An Experimental Study of Variable Neighborhood Search for General-Purpose Subgraph Optimization in Parallel Systems

Diogo Carvalho[1], **Mayron Moreira**[1], **Vinícius Dias**[1]

[1]Departamento de Ciência da Computação (DCC/UFLA) – Lavras – MG – Brazil

diogo.carvalho3@estudante.ufla.br, {mayron,viniciusdias}@ufla.br

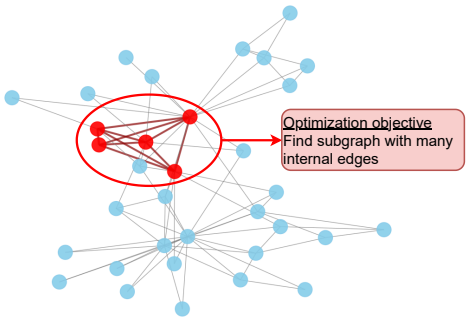
Abstract. *Many network analysis tasks over large graphs involve finding subgraphs that optimize a given scoring function. Scoring functions are defined according to the analysis objective and thus, they can be very diverse (e.g. high density, high reachability, high entropy). In this context we study the Connected Subgraph Optimization Problem (CSOP) in which the goal is to find a subgraph that optimizes a given scoring function within a graph. Such tasks are computationally intensive since (i) they require handling an exponential search space over large graphs and (ii) they involve very skewed and sparse real graphs, which is why algorithm scalability and efficiency are central for feasibility. In this work we provide an experimental characterization of the potential of Variable Neighborhood Search metaheuristic (VNS) in providing a competitive and efficient parallel framework for the CSOP problem. Our results show that VNS offers competitive, near-ideal scalability for accelerating CSOP, but its effectiveness can be limited by expensive scoring functions, branch mispredictions, and both backend and frontend CPU bottlenecks.*

1. Introduction

Many querying tasks on graphs can be modeled as optimization problems, in which the overall goal is to extract substructures of interest (a.k.a. subgraphs). For example, in a social network composed of users (vertices) and friendships among them (edges), finding a subset of vertices with a surprisingly high number of edges may unveil underlying communities or even suspicious groups. In this context, a very common category of tasks is the extraction of optimal connected subgraphs based on a specific scoring (objective) function[1] – we refer to this problem as the *Connected Subgraph Optimization Problem* (see Figure 1). For instance, the density of a subgraph refers to how well-connected a subgraph is but, nevertheless, many measures for density exist, including ratio of edges to vertices [Boob et al. 2020], conductance [Galhotra et al. 2015], to cite a few. Similarly, subgraph exceptionality scores aim to identify subgraphs that deviate from the network's expected behavior in terms of statistical modeling [Hooi et al. 2020], and numerous similarity measures can also be optimized depending on the outlier criterion used. These applications highlight the need for effective subgraph optimization techniques and, moreover, techniques that are not subject to very specific optimization objectives.

The existing literature predominantly focus on specialized methods designed to specific scoring functions. While these methods often demonstrate superior efficiency and efficacy for their intended purpose, database and high-performance systems may benefit

[1]in this work, the terms *scoring function* and *objective function* are used interchangeably



Other examples of objectives:

- Subgraph with many triangles
- Subgraph with low conductance (reachability)
- Subgraph with high entropy (diversity)
- Subgraph with the largest edge weight sum
- Any other scoring function ...

Figure 1. Problem addressed in this work: given graph and optimization objective, find a *high-scoring subgraph* according to this objective.

from more general-purpose approaches capable of representing customized and/or new objectives without significant changes in algorithm design. Such methods could be seamlessly integrated into existing database systems and query languages, offering greater flexibility and broader applicability – one of the major goals of systems research. One important step towards this generalization goal for the aforementioned problem is understanding the potential of existing optimization techniques (in particular, heuristics) in enabling scalability with competitive quality of solutions.

In this work we offer an experimental evaluation of a promising metaheuristic that may be well-suited for the CSOP: the Variable Neighborhood Search (VNS), a local-search based metaheuristic for optimization problems. Our choice of VNS is due to its adaptability to diverse scenarios, its ease for parallelization, and most important, its simplicity in terms of definition, which in turn implies in better experience with programming interfaces. An execution model based on VNS that delivers the best-effort solution within a *defined time budget* would be particularly well-suited for the database and systems communities, addressing a practical need for efficient and adaptable subgraph optimization.

Next we summarize the contributions of this work: (i) a parallel general-purpose framework for evaluating local-search based metaheuristics in the context of subgraph optimization problems; (ii) an experimental characterization of the potential of VNS metaheuristic for a diverse set of subgraph scoring functions. Our results indicate that VNS proves competitive in quality and runtime for accelerating subgraph optimization, demonstrating near-ideal scalability for larger graphs, though performance suffers with expensive scoring functions, and low-level analysis reveals the computation exhibits substantial branch mispredictions. This work is a first step towards high-performance scalable systems designed for CSOP problems.

2. Background

A **graph** G is defined by three sets, $V(G)$, $E(G)$, and $L(G)$, representing respectively the vertices, edges, and labels of G. Each edge $e = (u, v) \in E(G)$ connects a pair of vertices $u, v \in V(G)$. The edges are undirected and contain no *self-loops*. A function f maps each vertex in $V(G)$ to its label in $L(G)$, $f : V(G) \rightarrow L(G)$. Some real-world graphs do not contain labeling information, in which case we assume each vertex is mapped trivially to a single label. We say that S is a **induced subgraph** of G if and only if $V(S) \subseteq V(G)$ and

$E(S) = \{(u, v) \in E(G) \mid u, v \in V(S)\}$. Furthermore, we are interested in *connected subgraphs*, that is, there must exist a path between every pair of vertices in S. Some search applications on graphs benefit from extracting high-scoring subgraphs given a large graph. Many different criteria may exist (e.g. concerning density or sparsity, labeling coverage, entropy, to cite a few) and thus, we define a **subgraph scoring function** as a procedure that quantifies a user-defined interestingness of an induced subgraph of a graph via a real number. The definition of such functions is $score : \{(S, G) \in G \times G \mid S \text{ is an induced connected subgraph of } G\} \rightarrow \mathbb{R}$.

The **Connected Subgraph Optimization Problem (CSOP)** expects as input (i) a single large graph G, (ii) a subgraph scoring function $score$ (both as defined in this section); and asks for an output subgraph that maximizes the scoring function $score$. The search space of this problem is exponential, and even for the scoring functions that admit polynomial time algorithms, such cost may become prohibitive for large graphs (e.g. millions of vertices and/or edges). Next we describe the scoring functions ($score$) used as workload in this work. They were selected based on relevance in the field of data mining and network analysis and on their variety in terms of asymptotic cost and optimization goals.

1. Densest Subgraph (DS). This is the most widely used density measure of induced subgraphs within a graph [Boob et al. 2020]. The ideia is to maximize the ratio of edges in the subgraph to its vertices. Formally: $score(S, G) = \frac{|E(S)|}{|V(S)|}$.

2. Triangle Densest Subgraph (TDS). An alternative formulation for capturing high-density subgraphs is by considering the ratio of triangles (cliques with 3 vertices) in the subgraph to its vertices. Notice that this density measure directly captures the strength of a well-studied property in large networks, the *triadic closure*. Formally: $score(S, G) = \frac{|t(S)|}{|V(S)|}$, where $t(S)$ represent the number of unique triangles in S.

3. Conductance (CO). The conductance of a subgraph [Galhotra et al. 2015] captures how more internally connected a subgraph may be in comparison with its surroundings. Originally, the goal is to find subgraphs of low conductance, but we adapt this measure for a maximization objective. Formally: $score(S, G) = 1 - \frac{|e(S)|}{|i(S)+e(S)|}$, where $i(S)$ and $e(S)$ represent respectively edges of G involving only vertices in S (internal) and edges of G with one end in S and the other end outside S (external).

4. Degree Entropy (DE). Roughly speaking, entropy in the context of theory of information is used to quantify how much information (e.g. number of bits) an entity carries. Thus, degree entropy in subgraphs is a measure that captures how diverse a subgraph is in terms of the degree of its vertices. Therefore, a subgraph with high degree entropy has a larger quantity of vertex degree numbers. Formally: $score(S, G) = -\sum_{d \in ds(S)} p_d \log_2(p_d)$, where $ds(S)$ is the set of unique degrees in $V(S)$ and p_d is the proportion of vertices in $V(S)$ that have degree d.

5. Label Entropy (LE). The entropy of a subgraph w.r.t. labels is defined similarly as degree entropy, the only difference is that we must now take into consideration the number of *different labels in the subgraph and their frequencies*. Formally: $score(S, G) = -\sum_{d \in ls(S)} p_l \log_2(p_l)$, where $ls(S)$ is the set of unique labels in $V(S)$ and p_l is the proportion of vertices in $V(S)$ that have label l.

3. Related work

A common instance of the CSOP is the densest subgraph problem, in which the goal is to find a subgraph that maximizes the ratio of the number of edges by the number of vertices. This is a well-studied problem, with state-of-the-art strategies based on greedy peeling [Boob et al. 2020] and quadratic programming [Elfarouk et al. 2022]. Other variants include density with edge color constraints [Oettershagen et al. 2024], density based on triadic closures [Wickrama Arachchi et al. 2024], density anchored on specific nodes or regions [Ye et al. 2024] , parameterized and general density measures [Xu et al. 2023, Jiang et al. 2025], to cite a few. Many other scoring functions for subgraphs exist, each with very particular goals. Subgraph extraction for anomaly detection [Hooi et al. 2020], maximum edge weight sum subgraph (heaviest) with k vertices [Letsios et al. 2016], and diversified query generation with fairness [Ma et al. 2022] represent other examples that involve *finding high-scoring subgraphs*. Overall, whether based on density or other criteria, an important observation is that each of these problems rely on algorithms and heuristics specifically tailored for their respective objective function. In this work, our goal is to study these problems from a perspective of a general-purpose model, capable of modeling customized objective functions.

Variable neighborhood search (VNS) was introduced by [Mladenović and Hansen 1997]. Since then, several studies have applied this metaheuristic to solve graph-based problems such as facility location variants [Chagas et al. 2024], clustering problems [Zhao and Hifi 2025], and vehicle routing extensions [Sze et al. 2024]. Although the literature often encourages parallelization schemes [Crainic 2018], these approaches are not yet widely adopted among researchers in operations research. For some applications of parallelization, see [Xavier 2019]. Given this context, the development of general algorithmic solutions remains a challenge for researchers. Building on the need for broader adoption of parallelization in VNS, one of the contributions of this study is the proposition of an optimization framework for neighborhood-based metaheuristics in subgraph problems. As far as we know, high-level coding proposals remain a cutting-edge subject, attested by [Song and Wu 2025]. Therefore, our study aims to bridge the gap between VNS parallel and distributed implementations and their broader applications.

4. Evaluation framework

Our evaluation framework (Figure 2) is built on top of the Fractal system [Dias et al. 2019], a parallel graph pattern mining system, allowing us to scale VSN processing to multiple execution threads. The input is composed of: (1) the input graph; (2) the user-defined objective function; (3) the tentative number of initial solutions to be issued; (4) the number of execution threads to be used; and (5) the timeout for each parallel VNS run. Via Fractal parallel programming, multiple initial solutions are generated in parallel by each thread and forwarded to the VNS run module responsible for exploring the space of neighboring solutions in search for improvement. Although the stochasticity and graph topology-dependency of our initial solution generation can not guarantee an exact number of initial solutions, the effective number is very close to the target parameter (IS) in our evaluation scenarios. From each VNS run, a best solution is selected and the framework outputs the best solution across all threads. Next we detail each module.

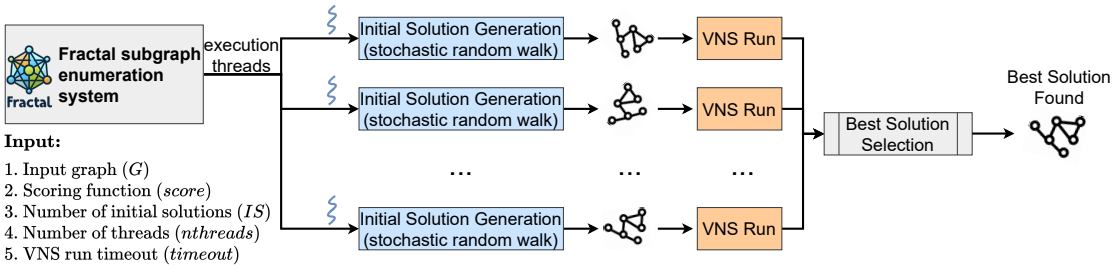


Figure 2. Evaluation framework implementing the VNS metaheuristic.

Initial solution generation. An initial solution for the CSOP is a connected induced subgraph with k vertices, generated via a random walk. Specifically, the algorithm starts at a random vertex and iteratively adds new random neighbors to the path until k vertices are selected. Then, from a simple path with k vertices, the induced subgraph is built by adding all remaining edges in the graph that exist among these vertices. This is the most straightforward approach for ensuring efficient initial solution generation and *reasonable diversity given that scoring functions may have customized objectives*. For simplicity, in this work we set $k = 10$ for all executions and leave the study of the impact of k as future work. We highlight although that this fixed parameter does not limit the output size of the best solution, as ensured by the neighborhood functions and detailed next.

VNS solution improvement (Algorithm 1). An initial solution is improved through successive iterations, each composed of a diversification phase (SHAKE), which perturbs the current solution to escape local optima, and a solution improvement phase (LOCAL-SEARCH), which explores its neighborhood for improvements. These phases rely on neighborhood functions, i.e., rules that determine how to visit a novel solution from the current one. Three general-purpose functions are included as $neighborhoods$: (1) `vertex-add`, which adds a new vertex to the current subgraph extracted from its immediate neighborhood; (2) `vertex-remove`, which removes a vertex from the current subgraph if its removal does not disconnect the current subgraph; and (3) `vertex-swap`, which swaps one vertex within the current subgraph with a new vertex in its neighborhood, always ensuring that the resulting subgraph remains connected. Combined, these neighborhoods enable systematic exploration of the space of connected subgraphs.

Algorithm 1 VNS-RUN($G, S_0, score, timeout, neighborhoods$)

$S \leftarrow S_0$; $S^* \leftarrow S$
while ELAPSED-TIME() $< timeout$ **do**
  $i \leftarrow 0$
  while $i < neighborhoods.size()$ **do**
    $S' \leftarrow$ SHAKE($S, neighborhoods[i]$) ▷ random neighbor (exploration)
    $S \leftarrow$ LOCAL-SEARCH($S', score, neighborhoods[i]$) ▷ locally optimal (exploitation)
    if $score(S, G) > score(S^*, G)$ **then** $S^* \leftarrow S$; $i \leftarrow 0$ **else** $i \leftarrow i + 1$
return S^*

5. Experimental study

All experiments, unless otherwise specified, were run on a machine having one CPU Intel(R) Core(TM) i9-14900KF 6GHz (32 cores), 64GB RAM, running Debian GNU/Linux 12 (Linux kernel 6.1.0-28-amd64). All the algorithms were

implemented within the Fractal framework [Dias et al. 2019]. The datasets used in our evaluation (Table 1) have been widely used previously to evaluate graph algorithms and systems [Teixeira et al. 2015, Jamshidi et al. 2020, Chen et al. 2018, Mawhirter and Wu 2019]. Amazon [Yang and Leskovec 2012] models frequently co-purchased items in the platform. Citeseer [Elseidy et al. 2014] is composed of Computer Science articles and citations among them. DBLP is a co-authorship network extracted using a public community parser[2]. Patents [Hall et al. 2001] models the citations of patents published in the US. LiveJournal [Yang and Leskovec 2012] model friendship in social networks, and Youtube [Cheng et al. 2008] models posted videos and how they are related. In our experiments all graphs are loaded into the main memory using a compressed sparse row graph representation (CSR). We consider the following parameter variation for the evaluated framework: graphs from Table 1, scoring functions from Section 2, number of initial solutions in $\{10^2, 10^3, 10^4\}$, number of threads from 1 to 32, VNS run timeout in $\{1, 2, 3\}$ seconds. The results are presented with a confidence interval of 95%. The source code, data, and reproducibility details are made publicly available[3].

Table 1. Real-world datasets used in the experiments.

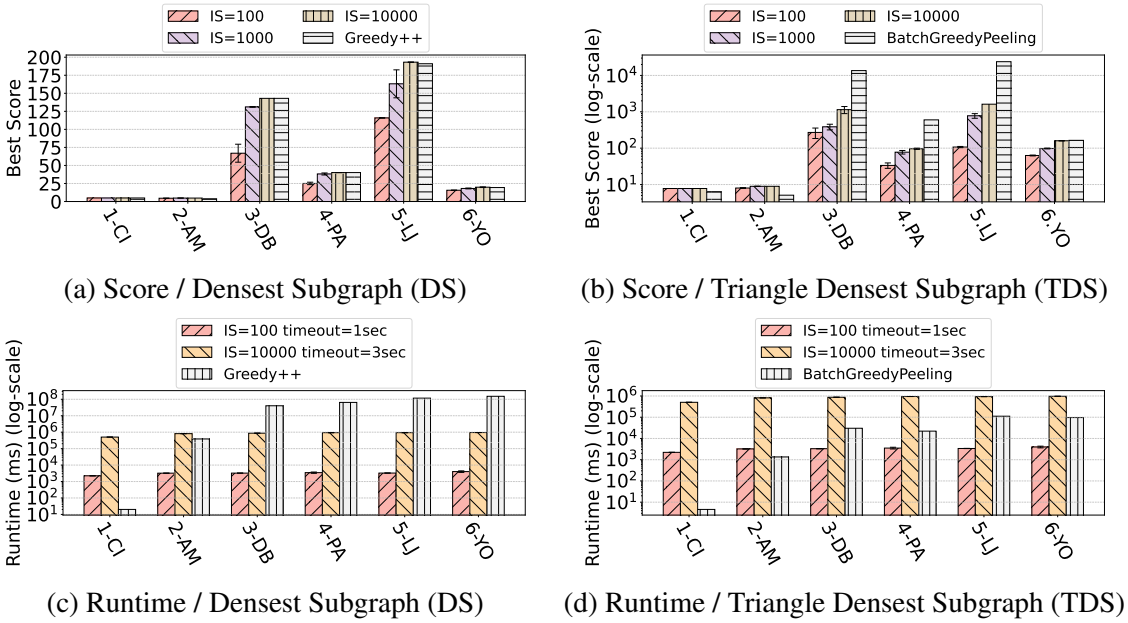
	Num. Vertices	Num. Edges	Max. Deg.	Avg. Deg.	Labels
Citeseer (CI)	3,312	4,536	99	2.7	6
Amazon (AM)	334,863	925,872	549	5.5	-
DBLP (DB)	2,414,634	13,246,813	3,420	10.9	-
Patents (PA)	2,745,761	13,965,409	789	10.1	37
LiveJournal (LJ)	3,997,962	34,681,189	14,815	17.3	-
Youtube (YO)	4,589,876	43,968,798	2,539	19.1	108

5.1. Optimality and Runtime: the quality of solutions

In this section we evaluate the potential of the parallel VNS framework in finding high-quality solutions at a reasonable time. For objectives DS and TDS, we compare VNS against state-of-the-art greedy specialized algorithms: Greedy++ [Boob et al. 2020] for DS and BatchGreedyPeeling [Tsourakakis 2014] for TDS. For the other objectives, we provide known upper theoretical bounds. Figures 3 and 4 show the average best score obtained against baselines (greedy or upper bound), considering all VNS timeout values (1, 2, and 3 seconds) and grouped by number of initial solutions. Compared to both greedy and upper bound baselines, in most cases VNS demonstrates competitive results in terms of quality, especially in experiments with 10^4 initial solutions. For the DS problem, the results were comparable to or superior than Greedy++, outperforming the baseline in larger graphs (Figure 3c). The experiments for the objective functions LE and CO yield solutions equivalent to or close to the upper bound. For the DE score function, our algorithm generated solutions substantially inferior to the upper bound - an expected outcome, given that this theoretical limit is not always achievable in practice. However, when the score function is computationally expensive (TDS), the search space remains under-explored, leading to results of lower quality compared to the baseline (BatchGreedyPeeling). Overall, we identify great potential but also challenges in adopting VNS, especially concerning proper parametrization and coupling with expensive objective functions.

[2] `https://github.com/anderson-pids/dataset-dblp_parser`

[3] `https://github.com/dcc-ufla-graph-mining/fractal/tree/sscad2025`



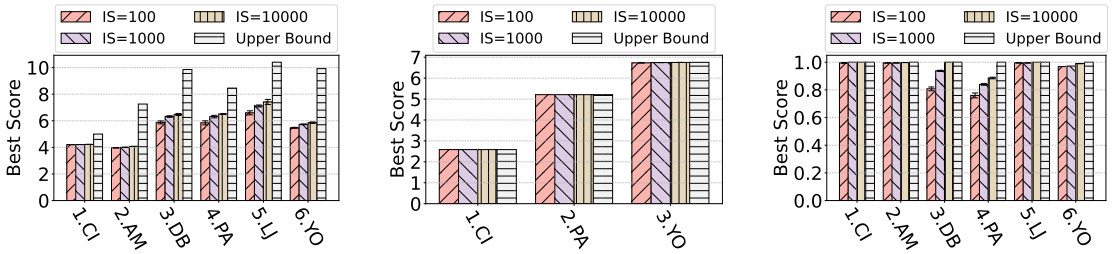
(a) Score / Densest Subgraph (DS)

(b) Score / Triangle Densest Subgraph (TDS)

(c) Runtime / Densest Subgraph (DS)

(d) Runtime / Triangle Densest Subgraph (TDS)

Figure 3. Optimality and Runtime: comparison with greedy baselines.



(a) Degree Entropy (DE)

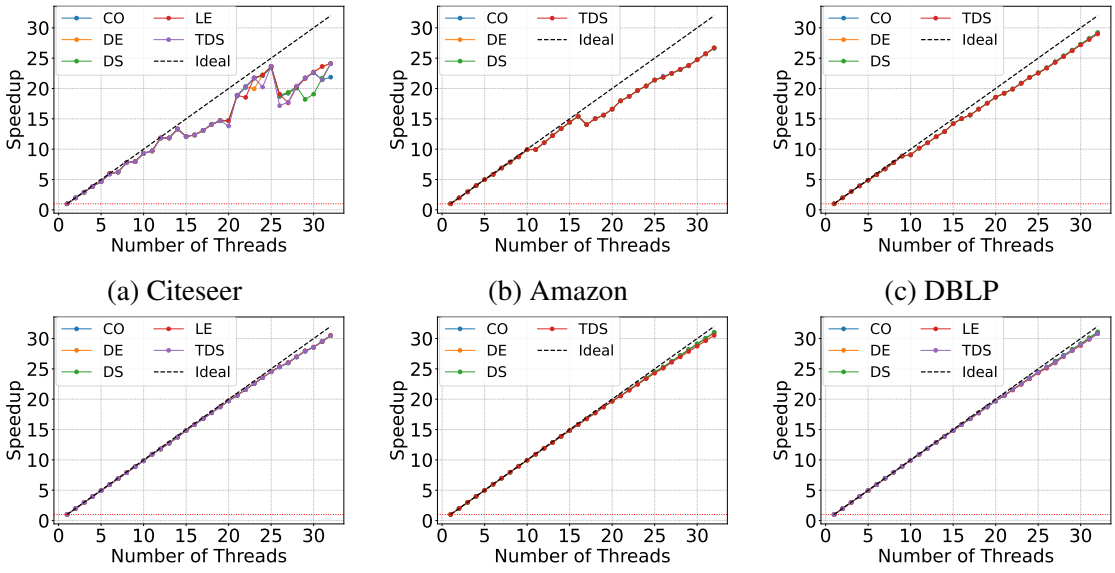
(b) Label Entropy (LE)

(c) Conductance (CO)

Figure 4. Optimality: comparison with known upper bound baselines.

5.2. Strong scalability analysis

This section evaluates the strong scalability of the VNS framework with respect to the *runtime per VNS run*, since our stochastic approach can not guarantee an exact number of initial solution and can be influenced by the graph topology. Figure 5 shows the results. In our experiments, we varied the thread count from 1 to 32. Most notably, our larger graphs exhibit near-ideal scaling behavior across the entire thread range (1-32) for all objective functions (DS, TDS, DE, LE and CO) - a result that highlights VNS's efficiency when processing dense, computationally intensive workloads. Smaller datasets (Citeseer) exhibit suboptimal scaling at higher thread counts. This behavior is expected, as sparse graphs typically contain numerous low-density regions, leading to thread under-utilization due to insufficient parallel workload. The Amazon dataset demonstrates excellent scaling up to 15 threads, after which its efficiency begins to degrade - though it still maintains significantly better performance than Citeseer across all thread counts. This consistent scaling behavior demonstrates that VNS can effectively utilize available computational resources regardless of the optimization objective being pursued. These results suggest that VNS is particularly well-suited as a general-purpose solution for subgraph optimization problems, particularly in environments where parallelization is critical.



(a) Citeseer (b) Amazon (c) DBLP

(d) Patents (e) LiveJournal (f) Youtube

Figure 5. Speedup considering a VNS timeout of 3 seconds.

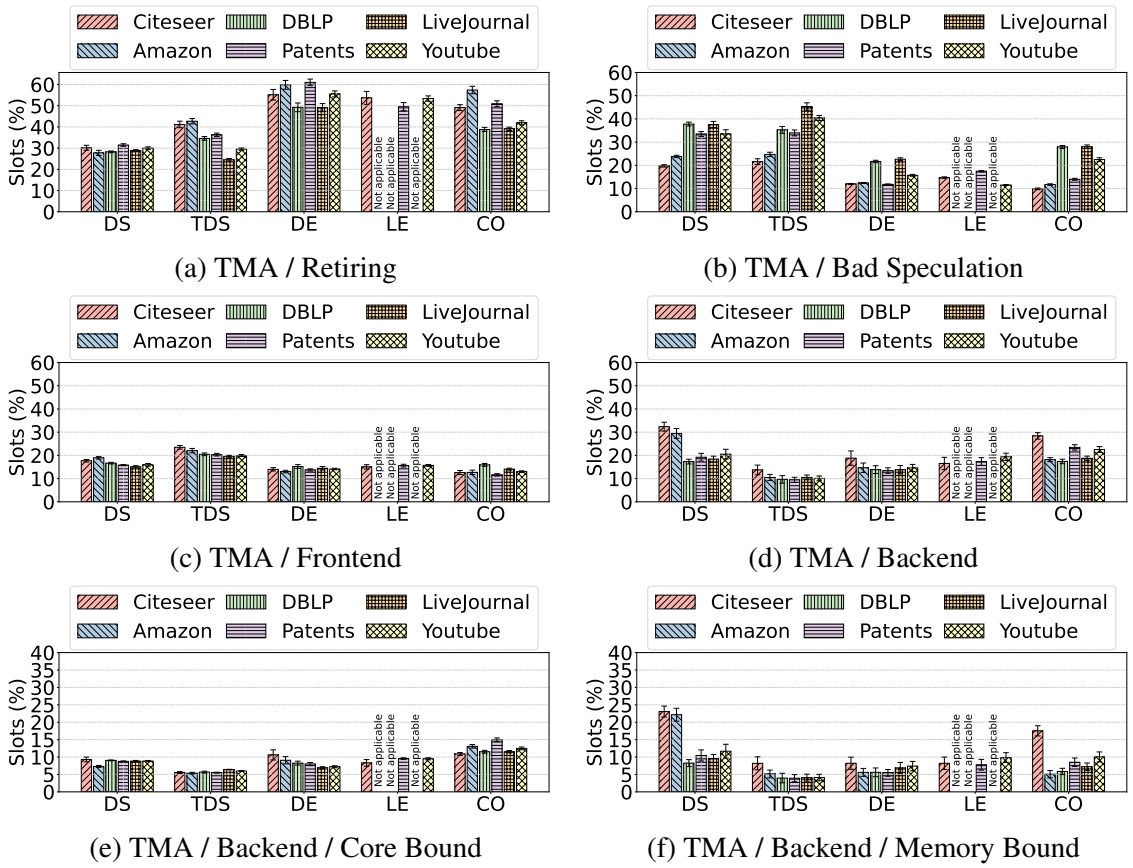
5.3. Understanding CPU efficiency and bottlenecks

The Top-down Microarchitecture Analysis (TMA) methodology [Yasin 2014] is a hierarchical framework for identifying performance bottlenecks in modern processors. TMA provides a structured approach to comprehending how *slots* of a CPU's execution are utilized, enabling a precise diagnosis of the root causes of performance limitations. At its highest level (top level), TMA classifies all available pipeline slots into four categories. *Retiring*, the fraction of slots successfully utilized by operations that complete execution and are retired – this category directly correlates with the Instructions Per Cycle (IPC) metric, signifying efficient core utilization and thus, higher values are preferred. The next three top-level categories are all bottlenecks of slots that do not retired and thus, lower values are preferred. *Bad Speculation*, quantifying the slots wasted due to incorrect speculative execution such as branch mispredictions. *Frontend Bound*, indicating pipeline stalls where the Frontend (instruction fetch and decode pipeline stages) fails to adequately supply the Backend (execution units) with operations. *Backend Bound*, denoting stalled slots resulting from a lack of available resources within the Backend. This category can be further detailed into: *memory bound*, execution stalls attributable to the memory subsystem (e.g. cache misses, contention); and *core bound*, indicating limitations within the core's execution units (e.g. execution ports are saturated). In our experiments, TMA analysis of *performance cores* were extracted via `perf`[4] monitoring Linux tool.

Figure 6 illustrates the results. We observe a growing tendency for bad speculation as graph size increases, with DBLP and LiveJournal showing the highest values – likely due to their highly imbalanced degree distribution. Backend stalls are slightly more substantial than frontend stalls for most scoring functions, with an average differential of only 4.35% (excluding TDS). A notable exception is TDS, which exhibits substantially higher frontend pressure – about 10% greater than backend stalls – consistent with its higher

[4] `https://perfwiki.github.io/main/`

computational complexity. Frontend stalls appear unaffected by the input graph but are influenced by the scoring function, reaffirming the latter's impact on the overall results. Finally, memory- and core-bound stalls show comparable relevance across most configurations, reflecting distinct hardware stress patterns shaped by both the scoring function and input graph. Overall, we confirm that branch misprediction is the primary bottleneck and hence, prioritizing techniques to mitigate conditional branching, very present in VNS especially within the randomized SHAKE procedure, can significantly enhance CPU performance for those workloads.



(a) TMA / Retiring

(b) TMA / Bad Speculation

(c) TMA / Frontend

(d) TMA / Backend

(e) TMA / Backend / Core Bound

(f) TMA / Backend / Memory Bound

Figure 6. CPU bottlenecks via Top-down Microarchitecture Analysis (TMA).

5.4. Impact of scoring function

In this section we study the impact of the scoring functions to the overall runtime via low-level lightweight profiling. Table 2 presents the experimental results. As anticipated, the choice of scoring function significantly affects execution time. The profiling analysis reveals particularly substantial variations in computational overhead across different functions. For elementary objective functions, the computational overhead remains minimal - the DS function accounted for less than 1% of total execution time across all graph sizes. In contrast, more complex functions require considerably greater processing time. The CO function consumed less than 11% of total time for large graphs, but approximately 20% for smaller graphs (Citeseer and Amazon). The entropy functions (DE and LE) demonstrated even greater impact, comprising 40-80% of total execution time. Most notably, TDS emerged as the most computationally demanding scoring function, requiring

over 70% of runtime for smaller graphs and nearly 90% for larger graphs. These findings underscore the critical importance of proper parameter configuration (timeout duration and number of initial solutions), particularly when using costly scoring functions like TDS. The substantial computational overhead of such functions directly limits the number of solutions a VNS run can evaluate, thereby constraining search-space exploration. In these cases, employing approximate scoring functions may prove essential to maintain adequate solution diversity and exploration capability.

Table 2. Profiling of Scoring Functions. This experiment indicates the percentage of running time (%) spent on running the subgraph scoring function.

	Citeseer	Amazon	DBLP	Patents	LiveJournal	Youtube
DS	0.65±0.05	0.59±0.04	0.28±0.05	0.57±0.05	0.19±0.03	0.41±0.1
TDS	71.53±1.91	77.51±1.83	85.33±1.43	89.05±1.52	90.51±1.8	93.97±0.33
DE	49.84±5.83	68.07±2.67	53.86±1.93	44.41±9.86	49.75±3.18	67.93±0.99
LE	55.59±2.06	-	-	62.44±1.29	-	76.24±1.24
CO	17.33±4.13	23.34±5.29	4.59±0.19	8.2±0.28	4.28±0.17	10.46±5.62

Missing entries (-) are due to Label Entropy (LE) not applicable for unlabeled graphs

6. Conclusion

In this work we study how well-suited the VNS metaheuristic can be for accelerating subgraph optimization problems in parallel systems. We built a parallel VNS framework and evaluated its performance against greedy baselines and known upper bounds in multiple real-world scenarios. Our results reveal that VNS is competitive in quality and runtime against competitors and that scales close to ideal for larger graphs. Exceptions are observed for very expensive scoring functions, that must be evaluated frequently, which hurts solutions quality and exploration diversity – observation confirmed via profiling. Finally, low-level CPU analysis is provided, revealing moderate backend- and frontend-related bottlenecks, along with substantial branch mispredictions.

We identify many fronts for future work. First, our initial solution generation is very straightforward and we would like explore approaches more adaptable to different scoring functions. The same reasoning may be applied to search space exploration and neighborhood functions. Second, our experimental observations suggest that the time required for evaluating the score function repeatedly can be a major impact on the solution quality and hence, we plan to investigate learning strategies for *approximate scoring function values* and mitigate this excessive computational cost. Third, we see an opportunity to extend this work to larger, more massively parallel, and distributed settings to study the scalability of such an approach of more challenging problems and instances.

7. Acknowledgment

This work was partially supported by Fapemig (FAPEMIG-UNIVERSAL, APQ-00202-24).

References

Boob, D., Gao, Y., Peng, R., Sawlani, S., Tsourakakis, C., Wang, D., and Wang, J. (2020). Flowless: Extracting densest subgraphs without flow computations. In *Proceedings of The Web Conference 2020*, WWW '20, page 573–583, New York, NY, USA. Association for Computing Machinery.

Chagas, G. O., Lorena, L. A., dos Santos, R. D., Renaud, J., and Coelho, L. C. (2024). A parallel variable neighborhood search for α-neighbor facility location problems. *Computers & Operations Research*, 165:106589.

Chen, H., Liu, M., Zhao, Y., Yan, X., Yan, D., and Cheng, J. (2018). G-miner: An efficient task-oriented graph mining system. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18.

Cheng, X., Dale, C., and Liu, J. (2008). Dataset for "statistics and social network of youtube videos". `http://netsg.cs.sfu.ca/youtubedata/`.

Crainic, T. (2018). Parallel metaheuristics and cooperative search. In *Handbook of metaheuristics*, pages 419–451. Springer.

Dias, V., Teixeira, C. H. C., Guedes, D., Meira Jr., W., and Parthasarathy, S. (2019). Fractal: A general-purpose graph pattern mining system. In *Proceedings of the 2019 International Conference on Management of Data (SIGMOD)*.

Elfarouk, H., Kent, Q., and Chandra, C. (2022). Faster and scalable algorithms for densest subgraph and decomposition. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

Elseidy, M., Abdelhamid, E., Skiadopoulos, S., and Kalnis, P. (2014). Grami: Frequent subgraph and pattern mining in a single large graph. *Proc. VLDB Endow.*, 7(7):517–528.

Galhotra, S., Bagchi, A., Bedathur, S., Ramanath, M., and Jain, V. (2015). Tracking the conductance of rapidly evolving topic-subgraphs. *Proc. VLDB Endow.*, 8(13):2170–2181.

Hall, B. H., Jaffe, A. B., and Trajtenberg, M. (2001). The nber patent citation data file: Lessons, insights and methodological tools. Working Paper 8498, National Bureau of Economic Research.

Hooi, B., Shin, K., Lamba, H., and Faloutsos, C. (2020). Telltail: Fast scoring and detection of dense subgraphs. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):4150–4157.

Jamshidi, K., Mahadasa, R., and Vora, K. (2020). Peregrine: A pattern-aware graph mining system. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA. Association for Computing Machinery.

Jiang, J., Yao, S., Li, Y., Wang, Q., He, B., and Chen, M. (2025). Dupin: A parallel framework for densest subgraph discovery in fraud detection on massive graphs. *Proc. ACM Manag. Data*, 3(3).

Letsios, M., Balalau, O. D., Danisch, M., Orsini, E., and Sozio, M. (2016). Finding heaviest k-subgraphs and events in social media. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 113–120.

Ma, H., Guan, S., Toomey, C., and Wu, Y. (2022). Diversified subgraph query generation with group fairness. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, WSDM '22, page 686–694, New York, NY, USA. Association for Computing Machinery.

Mawhirter, D. and Wu, B. (2019). Automine: Harmonizing high-level abstraction and high performance for graph mining. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP '19, pages 509–523, New York, NY, USA. ACM.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100.

Oettershagen, L., Wang, H., and Gionis, A. (2024). Finding densest subgraphs with edge-color constraints. In *Proceedings of the ACM Web Conference 2024*, WWW '24, page 936–947, New York, NY, USA. Association for Computing Machinery.

Song, A. and Wu, G. (2025). Rems: a unified solution representation, problem modeling and metaheuristic algorithm design for general combinatorial optimization problems. *arXiv preprint arXiv:2505.17108*.

Sze, J. F., Salhi, S., and Wassan, N. (2024). An adaptive variable neighbourhood search approach for the dynamic vehicle routing problem. *Computers & Operations Research*, 164:106531.

Teixeira, C. H. C., Fonseca, A. J., Serafini, M., Siganos, G., Zaki, M. J., and Aboulnaga, A. (2015). Arabesque: a system for distributed graph mining. In *Proceedings of the 25th Symposium on Operating Systems Principles*, SOSP '15. ACM.

Tsourakakis, C. E. (2014). A novel approach to finding near-cliques: The triangle-densest subgraph problem. *CoRR*, abs/1405.1477.

Wickrama Arachchi, C., Kumpulainen, I., and Tatti, N. (2024). Dense subgraph discovery meets strong triadic closure. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 3334–3344, New York, NY, USA. Association for Computing Machinery.

Xavier, E. (2019). Uma interface de programação de aplicações para o brkga na plataforma cuda. In *Anais do XX Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 13–24, Porto Alegre, RS, Brasil. SBC.

Xu, Y., Ma, C., Fang, Y., and Bao, Z. (2023). Efficient and effective algorithms for generalized densest subgraph discovery. *Proc. ACM Manag. Data*, 1(2).

Yang, J. and Leskovec, J. (2012). Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, MDS '12, New York, NY, USA. Association for Computing Machinery.

Yasin, A. (2014). A top-down method for performance analysis and counters architecture. In *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 35–44.

Ye, X., Li, R.-H., Liang, L., Liu, Z., Lin, L., and Wang, G. (2024). Efficient and effective anchored densest subgraph search: A convex-programming based approach. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 3907–3918, New York, NY, USA. Association for Computing Machinery.

Zhao, J. and Hifi, M. (2025). Reinforcement learning-enhanced variable neighborhood search strategies for the k-clustering minimum biclique completion problem. *Computers & Operations Research*, 178:107008.