2	**7**	0	1
	5997	5954	**5933**	5946	5937	5940	**48**	**48**	37	30	24
	6323	6287	6279	**6260**	6299	6298	27	20	**29**	6	8

Tabela 4.3. Experimento para determinar o valor de α .

Em todos os casos, as 50 soluções geradas foram diferentes para cada valor de α analisado. No experimento desenvolvido, pode se observar que o valor 0.9 consegue bons resultados para os valores mínimos e consegue varias vezes valores menores que o valor obtido na solução dada. Por esta razão foi escolhido $\alpha = 0.9$ para criar a lista restrita de candidatos usada na geração de soluções iniciais.

No caso do parâmetro β , foi desenvolvido um experimento similar para determinar o valor apropriado. A tabela 4.4 apresenta os resultados sobre as instâncias Power Grid, ca-GrQc e caHepPh. Os limites $t(v)$ de cada vértice $v \in V_G$ foram calculados como um valor aleatório inteiro em $[1, d(v)]$ e para os custos foi utilizado o critério $c(v) \in [1,10]$.

	WTSS2	MIN					<WTSS2				
		0.05	**0.1**	**0.2**	**0.3**	**0.5**	**0.05**	**0.1**	**0.2**	**0.3**	**0.5**
Power grid	4255	4223	4219	**4201**	4188	4183	50	50	50	50	50
	4200	4172	4152	4129	**4112**	4095	49	50	50	50	50
	4203	4175	4167	4138	4122	**4106**	50	50	50	50	50
	4217	4184	4156	4149	4134	**4111**	50	50	50	50	50
	4228	4201	4171	4162	4133	**4125**	49	49	50	50	50
Ca-GrQc	3856	3831	3817	3797	3793	**3785**	50	49	50	50	50
	3936	3897	3870	3842	3812	**3790**	50	50	50	50	50
	4020	3986	3971	3929	3921	**3891**	50	50	50	50	50
	4109	4063	4041	4003	**3971**	3977	49	50	50	50	50
	3999	3955	3940	3906	**3877**	3879	50	50	50	50	50
ca-HepPh	5677	5590	5506	5370	5327	**5254**	50	50	50	50	50
	5977	5868	5797	5648	5584	**5544**	50	50	50	50	50
	6148	6033	5959	5860	5740	**5727**	50	50	50	50	50
	5989	5882	5807	5664	**5593**	5600	50	50	50	50	50
	5918	5804	5729	5593	5494	**5423**	50	50	50	50	50

Tabela 4.4. Experimento para determinar o valor de β .

Uma vez gerado os valores de t e c , são obtidas 50 soluções usando o algoritmo WTSS2 que são destruídas e reconstruidas usando os valores de β : 0.05, 0.1, 0.2, 0.3 e 0.5. Os valores de t e c são mudados em cada uma das 5 linhas correspondentes a cada base de dados. A primeira coluna contém o nome da base de dados. A segunda coluna representa o valor obtido

pelo algoritmo WTSS2. As colunas 3-7 representam os valores mínimos que foram obtidos nas 50 soluções geradas com o valor de β que aparece na coluna correspondente. Finalmente, os valores nas colunas 8-12 mostram quantas das 50 soluções são menores que o valor obtido por o algoritmo WTSS2.

De acordo com os resultados obtidos, o valor escolhido para β foi 0.5. Porém, com todos os valores se consegue diminuir o valor obtido pelo algoritmo WTSS2 em quase a totalidade de soluções encontradas.

4.3 Resultados sobre as bases dados utilizadas

O algoritmo proposto, foi comparado o nosso algoritmo contra o algoritmo GREEDY, que consiste um um procedimento construtivo guloso que em cada iteração seleciona o vértice de menor custo para ser adicionado ao conjunto vértices inicias até garantir a infeção da rede e o algoritmo WTSS.

Em (Zhanh e Sahin 2015) comparam-se os resultados do algoritmo branch-and-cut com o algoritmo GREEDY em 10 execuções sobre instâncias geradas com 20000 arestas e 2500, 5000 e 10000 vértices, usando os critérios $c(v) \in [1,100]$ e $t(v) \in [1, d_G(v)]$, para todo v em V_G . Os tempos dados como limite para executar o procedimento foram de 5 min, 1 hr e 2 hrs respetivamente.O procedimento branch-and-cut melhorou a solução inicial do GREEDY em 30% para as instâncias usadas.

Para realizar a comparação entre o algoritmo desenvolvido neste trabalho e os algoritmos mencionados foram selecionadas 6 bases de dados e geradas 10 instâncias usando os critérios $c(v) \in [1,100]$ e $t(v) \in [1, d_G(v)]$, para todo v em V_G . Para o caso do GRASP, se utilizou como critério de parada o número máximo de iterações $IMAX = 100$ ou o tempo limite $TLIM = 300\, seg$; e no caso da vizinhança foram analisados $K = 2$ vizinhos. Os outros parâmetros se mantiveram como $\alpha = 0.9$ e $\beta = 0.5$.

Na tabela 4.5 são apresentados os dados das bases utilizadas. Nas tabelas 4.6 e 4.7 são apresentados os resultados relacionados com a qualidade das respostas de nosso algoritmo tomando como referencia o algoritmo GREEDY e o algoritmo WTSS respetivamente. O incremento de melhora vai depender claramente da topologia da instância. Em bases de dados como Facebook, que possuem alguns vértices com um grau alto (25% do número de vértices) , só se conseguiu melhorar em média o 16.98% com relação ao GREEDY. De fato, em grafos densos, a estratégia de selecionar o vértice de menor custo poderia obter bons resultados. Interessante o fato de que na base BlogCatalog3 se conseguiu melhorar em media 59.63% em relação ao algoritmo GREEDY, obtendo-se uma melhora máxima de 82,99%.

Nome	Nro. de vértices	Nro. de arestas
Power grid	4941	6594
Ca-GrQc	5242	14484
Facebook	4039	88234
Ca-HepPh	12008	118489
BlogCatalog3	10312	333983
Ca-HepPt	9877	25973

Tabela 4.5. Dados das bases usadas.

	Power grid	Ca-GrQc	Facebook	Ca-HepPh	BlogCatalog3	Ca-HepTh
1	24,99	20,03	28,98	19,38	70,22	22,25
2	25,22	21,14	7,67	16,33	66,44	21,34
3	24,63	23,39	4,72	18,99	59,61	23,70
4	24,35	24,28	23,83	15,71	66,32	22,06
5	24,10	17,92	12,20	20,20	82,99	21,13
6	25,12	20,83	21,21	17,85	40,95	21,49
7	23,73	23,03	6,81	14,66	54,04	23,65
8	22,76	21,55	9,66	20,23	45,58	21,61
9	23,88	22,38	25,98	18,12	48,45	22,22
10	24,05	20,35	28,71	20,73	61,67	21,96
Melh. Med.	24,28	21,49	16,98	18,22	59,63	22,14

Tabela 4.6. Incremento (em porcentagem) de melhora com relação ao algoritmo GREEDY.

	Power grid	Ca-GrQc	Facebook	Ca-HepPh	BlogCatalog3	Ca-HepTh
1	4,39	4,58	35,62	13,23	54,74	6,15
2	3,05	4,81	34,10	9,94	35,49	5,41
3	4,28	4,60	30,75	10,68	47,75	5,42
4	3,16	6,76	34,17	12,60	48,42	4,99
5	3,61	5,02	30,13	10,19	50,12	5,34
6	3,47	5,02	39,42	10,31	42,91	5,34
7	3,74	4,85	27,12	11,39	61,19	5,52
8	4,08	6,02	26,31	12,35	48,90	5,33
9	3,97	4,95	32,50	11,43	42,30	5,12
10	3,47	5,18	29,73	13,24	45,52	6,65
Melh. Med.	3,72	5,18	31,99	11,54	47,73	5,53

Tabela 4.7. Incremento (em porcentagem) de melhora com relação ao algoritmo WTSS.

No caso da comparação com o algoritmo WTSS, os resultados também variam dependendo das instâncias. Novamente com as instâncias de BlogCatalog3 se conseguem as maiores melhoras. Pode se concluir que os resultados variam dependendo da topologia da instância e que o procedimento desenvolvido produz bons resultados utilizando pouco tempo computacional. Em instâncias pequenas como Power grid, Ca-GrQc e Facebook os resultados foram alcançados com um tempo médio 113.00s , 125.07s e 37.08s respetivamente.

5 Conclusões

O trabalho desenvolvido propõe uma heurística GRASP para abordar o problema WTSS. Os experimentos realizados mostraram que o algoritmo proposto obtém bons resultados em comparação com os algoritmos GREEDY e WTSS, conseguindo melhoras máximas de até 82.99% e 61.19% . Foi apresentada também uma variante do algoritmo guloso proposto em (Cordasco et al. 2015) que consegue melhorar a qualidade dos resultados obtidos em 92% das instâncias analisadas.

Referências

- Chen, N. (2009). On the approximability of influence in social networks. *SIAM Journal on Discrete Mathematics*, *23*(3), 1400-1415.
- Chen, W., Wang, Y., & Yang, S. (2009, June). Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 199-208). ACM.
- Cordasco, G., Gargano, L., Rescigno, A. A., & Vaccaro, U. (2015). Optimizing Spread of Influence in Weighted Social Networks via Partial Incentives. *arXiv preprint arXiv:1512.06372*.
- Feo, T. A., & Resende, M. G. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations research letters*, *8*(2), 67-71.
- Resende, M. G., & Ribeiro, C. C. (2010). Greedy randomized adaptive search procedures: Advances, hybridizations, and applications. In *Handbook of metaheuristics* (pp. 283-319). Springer US.
- Zhang, R., & Sahin, M. (2014). Combinatorial Optimization Games Arise in Social Networks.