Breakout Local Search for Heaviest Subgraph Problem*

He Zheng and Jin-Kao Hao*

LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers, France
hzzheng65@gmail.com, jin-kao.hao@univ-angers.fr

Abstract. This paper presents a breakout local search (BLS) heuristic algorithm for solving the heaviest k-subgraph problem - a combinatorial optimization graph problem with various practical applications. BLS explores the search space by alternating iteratively between local search phase and dedicated perturbation strategies. Focusing on the perturbation phase, the algorithm determines its jump magnitude and perturbation type according to the search history to obtain the most appropriate degree of diversification. Computational experiments are performed on a number of large random graphs. The experimental evaluations show that the results obtained by BLS are comparable to, and in most cases superior to, those of the current state-of-the-art approaches.

Keywords: Iterated local search · heaviest subgraph problem · adaptive perturbation.

1 Introduction

Given an edge-weighted undirected graph $G(V, E)$, where V is a set of vertices with $|V| = n$ and E is a set of edges, the Heaviest k-Subgraph Problem (HSP) is to determine a subset U of k vertices (k is given) such that the total edge weight of the subgraph induced by U is maximized. The NP-hard Densest k-Subgraph Problem (DSP), also known as the k-Cluster Problem [4], is a special case of HSP when the edge weight equals one. HSP is a relevant model for many important applications in areas such as social networks, protein interaction graphs, and the world wide web, etc. However, solving the problem is computationally challenging since it generalizes the NP-hard DSP.

Several exact approaches have been proposed to solve the problem [5], but they can only deal with small and sparse graphs with a small range of k. To solve large instances, heuristics and metaheuristics have been used to find approximate solutions in a reasonable time. Macambira proposed a tabu-based heuristic for solving HSP [6], which is based on three construction strategies and a neighborhood search strategy. Brimberg et al. presented a basic variable neighborhood search (BVNS) and some variants of the heuristic for the problem, using the swap neighborhood [3]. Saarinen et al. introduced an opportunistic version of

* We would like to thank the reviewers for their insightful comments. The first author is supported by a CSC scholarship (No. 202306290083).

Algorithm 1 The BLS algorithm for HSP

Input: Edge-weighted graph $G(V, E)$, integer k
Output: The best solution S^* found
$S \leftarrow Initializing(G)$
$S^* \leftarrow S$ and $f(S^*) \leftarrow f(S)$
$L \leftarrow L_{min}$ /* Set initial jump magnitude */
$\omega \leftarrow 0$ /* Set the number for consecutive non-improving local optima */
$prev \leftarrow f(S)$ /* Set the best objective value of the last descent phase */
while stopping condition is not met **do**
    $S \leftarrow LocalSearch(S, H)$ /* H is a vector with historical search information */
    if $f(S) > f(S^*)$ **then**
        $S^* \leftarrow S$ and $f(S^*) = f(S)$
        $\omega \leftarrow 0$
    else if $f(S) \neq prev$ **then**
        $\omega \leftarrow \omega + 1$
    end if
    $L \leftarrow DetermineJumpMagnitude(L, S, \omega, prev)$
    $T \leftarrow DeterminePerturbationType(S, \omega)$
    $prev \leftarrow f(S)$
    $S \leftarrow Perturb(S, L, T, \omega, H)$
end while
return S^*

the VNS heuristic (OVNS) [7], which exploits the characteristics of the problem instance during the search process. These algorithms have improved the state of the art in solving HSP. However, their performance often depends on the instances studied. The VNS algorithms lack stability in their results. BVNS mainly performs well on sparse graphs, while OVNS is more suitable for dense graphs. In this work, we present an effective algorithm for HSP based on breakout local search (BLS).

2 Breakout Local Search for HSP

2.1 General Framework

Breakout local search [1,2] follows the basic scheme of the iterated local search (ILS) approach. In general, BLS repeats a descent-based local search phase to perform an intensive search in a given region, and an adaptive perturbation phase to discover new promising regions. Special attention is paid to the design of the perturbation, which aims to introduce an appropriate degree of diversification according to the search stage. This is achieved by dynamically and adaptively determining the number of perturbation moves (the jump magnitude) and the type of the perturbation moves based on the search information.

The BLS algorithm for HSP (Algorithm 1) starts from an initial solution S given by the *Initializing* procedure and then uses the best-improvement descent *LocalSearch* procedure to attain a local optimum. At this point, BLS

tries to escape from the current optimum by setting the jump magnitude L to an appropriate value and choosing a suitable perturbation type T of a certain intensity, where L and T are determined by $DetermineJumpMagnitude$ and $DeterminePerturbationType$, respectively, based on the search history. The perturbed solution becomes the new starting point for the next search round of the algorithm. This process is repeated until the stopping condition (e.g., time limit, maximum number of iterations...) is met.

2.2 Initial Solution

For a given graph $G(V, E)$ and an integer k, a candidate solution S is represented by a vector of length n, $S = \{x_1, x_2, ..., x_n\}$, where $x_v = 1$ $(1 \leq v \leq n)$ if vertex v is among the k selected vertices in the current solution; $x_v = 0$ otherwise.

For each vertex v, the total weight of the edges from v to all the selected vertices is recorded in $\alpha_v = \sum_{u \in \{i \in V | x_i = 1\}} w_{vu}$, where w_{vu} denotes the weight of edge joining vertices v and u. The vector α is created when constructing an initial solution and is updated each time a move is performed. We get an initial solution of reasonable quality using the drop operator. We start by setting $x_v = 1$ for $v = 1, 2, ..., n$ and compute the α_v value for each vertex v. Then we iteratively drop $(n - k)$ vertices to obtain a solution with k selected vertices, each drop involving the vertex with the smallest α value. We then update the α vector in constant time.

2.3 Local Search

To move from one solution S to another in the search space, BLS uses the popular move operator $Swap(v, u)$, which exchanges a selected vertex v in the solution $(x_v = 1)$ against a non-selected vertex u $(x_u = 0)$. Let $S \oplus Swap(v, u)$ denote the neighboring solution obtained by applying $Swap(v, u)$ to solution S, then the corresponding neighborhood can be defined as $N(S) = \{S \oplus Swap(v, u) : x_v = 1, x_u = 0, 1 \leq v, u \leq n\}$. We use $\delta_{vu} = \alpha_u - \alpha_v - w_{vu}$ to compute the move gain, i.e., the change in the objective function value if vertex v is replaced by u in the solution. Each step of the local search with the best improvement strategy selects, among all neighboring solutions in $N(S)$, the one with the best (largest) move gain δ_{vu}.

2.4 Adaptive Perturbation

Jump magnitude The basic idea of BLS adaptive perturbation is to increase the number of perturbation moves (jump magnitude L) to redirect the search to a new and sufficiently distant area when the search seems to have stalled, as shown in Algorithm 2. The number of perturbation moves is usually set to a small value L_{min} at the beginning of the algorithm or when a new local optimum is found. If L is not large enough to escape the basin of attraction of the current local optimum, L is increased. Otherwise, it is reduced to its initial value L_{min}. If the best solution is not improved for MI successive search rounds, the jump magnitude is set to a large number L_{max} to allow for strong perturbations.

Algorithm 2 DetermineJumpMagnitude($L, S, \omega, prev$)

Input: Current jump magnitude L, local optimum S, history information ω, $prev$
Output: The jump magnitude L for the next perturbation
if $\omega > MI$ **then**
    $L \leftarrow L_{max}$
    $\omega \leftarrow 0$
else if $f(S) = prev$ **then**
    $L \leftarrow L + 1$
else
    $L \leftarrow L_{min}$
end if
return L

Algorithm 3 DeterminePerturbationType(S, ω)

Input: Current local optimum S, constant in (0,1) Q, counter of successive non-improving search rounds ω
Output: The perturbation type T
Determine probability P of directed perturbation considering ω
With probability P, $T \leftarrow DirectedPerturbation$
With probability $(1-P) \cdot Q$, $T \leftarrow RecencyBasedPerturbation$
With probability $(1-P) \cdot (1-Q)$, $T \leftarrow RandomPerturbation$
return T

Three types of perturbation moves To introduce different perturbation intensities, the BLS algorithm adopts three types of perturbations.

- *Directed Perturbation.* The directed perturbation applies a selection rule similar to tabu search, which favors swap moves that cause the least decrease in the objective value, with the constraint that the moves are not forbidden at the current search stage. A forbidden move involves a vertex v such that v has been removed from the solution during the last γ iterations (tabu tenure) (γ takes a random value from a given range related to k).
- *Recency Perturbation.* It relies only on the historical information stored in a vector H that counts the number of times each vertex has been moved during the search. The recency perturbation focuses on the least recently moved vertices, regardless of the objective degradation of the perturbation moves performed.
- *Random Perturbation.* The random perturbation introduces the greatest degree of diversification. It selects the two vertices to be swapped uniformly at random regardless of the objective degradation of the perturbation move.

These three types of perturbations are selected with different probabilities depending on the stage of the search (as shown in Algorithm 3). The number ω of successive non-improving search rounds is used to determine the current search state, which is reset to zero each time the best solution is improved or when ω reaches the maximum bound. Precisely, when ω is small, the search can

go back to the basin of attraction of the current local optimum solution. To avoid this, the directed perturbation is applied with a higher probability. If an increasing ω fails to help the algorithm to escape from the current search region, BLS applies the Recency-based perturbation or the Random perturbation to introduce a strong degree of diversification. According to [1], the probability P of applying the directed perturbation is determined by $P = e^{-\omega/MI}$. Given P, the probability of applying the recency-based perturbation and the random perturbation is $(1-P) \cdot Q$ and $(1-P) \cdot (1-Q)$ with the constant Q in (0,1).

3 Experimental Results

3.1 Test Instances

We used two sets of 129 instances generated from 43 random graphs according to [3] with integer edge weights uniformly taken in the range [100...1000].

- SET I (81 instances). This set contains 81 instances generated from 27 graphs with $|V| = 1000$ vertices, including 16 sparse graphs with an average vertex degree of 10 to 40, incremented by 2, and 11 dense graphs with an average vertex degree of 200 to 400, incremented by 20. k is set to $300, 400, 500$, giving 81 instances (27 graphs $\times$ 3 k values).
- SET II (48 instances). This set has 16 random sparse graphs with $|V| = 3000$ vertices and an average vertex degree of 10 to 40. For each graph, k is set to $900, 1200, 1500$, giving a total of $16 \times 3 = 48$ instances.

3.2 Results

We ran our algorithm and the two best-performing algorithms BVNS [3] and OVNS [7] to solve each instance 5 times (3600s per run). We also ran the Branch and Bound (BB) algorithm of the CPLEX solver once on each instance with a cut-off time of 3600s. Table 1 summarizes the comparison results between BLS and the reference algorithms. $\#win$, $\#ties$, and $\#losses$ respectively denote the number of instances where our BLS algorithm achieves better, equal and worse values compared to the reference algorithms in terms of the best objective values. We also give the deviation dev of each algorithm's average objective value f^{avg} from the best objective value f^* found by all algorithms, defined as $\%dev = (f^* - f^{avg})/f^*$. The results show that BLS always achieves equal or better results compared to BVNS and OVNS, with a clear dominance on sparse graphs. BLS outperforms BVNS on 102 out of the 129 instances, and its results are better than those of OVNS on 87 instances, resulting in the smallest average deviation of 0.02% over all instances against 12.44% for BB, 0.16% for BVNS and 0.11% for OVNS. To check whether the proposed algorithm is statistically better than the reference algorithms, we applied the Wilcoxon signed-rank test with a significance level of 0.05 to the best results of the compared algorithms. The small *p-value* ($\ll 0.05$) confirms that the difference between the results of BLS and those of each reference algorithm is statistically significant.

Table 1. Comparison between BLS and the reference algorithms BB, BVNS and OVNS

Type	k	BLS *vs* BVNS [3]				BLS *vs* OVNS [7]				*%dev*			
		#win	*#ties*	*#losses*	*p-value*	*#win*	*#ties*	*#losses*	*p-value*	BB	BVNS	OVNS	BLS
SET I - sparse	300	**14**	2	0	-	**11**	5	0	-	6.27	0.24	0.12	0.01
	400	**15**	1	0	-	**13**	3	0	-	3.15	0.13	0.08	0.01
	500	**15**	1	0	-	**13**	3	0	-	1.65	0.08	0.05	0.00
SET I - dense	300	**2**	9	0	-	0	11	0	-	18.17	0.05	0.01	0.00
	400	**8**	3	0	-	**3**	8	0	-	13.56	0.02	0.01	0.01
	500	0	11	0	-	**1**	10	0	-	10.18	0.01	0.00	0.00
SET II - sparse	900	**16**	0	0	-	**14**	2	0	-	26.71	0.53	0.34	0.07
	1200	**16**	0	0	-	**16**	0	0	-	18.95	0.22	0.26	0.04
	1500	**16**	0	0	-	**16**	0	0	-	13.29	0.12	0.16	0.02
Total		**102**	27	0	1.85e-18	**87**	42	0	2.48e-15				
Average										12.44	0.16	0.11	0.02

4 Conclusion

A heuristic approach based on breakout local search is developed to solve the NP-hard heaviest k-subgraph problem. The proposed method is characterized by its informed perturbation mechanism, which adaptively chooses between directed, recency-based, and random perturbations to introduce an appropriate degree of diversification at different search stages. Computational results on different types of instances demonstrate the effectiveness of the proposed algorithm. However, in some cases it is still time consuming for the algorithm to obtain high quality solutions. Additional strategies in combination with learning techniques can be explored to improve the method.

References

1. Benlic, U., Hao, J.K.: Breakout local search for the quadratic assignment problem. Applied Mathematics and Computation **219**(9), 4800–4815 (2013)
2. Benlic, U., Hao, J.K.: Breakout local search for the vertex separator problem. In: Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence. pp. 461–467. AAAI Press (2013)
3. Brimberg, J., Mladenović, N., Urošević, D., Ngai, E.: Variable neighborhood search for the heaviest k-subgraph. Computers & Operations Research **36**(11), 2885–2891 (2009)
4. Corneil, D.G., Perl, Y.: Clustering and domination in perfect graphs. Discrete Applied Mathematics **9**(1), 27–39 (1984)
5. Letsios, M., Balalau, O.D., Danisch, M., Orsini, E., Sozio, M.: Finding heaviest k-subgraphs and events in social media. In: 2016 IEEE 16th International Conference on Data Mining Workshops. pp. 113–120. IEEE (2016)
6. Macambira, E.M.: An application of tabu search heuristic for the maximum edge-weighted subgraph problem. Annals of Operations Research **117**, 175–190 (2002)
7. Saarinen, V.P., Chen, T.H.Y., Kivelä, M.: Ovns: Opportunistic variable neighborhood search for heaviest subgraph problem in social networks. arXiv preprint arXiv:2305.19729 (2023)