Large neighbourhood search para o problema do *safe set*

José Paulo de Faria Pedrosa[1], **Edna A. Hoshino**[1], **Vagner Pedrotti**[1]

[1]Faculdade de Computação - Universidade Federal de Mato Grosso do Sul (UFMS)
Campo Grande - MS - Brasil

{jose.paulo.pedrosa, edna.hoshino, vagner.pedrotti}@ufms.br

Abstract. *The safe set problem consists of finding a subset of vertices, whose connected components dominate the adjacent connected components. In this work, we propose a two-phase approach to build solutions for the safe set problem. A initial solution is obtained by a constructive phase and optimized by large neighbourhood search procedures. Computational experiments in instances from the literature show that this matheuristic improves, on average, the value of the solution by 50%.*

Resumo. *O problema do safe set consiste em encontrar um subconjunto de vértices de peso mínimo, que possui componentes conexas que dominam as componentes conexas adjacentes. Neste trabalho, propomos um método de duas fases para construir soluções para o problema do safe set. Uma solução inicial é obtida por uma fase construtiva e otimizada por procedimentos de large neighbourhood search. Experimentos computacionais em instâncias da literatura mostram que a matheurística melhora, em média, em* 50% *o valor da solução.*

1. Introdução

O problema do *safe set* é um problema de particionamento em grafos usado para modelar situações em que é desejável identificar subconjuntos de vértices para ganhar controle de toda a rede. Dado um grafo simples, conexo e não-orientado, com pesos nos vértices, um subconjunto S dos vértices é um ***safe set*** se, para toda componente conexa C induzida por S e toda componente conexa D induzida por $V \setminus S$, nas quais existe uma aresta entre C e D, a soma dos pesos dos vértices em D não excede a soma dos pesos dos vértices em C. O problema do *safe set* (SSP) consiste em encontrar um *safe set* de peso mínimo.

O SSP apresenta várias aplicações práticas, por exemplo, o problema de identificar comunidades influentes em uma rede social e o problema de identificar grupos majoritários que podem criar uma instabilidade em uma rede vulnerável [Bapat et al. 2016]. Nessas aplicações, comunidades são associadas aos vértices de um grafo e os pesos dos vértices referem-se a alguma medida de interesse. Outra aplicação é a alocação de refúgios temporários em grandes edifícios comerciais [Fujita et al. 2016]. Nesse contexto, cada sala segura deve ser acessível aos usuários das salas adjacentes, o que requer que sua capacidade seja pelo menos igual à capacidade das salas conectadas a ela.

Este problema foi introduzido por [Bapat et al. 2016], sendo posteriormente apresentada a sua versão com pesos unitários nos vértices por [Fujita et al. 2016]. Em uma abordagem computacional, [Macambira et al. 2019] propuseram uma formulação linear inteira, resolvida por um algoritmo exato *Branch-and-Cut* com o auxílio de uma heurística gulosa. Já [Hosteins 2020] apresentou um modelo compacto para o problema com

um número polinomial de variáveis e restrições, baseado em fluxo de rede. Mais recentemente, outro algoritmo *Branch-and-Cut* foi proposto por [Malaguti and Pedrotti 2023].

O SSP é um problema NP-difícil [Bapat et al. 2016] e, portanto, é importante construir heurísticas para obter soluções viáveis para o problema em tempo computacionalmente razoável. Este trabalho propõe aplicar técnicas de metaheurísticas baseadas em programação matemática, conhecidas como matheurísticas, para construir soluções viáveis para o SSP. Mais especificamente, analisamos diferentes procedimentos de *large neighbourhood search* (LNS) para otimizar soluções iniciais previamente construídas.

Com esse objetivo, também foi implementado no presente trabalho um algoritmo *Branch-and-Bound*, baseado na formulação linear inteira desenvolvida por [Hosteins 2020], o qual permitiu analisar a viabilidade e a eficiência da aplicação das heurísticas propostas. Testes computacionais foram realizados em instâncias da literatura propostas por [Hosteins 2020], compostas por grafos pseudoaleatório conexos.

2. Descrição formal do problema

Considere um grafo simples, não orientado e conexo $G = (V, E)$. Definimos como $G[S]$ o subgrafo de G induzido por um subconjunto $S \subseteq V$ e denotamos como P_v a componente conexa de $G[S]$ que contém $v \in V$, se $v \in S$; ou a componente conexa de $G[V \setminus S]$ que contém v, se $v \notin S$. O conjunto S é um *safe set* de G se, para cada aresta uv de G, tal que $u \in S$ e $v \notin S$, $|P_u| \geq |P_v|$. Um exemplo de *safe set* está representado na Figura 1.

Para definir o *safe set* com pesos, consideramos a função $w : V \to \mathbb{R}^+$ que atribui um peso positivo a cada vértice $i \in V$. Para qualquer subconjunto S de V, seja $w(S) = \sum_{i \in S} w(i)$. Um subconjunto não vazio $S \subseteq V$ é um *safe set* com pesos se, para cada componente conexa maximal C de $G[S]$ e cada componente conexa maximal D de $G[V \setminus S]$ adjacente a C, a desigualdade $w(C) \geq w(D)$ é satisfeita. O SSP sem e com pesos consiste em encontrar, respectivamente, um *safe set* de cardinalidade mínima e um *safe set* de peso mínimo de um grafo.

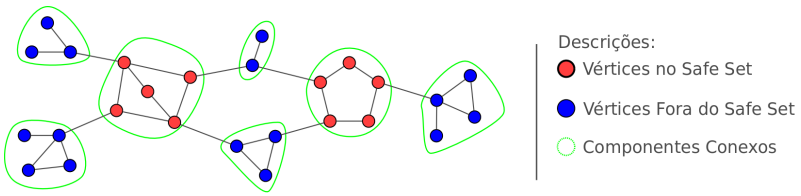

Figura 1. Exemplo de *safe set* viável com pesos unitários.

3. Matheurística proposta

Neste trabalho, propusemos uma matheurística primal, a fim de buscar soluções eficientes para os problemas citados, em tempo computacional razoável. Para tanto, foi implementado um algoritmo de duas etapas. A primeira consiste na criação de soluções iniciais viáveis, a partir de um algoritmo guloso. A segunda, faz a aplicação da matheurística LNS, proposta por [Shaw 1998], para melhorar a solução encontrada na etapa anterior.

Para a primeira etapa, implementamos um algoritmo inspirado na heurística apresentada por [Macambira et al. 2019], que toma como entrada um grafo simples conexo, e, a partir do *safe set* trivial que consiste em todos os vértices do grafo, aplica um processo

iterativo, tentando remover cada um dos vértices do conjunto em uma ordem aleatória sobre uma lista de candidatos restrita, verificando após cada iteração a viabilidade da solução encontrada. Ao final deste processo, é obtida uma solução inicial viável. Todo o procedimento é repetido e a melhor solução encontrada é usada como solução inicial.

O LNS apresenta duas fases: uma de destruição e outra de reparação. Na fase de destruição, remove-se parte da solução, geralmente utilizando algum critério dependente do problema. Na fase de reparação, tenta-se criar uma nova solução, reconstruindo apenas a parte destruída. Esse processo é repetido e a melhor solução é selecionada.

Em nossa proposta, seleciona-se um subconjunto dos vértices do grafo, denominados *destruídos*, que contém tanto vértices do *safe set* como fora dele. Na fase de destruição, removem-se os vértices destruídos do *safe set* inicial, possivelmente tornando-o inválido, enquanto na fase de reparação procura-se uma nova solução adicionando vértices destruídos. O processo é executado resolvendo o modelo de [Hosteins 2020] com as variáveis associadas aos vértices não destruídos são fixadas conforme a solução inicial.

Utilizamos duas estratégias para a escolha de candidatos a serem destruídos para o LNS. A primeira seleciona como candidatos os vértices que se encontram nas bordas das componentes do grafo, como representado na Figura 2. Já a segunda, seleciona aqueles que se encontram fora da borda. Para estar na borda, um vértice pertencente ao *safe set* deve ser adjacente a um vértice não pertencente ao *safe set* ou então, caso não pertença ao *safe set*, deve ser adjacente a um vértice que pertença ao *safe set*.

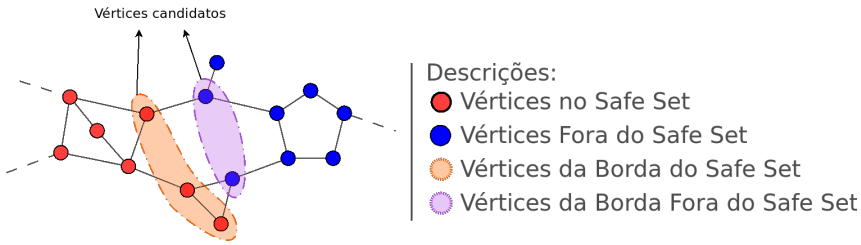

Figura 2. Ilustração da estratégia de vértices da borda no algoritmo LNS.

Em seguida, realizamos um procedimento de escolha dos vértices que serão destruídos dentre os vértices candidatos a partir da formação de listas de candidatos restrita (RCL). Neste caso, pertencerão à RCL os vértices candidatos j que tenham peso $w(j) \geq \alpha w_{\text{max}} + (1-\alpha) w_{\text{min}}$, sendo w_{max} e w_{min}, respectivamente, o maior e menor peso entre os pesos dos elementos da lista de candidatos. Formada a RCL, um de seus vértices é selecionado aleatoriamente para ser retirado da solução. O grau de aleatoriedade é controlado por α, um valor em $[0, 1]$. Em nossos experimentos, adotamos $\alpha = 0,7$.

A formação da RCL e a escolha aleatória do vértice a ser destruído são repetidos até que metade dos vértices candidatos sejam destruídos.

4. Resultados Computacionais

Para analisarmos o desempenho das estratégias escolhidas, foram utilizados 120 grafos pseudoaleatórios conexos, gerados por [Hosteins 2020], os quais possuem entre 20 e 50 vértices e densidade variando entre 10% e 40% do número de arestas do grafo completo.

Todos os testes foram executados em uma máquina com um Intel(R) Core(TM)

i5-10210U (1,60 GHz), 20GB de memória RAM, usando o resolvedor SCIP v.8.1.0, utilizado no modo *single thread*, com tempo máximo de execução limitado a 10s para a etapa de construção da solução inicial e 25s para a execução do LNS. O código-fonte do programa foi escrito em C e compilado com gcc versão 11.4.0.

Foram avaliadas quatro configurações diferentes dos parâmetros. Nas configurações B-1 e F-1, a criação da solução inicial viável é realizada por apenas uma iteração do respectivo algoritmo, enquanto nas configurações F-10 e B-10, seleciona-se a melhor entre 10 iterações. Para o LNS, as configurações B-1 e B-10 utilizam como candidatos os vértices pertencentes à borda e as configurações F-1 e F-10, os vértices fora da borda.

Os resultados estão sumarizados, por densidade das instâncias (d), na Tabela 1, e por tamanho (número de vértices, n), na Tabela 2. As configurações são identificadas pelas colunas I-1, B-1, B-10, F-1 e F-10, sendo que a primeira indica o valor da solução inicial obtida em uma iteração. Os valores apresentados representam a razão entre o acréscimo da solução encontrada em relação à melhor solução obtida por [Hosteins 2020].

Tabela 1. Comparação com o desempenho do algoritmo exato por densidade.

d	I-1	B-1	F-1	B-10	F-10
10%	137,2%	113,2%	108,3%	86,0%	84,4%
20%	82,5%	60,7%	58,6%	42,6%	44,0%
30%	53,2%	32,9%	33,5%	30,2%	25,1%
40%	36,5%	17,5%	18,5%	12,7%	12,6%

Tabela 2. Comparação com o desempenho do algoritmo exato por tamanho.

n	I-1	B-1	F-1	B-10	F-10
20	87,1%	57,4%	66,5%	47,1%	46,3%
25	76,4%	61,6%	55,7%	47,3%	45,4%
30	86,5%	69,2%	66,1%	51,4%	51,4%
35	91,1%	64,5%	60,1%	44,6%	41,0%
40	68,3%	49,7%	48,0%	38,1%	36,3%
50	54,6%	34,0%	32,0%	28,9%	28,8%

Nota-se que as estratégias avaliadas têm uma eficácia maior tanto para instâncias mais densas como para maiores, fornecendo soluções interessantes em uma fração do tempo do algoritmo exato. As configurações com 10 iterações para a solução inicial são notadamente melhores e a seleção de candidatos fora da borda é marginalmente melhor. É significativa também a melhoria que a aplicação do LNS proporciona na solução inicial.

5. Conclusões

Neste trabalho avaliamos o uso do LNS para obter soluções viáveis para o problema do *safe set*. Testes computacionais realizados em instâncias da literatura mostram que o LNS melhora, em média, 50% o valor da solução inicial e que a estratégia que seleciona os vértices fora da borda, usando uma RCL, obtiveram as melhores soluções em 77% das instâncias. Embora, as soluções geradas pela matheurística estejam um pouco distantes das melhores soluções encontradas na literatura, a matheurística se comporta bem nas instâncias maiores e mais densas. Como trabalhos futuros, pretendemos investigar outros critérios para a construção da RCL e outras matheurísticas.

Referências

Bapat, R., Fujita, S., Legay, S., Manoussakis, Y., Matsui, Y., Sakuma, T., and Tuza, Z. (2016). Network majority on tree topological network. *Electron. Notes Discrete Math.*, 54:79–84.

Fujita, S., MacGillivray, G., and Sakuma, T. (2016). Safe set problem on graphs. *Discrete Applied Mathematics*, 215:106–111.

Hosteins, P. (2020). A compact mixed integer linear formulation for safe set problems. *Optimization Letters*, 14:2127–2148.

Macambira, A. F. U., Simonetti, L., Barbalho, H., Gonzalez, P. H., and Maculan, N. (2019). A new formulation for the safe set problem on graphs. *Computers and Operations Research*, 111:346–356.

Malaguti, E. and Pedrotti, V. (2023). Models and algorithms for the weighted safe set problem. *Discrete Applied Mathematics*, 329:23–34.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In Maher, M. and Puget, J.-F., editors, *Principles and Practice of Constraint Programming — CP98*, pages 417–431, Berlin, Heidelberg. Springer.