ARS COMBINATORIA 167 (2026) 195–211

Ars Combinatoria
www.combinatorialpress.com/ars

Combinatorial Press

A branch and bound algorithm for improving the Upper Bound on the edge irregularity strength of complete graph

Muhammad Shahzad, Muhammad Ahsan Asim*, Roslan Hasni and Ali Ahmad

ABSTRACT

This article studies edge irregular (k)-labelings of complete graphs (K_n) and aims to improve the existing upper bound for the edge irregularity strength $(es(K_n))$ through an algorithmic approach. The improvement is achieved using the branch and bound algorithm design strategy, whose selection is important because of the problem structure, computational complexity, possible serial or parallel execution, and accuracy requirements. Labeling complete graphs is difficult because the number of edges grows rapidly and many triangles are involved, making it challenging to maintain unique edge weights while searching for optimal labels. Since complete graphs serve as supergraphs for many graph families, results on them are particularly valuable. In 2018, Asim et al. proposed an algorithmic solution for complete graphs and gave the upper bound $(es(K_n) \leq E \log_2|V|)$. This article uses the branch and bound strategy to address edge deficiency and improve that bound. Computational experiments are conducted for higher-order graphs, where the algorithm recursively generates constrained combinatorial structures to ensure unique edge weights. The results show that the proposed branch and bound algorithm significantly improves the upper bound for $(es(K_n))$ compared with previous results.

Keywords: edge irregular k-labeling, edge irregularity strength, complete graphs, branch and bound algorithm, algorithmic efficiency, edge weights, recursive approach

2020 Mathematics Subject Classification: 05C78, 05C85.

* Corresponding author.

Received 29 Jun 2025; Revised 29 Dec 2025; Accepted 12 May 2026; Published Online 17 June 2026.
DOI: 10.61091/ars167-13

1. Introduction

Algorithm design and optimization play an important role in increasing computational efficiency, reducing processing cost and uses memory efficiently. Optimization consists of tweaking algorithm design strategies to improve performance, reduce computational complexity, and ensure scalability. By proper use of these strategies, we can achieve significant improvements in different types of applications, from data processing to problem-solving in diverse fields.

Graph labeling is a critical area of study in computer science and mathematics by focusing on assigning integer number to the edges or vertices or both of a graph according to certain rules. Consider G is a simple, undirected graph consists of a vertex set, $V(G)$, and an edge set, $E(G)$. When referring to labeling, we denote any mapping that assigns a set of integer numbers, known as labels, to the vertices or edges of the graph. Over the past decades, interest in graph labeling has grown substantially, with more than 200 labeling methods introduced and over 3000 research works published in the field [19].

Chartrand et al. [15] introduced the concept of edge k-labeling δ of a graph G such that $w_\delta(x) \neq w_\delta(y)$ for all distinct vertices $x, y \in V(G)$. Such a labeling is called an irregular assignment, and the irregularity strength $s(G)$ of a graph G is defined as the smallest k for which G admits an irregular labeling using labels from $\{1, 2, \ldots, k\}$. This parameter has garnered significant attention in articles [7, 8, 14, 17, 18].

Bača et al. [13] explored two modifications of the irregularity strength of graphs, specifically a total edge irregularity strength, denoted by $tes(G)$, and a total vertex irregularity strength, denoted by $tvs(G)$. Some results regarding total edge irregularity strength and total vertex irregularity strength can be found in the articles [3, 5, 4, 20, 21, 24, 26].

In 2014, Ahmad et al.[6] introduced an edge irregular k-labeling and established a lower bound for this type of labeling. A vertex k-labeling $V(G) \to \{1, 2, 3, \ldots, k\}$ ensures that for any two distinct edges e and f, their respective weights $w_\phi(e)$ and $w_\phi(f)$ are different. Here, the weight of an edge $e = xy \in E(G)$ is calculated as $w_\phi(xy) = \phi(x) + \phi(y)$. In this case, the weight of an edge is defined as the sum of the labels of its ends. The minimum k for which a graph G has an irregular k-labeling is called the edge irregularity strength of G, denoted by $es(G)$. In recent years, a lot of work has been done on $es(G)$ for different families of graphs and trees using mathematically and algorithmically [2, 10, 11, 12, 23, 27, 28, 30, 29].

Theorem 1.1 ([6]). *Let $G = (V, E)$ be a simple graph with maximum degree $\Delta = \Delta(G)$. Then*

$$es(G) \geq \max\left\{\left\lceil \frac{|E(G)|+1}{2} \right\rceil, \Delta(G)\right\}.$$

Edge irregular k-labeling is a significant area of study in graph theory due to its applications in network design, resource distribution, and information encoding, where computing unique weights to edges can help prevent ambiguity and conflicts. Among various graph types, complete graphs K_n present a particularly challenging case because of the highest number of edges $\left(\frac{n(n-1)}{2}\right)$, leading to a combinatorially complex labeling space.

Asim et al. [9] as first attempt have computed the edge irregular k-labeling using an algorithmic approach satisfying Theorem 1. Their contribution established an upper bound for the edge irregularity strength of complete graphs and laid a foundation for further algorithmic investigation. However, as the graph size increases, the difficulty of ensuring unique edge weights through vertex labeling grows exponentially, and several edge deficiencies remain in the computed results.

While they have explored the edge irregular k-labeling of complete graphs K_n, but their findings have uncovered several edge deficiencies that needs attention and improvement through various design strategies. By addressing these deficiencies,it is possible to reducing the minimum of the maximum lable k, which directly translates to optimizing the use of labels and improving algorithmic efficiency. These shortcomings could be mitigated by employing optimization techniques, algorithmic modifications, or restructuring the computation process. However, it is important to note that the algorithms presented in [9] computes the edge irregular k-labeling using decrease and conquer strategy bearing the time complexity as $O(|V|) \leq |V|^2$.

In this paper, our primary objective is to improve the mathematical results on edge irregular k-labeling and remove edge deficiencies in complete graphs K_n by leveraging a branch and bound algorithm design strategy [9]. The algorithm is intentionally designed to eliminate edge deficiencies, prioritizing labeling accuracy over time complexity. In other words, the focus is placed on finding edge irregular k-labelings with smaller maximum labels that is, improved values or bounds for $es(K_n)$, even if this requires additional computational resources [1]. Unlike sequential execution, which processes one labeling branch at a time, the parallel version distributes multiple labeling paths across CPU cores to explore the search space concurrently [16]. This design choice enables the algorithm to remain computationally feasible for larger values of $|V|$, while still addressing the limitations of prior single-core approaches. Although this parallel strategy increases implementation complexity, it significantly improves runtime performance without compromising the algorithm's primary goal—reducing the maximum label used in an edge irregular k-labeling.

2. Algorithmic optimization for edge irregular labeling of complete graph K_n

This section is used to work on a parallel algorithmic framework for computing edge-irregular k-labelings of complete graphs K_n. The purpose is to reduce the edge deficiencies by ensuring that every edge weight is unique and at the same time, the main goal is to minimize the maximum assigned label. A non recursive branch-and-bound approach is used to explore feasible labeling configurations and discard the invalid or suboptimal paths efficiently. To improve scalability, the algorithm implements parallel processing, enabling concurrent traversal of the exponential search space and significantly reducing computation time.

2.1. Algorithm structure and workflow

The algorithm follows a stack-based depth-first search strategy, distributed across multiple cores. It starts with a fixed seed of initial labels $[1, 2, 3, 5]$, and the total number of edges $E = \frac{|V|(|V|-1)}{2}$ is used to estimate an upper bound on label values. Each parallel worker explores a disjoint section of the search space, evaluating label combinations and verifying edge weight uniqueness via a hash set. Valid extensions are pushed back onto the stack for further exploration. Shared data structures are used to coordinate the best result across processes.

2.2. Key components and data structures

- *Combination (combo):* contains the current sequence of vertex labels used in the graph.
- *Edge Set (edge_ set):* is used to store the existing edge weights and it must keep all edge weight unique.
- *Stack (stack):* Used for non-recursive DFS to manage the state of the combination.
- *Shared Best (shared_ best):* A synchronized dictionary is used to track the global best labeling across all threads.
- *Lock (lock):* Ensures safe access to shared data during concurrent updates.
- *Worker Arguments (args):*is used to contains input configurations for each parallel worker.

2.3. Optimization techniques

To improve efficiency and scalability, several optimization strategies are integrated:

- *Pruned Depth-First Exploration:* DFS explores different branches to achieve the target combination of labeling but infeasible branches are discarded early through branch-and-bound pruning to limit exponential expansion.
- *Hash-Based Edge Validation:* Dynamic hash sets are used to check the uniqueness of new edge with near-constant-time.
- *Golden-Ratio Spacing Heuristic:* Label growth and gap within consecutive label is controlled using a geometric spacing rule derived from the golden ratio [25].
- *Early Termination Threshold:* Any branch where the next label exceeds twice the previous label is immediately terminated.
- *Redundancy-Aware Computation:* Intermediate edge-states are reused to avoid re-calculation of the labels.
- *Stack-Based Memory Optimization:* Iterative DFS replaces recursion, lowering overhead for large values of $|V|$.
- *Parallel Search Synchronization:* In parallel processing multiple workers share and update a global best bound to ensure consistent pruning across threads.

2.4. Parallel processing enhancement

The parallel processing algorithm is used to maintain computational feasibility for large and complex graphs. Independent labeling paths are distributed across multiple CPU cores, with each core maintaining its own edge state and exploring the search space from a distinct initial label. A shared global tracker records and updates the best solution found across all workers, ensuring correctness while efficiently coordinating progress. This type of design enables the method to remain tractable even for large and complex graphs, where sequential exploration would become prohibitively expensive [16, 22].

3. Main results and discussion

3.1. Improving edge irregular k-labeling for complete graph K_n

This section presents some examples as pilot testing to compare the significant improvements of this work with previously known results. A new algorithm is designed using a branch and bound strategy, and the results are verified by implementing the algorithms using the computer programming language. The success of the new strategy is based on addressing edge deficiencies.

The presented algorithm is a combination generator with specific constraints to avoid edge duplication. Once a combination is formed, it records the value of k and then systematically explores other combinations for further improvements, as shown in Figures 1 and 2.

Combination - 1	1	2	3	5	8	13	21	30	39	53	74
Combination - 2	1	2	3	5	8	13	21	30	39	54	69
Combination - 3	1	2	3	5	8	14	23	33	41	49	65
Combination - 4	1	2	3	5	8	14	25	35	45	53	61
Combination - 5	1	2	3	5	9	16	29	38	50	55	60
1		3	4	6	10	17	30	39	51	56	61
2			5	7	11	18	31	40	52	57	62
3				8	12	19	32	41	53	58	63
5					14	21	34	43	55	60	65
9						25	38	47	59	64	69
16							45	54	66	71	76
29								67	79	84	89
38									88	93	98
50										105	110
55											115
60											

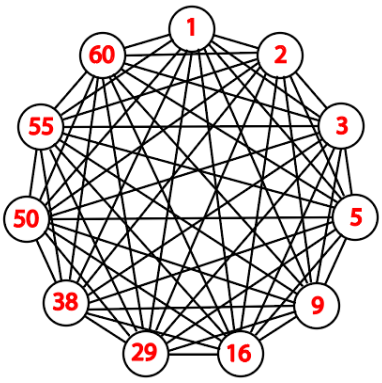

Fig. 1. Upper bound on the edge irregularity strength $es(K_n)$ for $|V|= 11$ is 60

Figure 1 displays the vertex label combinations generated for $|V|= 11$. Five distinct combinations were obtained:

$$\{1, 2, 3, 5, 8, 13, 21, 30, 39, 53, 74\},$$
$$\{1, 2, 3, 5, 8, 13, 21, 30, 39, 54, 69\},$$
$$\{1, 2, 3, 5, 8, 14, 23, 33, 41, 49, 65\},$$
$$\{1, 2, 3, 5, 8, 14, 25, 35, 45, 53, 61\},$$
$$\{1, 2, 3, 5, 9, 16, 29, 38, 50, 55, 60\}.$$

Each combination yields an edge irregular k-labeling of K_{11}, producing maximum label

values 74, 69, 65, 61, and 60, respectively. Since 60 is the smallest among them, we conclude that an improved upper bound on $es(K_{11})$ is 60, confirming an improvement over previously reported results. The weight of each edge is obtained by summing the labels corresponding to the row and column entries. In constructing the edge-weight matrix, the column index c is considered only when $c > r$ and $c \leq |V|$, ensuring that each pair is counted once and no duplicate edge weights arise.

By using this approach, an edge label deficiency can be reduced identified in previously published work. Through systematic exploration and adherence to strict constraints, our algorithm offers a robust solution for improving the edge irregular k-labeling of complete graphs K_n, providing valuable insights for advancing graph theory and computational methodologies.

Combination - 1	1	2	3	5	8	13	21	30	39	53	74	95	128	152	182	212	258	316	374	413
Combination - 2	1	2	3	5	8	13	21	30	39	53	74	95	128	152	182	212	258	317	379	394
Combination - 3	1	2	3	5	8	13	21	30	39	53	74	95	128	152	182	212	261	324	367	382
Combination - 4	1	2	3	5	8	13	21	30	39	53	74	95	134	175	218	242	275	330	345	360
Combination - 5	1	2	3	5	8	13	21	30	39	53	76	97	118	153	196	243	276	300	315	330
Combination - 6	1	2	3	5	8	13	21	30	39	65	108	122	155	187	211	226	241	281	302	323
Combination - 7	1	2	3	5	8	13	21	34	67	76	121	136	173	203	217	231	253	270	287	310
Combination - 8	1	2	3	5	8	13	23	46	74	83	102	132	148	161	190	216	252	266	283	300
Combination - 9	1	2	3	5	8	13	24	37	46	73	91	128	143	158	189	209	237	251	265	290
Combination - 10	1	2	3	5	8	14	23	44	70	80	99	128	144	152	160	198	233	247	261	284
1		3	4	6	9	15	24	45	71	81	100	129	145	153	161	199	234	248	262	285
2			5	7	10	16	25	46	72	82	101	130	146	154	162	200	235	249	263	286
3				8	11	17	26	47	73	83	102	131	147	155	163	201	236	250	264	287
5					13	19	28	49	75	85	104	133	149	157	165	203	238	252	266	289
8						22	31	52	78	88	107	136	152	160	168	206	241	255	269	292
14							37	58	84	94	113	142	158	166	174	212	247	261	275	298
23								67	93	103	122	151	167	175	183	221	256	270	284	307
44									114	124	143	172	188	196	204	242	277	291	305	328
70										150	169	198	214	222	230	268	303	317	331	354
80											179	208	224	232	240	278	313	327	341	364
99												227	243	251	259	297	332	346	360	383
128													272	280	288	326	361	375	389	412
144														296	304	342	377	391	405	428
152															312	350	385	399	413	436
160																358	393	407	421	444
198																	431	445	459	482
233																		480	494	517
247																			508	531
261																				545
284																				

Fig. 2. Upper bound on the edge irregularity strength $es(K_n)$ for $|V|= 20$ is 284.

In terms of *data structure and memory space* point of view, the proposed algorithm optimally utilizes *hash sets and combination generators* to ensure efficient storage and retrieval of vertex labels. By leveraging *hash sets*, the algorithm minimizes redundant calculations and prevents duplicate edge weights, leading to a reduction in *memory overhead*. Additionally, the *branch and bound approach* strategically prunes infeasible solutions, significantly reducing the search space and improving *computational efficiency*. These enhancements contribute to a *scalable and memory-efficient solution* for computing edge irregular k-labeling of complete graphs.

Figure 2 presents the computed edge irregular k-labelings for $|V|= 20$. In previously published work, the edge irregularity strength of K_{20} was given as $es(K_{20}) = 413$. Using our branch and bound approach, multiple valid k-labelings were generated and the maximum label in each case was recorded. Among these, the smallest value obtained was 284,

thereby establishing that an improved upper bound on $es(K_{20})$ is 284. This demonstrates a significant improvement over the earlier reported value, validating the effectiveness of the proposed computational strategy.

3.2. Algorithmic Approach

Algorithm 1: Complete-Graph-Labeling(n)

```
Number of vertices n = |V(K_n)| Vertex labeling minimizing the maximum label;
E = ⌊n(n − 1)/2⌋;
Last = E · ⌊log_2 n⌋;
fixed = [1, 2, 3, 5];
best_label = Last,   best_combo = [];
start_values = values from max(fixed) + 1 to Last − 1;
Create a pool with max_workers processors;
for i = 0 to (start_values) CHUNK_SIZE do
    chunk = sublist of start_values from i to i + CHUNK_SIZE;
    foreach val in chunk do
        Submit (val, n, Last, fixed) to the pool;
    foreach completed result do
        if result is valid then
            (label, combo) = result;
            if label < best_label then
                best_label = label;
                best_combo = combo;
return best_combo;
```

Algorithm 1 takes a positive integer n as input and aims to compute the smallest known upper bound on the edge irregularity strength $es(K_n)$ of a complete graph by generating vertex k-labelings such that all edge weights—defined as the sum of labels on adjacent vertices—are pairwise distinct. It begins by computing the number of edges E and estimating an upper bound *Last* for label values using $E \cdot \lfloor \log_2 n \rfloor$. A predefined set of seed labels $[1, 2, 3, 5]$ is used to initialize the labeling process; these are included in every candidate combination to reduce the initial search depth by forming a partially valid configuration. The remaining label values are drawn from the range max(fixed) + 1 to Last − 1, providing the basis for further exploration. To address the exponential complexity of the search space, the algorithm leverages parallel processing via a process pool executor. The candidate label range is divided into chunks, each assigned to a separate CPU core. Within each core, a worker() function employs an iterative, stack-based depth-first search to construct valid labelings while ensuring all edge weights remain unique. As separate processes return results, their outputs are compared and then globally minimal maximum label—representing the best known upper bound on the edge irregularity strength —is updated. This parallelized design improves computational efficiency and scalability, allowing

the framework to process larger complete graphs that would otherwise be impractical to handle sequentially.

Algorithm 2 explores valid vertex labelings using an iterative, stack-based depth-first search. Starting from a partial labeling, it extends the combination by appending new labels while ensuring that all resulting edge weights remain distinct. Labelings that exceed structural constraints or duplicate edge weights are pruned early. Upon reaching a complete labeling of size $|V|$, the algorithm updates the global best solution if a smaller maximum label is achieved, ensuring thread-safe access to shared data structures.

Algorithm 2: DFS-Worker($|V|$, combo, edge_set, start, shared_best, lock)

```
1  stack, valid = [(combo, edge_set, start)], false;
2  while stack is not empty do
3      (c, e, val) = pop(stack);
4      if length(c) = |V| then
5          label = max(c);
6          Acquire(lock);
7          if label < shared_best.label then
8              shared_best.label = label;
9              shared_best.combo = c;
10         Release(lock);
11     else
12         for i = val to shared_best.label − 1 do
13             if length(c) > 4 and i < 2 × max(c) then
14                 valid, new_edges = IsUnique(e, c, i);
15             if valid then
16                 new_set = e ∪ new_edges;
17                 push(stack, (c + [i], new_set, i + 1));
```

Algorithm 3 constructs valid vertex labelings by iteratively extending a partial vertex labeling $f : V(K_n) \to \mathbb{Z}^+$, starting from a given index. At each step, a candidate label is appended, and the uniqueness of all induced edge weights $w(uv) = f(u) + f(v)$ is verified. If the labeling remains feasible, the process continues recursively. Upon reaching a complete labeling of size $|V|$, the algorithm returns the maximum assigned label. If this value is smaller than the current upper bound of the edge irregularity strength $es(K_n)$, it updates the bound accordingly. The search explores all label extensions within a bounded domain while pruning infeasible paths.

Algorithm 4 verifies the uniqueness of newly formed edge weights resulting from extending a partial vertex labeling $f : V(K_n) \to \mathbb{Z}^+$. It constructs the set of all previously generated edge weights and checks whether any new edge weight of the form $w(uv) = f(u) + f(v)$, where u is the newly added vertex and v is an existing vertex, already exists. If duplication is detected, the algorithm terminates early. Otherwise, it returns the list of newly generated unique edge weights. The verification process operates

in $O(|V|)$ time, improving efficiency during incremental label extension.

Algorithm 3: CombGenerator(start, $|V|$, hash_set, fixed_numbers, combo, Label, Last, result)

Input: Start index, number of vertices $|V|$, current labeling, hash set, best label
Output: Best updated label and labeling

```
if length (combo) = |V| then
    return (max (combo), combo);
for i = start to Last do
    new_combo = combo extended with i;
    (flag, t) = IsUnique(hash_set, new_combo);
    if flag is true then
        hash_set[length(combo)] = t;
        (temp_label, temp_combo) = CombGenerator(i +
          1, |V|, hash_set, fixed_numbers, new_combo, Label, Last, result);
        if temp_label < Label then
            Label = temp_label;
            result = temp_combo;
return (Label, result);
```

Algorithm 4: IsUnique(hash_set, combo)

Input: Current hash set of edge weights, partial label combination
Output: Boolean flag indicating uniqueness, list of new edge weights

```
edge_set = flatten all lists in hash_set into a single set;
for r = 0 to length (combo) − 2 do
    if last (combo) + combo[r] exists in edge_set then
        return (false, empty list);
new_edges = empty list;
for r = 0 to length (combo) − 2 do
    append (last (combo) + combo[r]) to new_edges;
return (true, new_edges);
```

3.3. Computer-based results

A computer program has been developed to test the algorithm under discussion and compute the results for various input values of vertices $|V|$ of the complete graph K_n.

The comparison of branch and bound and iterative approach while labeling the vertices is presented in Table 1.

The proposed algorithm was tested up to $|V|= 50$ due to the exponential increase in computational complexity and memory requirements as the graph size grows. For instance, even a relatively small instance such as $|V|= 5$, when selecting labels from the set $\{1, 2, \ldots, 8\}$, requires evaluating $P(8, 5) = 6,720$ unique permutations. Each

permutation involves checking $\frac{|V|(|V|-1)}{2}$ edge weights to ensure all are unique—a process that becomes increasingly exhaustive as $|V|$ increases.

To address this challenge, our branch and bound (B&B) strategy initiates the labeling process with a fixed seed combination $\{1, 2, 3, 5\}$. The next (fifth) element is generated starting from 6 onwards, and subsequent elements are incrementally determined based on the previous value. Importantly, the gap between consecutive elements is restricted to no more than 1.6 times the value of the preceding element. This constraint aligns with the golden ratio, serving as a termination condition to optimize the search space and improve efficiency.

During execution, the algorithm records the last successful labels of the current combination. It then backtracks to explore other possible combinations starting from that point. If a newly generated combination yields a lower final value (the corresponding value of k) than any previously recorded, it is retained as the current best solution. This recursive exploration continues until all valid combinations have been tested, and the labeling configuration with the smallest maximum label is identified for the edge irregularity strength of the given complete graph.

Table 1. Comparison of branch and bound and iterative approach while labeling the vertices

Input List		Calculated Values		
—V—	E	Branch & Bound	Iterative	$\lfloor E \log_2 \lvert V \rvert \rfloor$
5	10	8	8	23
6	15	13	13	38
7	21	19	21	58
8	28	25	30	84
9	36	35	39	114
10	45	47	53	149
11	55	60	74	190
12	66	74	95	236
13	78	91	128	288
14	91	108	152	346
15	105	132	182	410
20	190	284	413	821
25	300	595	717	1393
30	435	977	1161	2134
35	595	1501	1722	3051
40	780	2154	2497	4151
45	990	2961	3332	5436
50	1225	3914	4447	6913

In Figure 3, the chart shows the differences between the branch and bound approach and the iterative approach, as well as the calculated results in relation to $es(K_n) \leq E \log_2 |V|$, we can observe the following:

1. *Branch and Bound Approach vs. Iterative Approach in K_n:* The branch and bound approach tends to yield lower values compared to the iterative approach for $es(K_n)$ across various values of $|V|$ and E. This suggests that the branch and bound approach is more effective at finding labelings whose maximum label (and thus the computed value of $es(K_n)$) is smaller than that obtained by the iterative approach. The iterative approach produces higher values for $es(K_n)$, indicating that it may not be as effective in reducing edge irregularity or optimizing graph labeling compared to branch and bound.

2. *Comparison with Theoretical Bound $es(K_n) \leq E \log_2|V|$:* - The calculated $es(K_n)$ values from both approaches are compared against the theoretical upper bound given by $E \log_2|V|$. - In most cases, the calculated $es(K_n)$ values are within reasonable proximity to the theoretical bound, indicating that the graph labeling algorithms are performing relatively well.

Overall, the comparison highlights the effectiveness of the branch and bound approach in producing smaller computed values of $es(K_n)$ compared to the iterative approach. Additionally, the analysis against the theoretical bound provides insights into the performance of the graph labeling algorithms in relation to established theoretical limits.

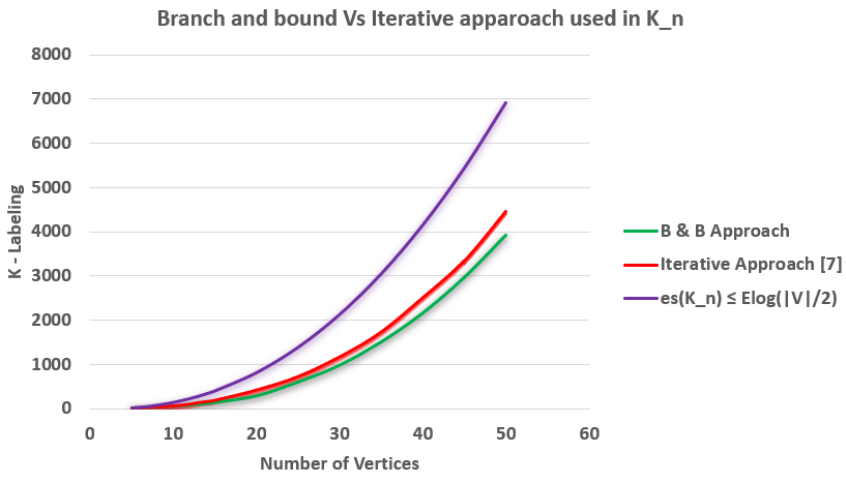

Fig. 3. The comparison between branch and bound and iterative approach

3.4. Time complexity

The time complexity of an algorithm shows the effectiveness and scalability of the proposed algorithm. The primary goal of this section is to quantify the computational cost associated with each function used in the edge irregular labeling process. Since the overall algorithm employs recursive, combinatorial search strategies with parallel processing enhancements, it is very important to compute the time complexity of each function step by step. We analyze each function individually IsUnique, CombGenerator, Worker, and Complete-Graph-Labeling to determine their theoretical worst-case complexity in terms of the number of vertices $|V|$ and edges $E = \frac{|V|(|V|-1)}{2}$. This breakdown enables us to estimate the total computational load and justify the use of parallelization for handling larger graph sizes.

3.4.1. Time complexity of IsUnique. The function IsUnique verifies whether adding a new vertex label preserves the uniqueness of all induced edge weights. In the proposed implementation, the edge weights are conceptually treated as a flattened hash set for efficient membership testing. However, in the provided pseudocode implementation, the edge weights are dynamically flattened from hash_set into a temporary set during each function call.

When a new label is considered, the algorithm checks the sums between the new label and each existing vertex label.

Since the combination contains at most $|V|$ labels, and each membership test in the hash set is performed in constant time, the overall time complexity of IsUnique is

$$O(|V|).$$

This optimized design avoids iterating over all edges and ensures that the complexity of IsUnique does not depend on the number of edges E.

3.4.2. Time complexity of CombGenerator. The CombGenerator function incrementally extends a partial labeling by assigning new vertex labels while maintaining the uniqueness of edge weights. Initially, four labels are fixed, reducing the number of labels to be assigned to $|V|-4$, where $|V|$ is the number of vertices. The available label pool is of size $O(|V|^2\log|V|)$, based on the estimated upper bound for the maximum label value.

At each recursion level, the function loops over up to $O(|V|^2\log|V|)$ candidate labels. Each candidate requires a validation step using the optimized IsUnique function, which operates in $O(|V|)$ time. Consequently, the cost per level is:

$$O(|V|^3\log|V|).$$

Since the recursion depth can reach up to $|V|-4$ levels, the total number of explored combinations follows an exponential growth with respect to $|V|$. Thus, the worst-case total time complexity of CombGenerator is:

$$O\left((|V|^2\log|V|)^{|V|} \times |V|\right).$$

3.4.3. Time complexity of DFS-worker. The DFS-Worker algorithm implements a stack-based depth-first search to generate edge irregular k-labelings. It explores different combinations of vertex labels iteratively by:

- Maintaining a stack of partial labelings.
- Extending partial labelings by attempting to add new labels from the candidate range.
- Validating the uniqueness of edge weights for each extension using the IsUnique function.

At each node (i.e., each partial labeling), the following operations occur:

- Checking if the current combination is complete ($O(1)$).

- Finding the maximum label ($O(1)$), updating the shared best ($O(1)$).
- Iterating over candidate labels (up to $O(|V|^2\log|V|)$ candidates).
- For each extension, calling IsUnique which runs in $O(|V|)$ time.

The total number of nodes (i.e., partial labelings) that may be generated is asymptotically bounded by the number of ways to assign $|V|-4$ labels:

$$O\left((|V|^2\log|V|)^{|V|}\right).$$

Thus, multiplying the cost per node by the number of nodes, the overall time complexity of DFS-Worker is:

$$O\left((|V|^2\log|V|)^{|V|} \times |V|\right).$$

3.4.4. Time complexity of complete-graph-labeling. The Complete-Graph-Labeling algorithm serves as the main driver function, orchestrating the parallel search for an optimal edge irregular k-labeling for complete graphs K_n. Its key steps involve:

- Calculating the number of edges $E = \frac{|V|(|V|-1)}{2}$ and the upper bound for labels $Last = E \times \lfloor \log_2|V| \rfloor$.
- Generating a range of candidate label values, of size approximately $O(|V|^2\log|V|)$.
- Dividing the candidate label range into chunks and distributing them across multiple processors.
- For each candidate, a parallel worker process attempts to construct a valid labeling.

Each call to a Worker executes a DFS-Worker routine, whose time complexity has been established as:

$$O\left((|V|^2\log|V|)^{|V|} \times |V|\right).$$

The number of parallel tasks submitted is proportional to the number of candidate starting values, that is $O(|V|^2\log|V|)$. However, since the search is distributed across P processors, the total effective time complexity is reduced by a factor of P.

Thus, the overall time complexity of Complete-Graph-Labeling becomes:

$$O\left(\frac{(|V|^2\log|V|)^{|V|} \times |V|}{P}\right),$$

where P is the number of available parallel processing units.

3.4.5. Overall time complexity. The proposed framework for computing edge irregular k-labeling of complete graphs integrates four core algorithms: *IsUnique*, *CombGenerator*, *DFS-Worker*, and *Complete-Graph-Labeling*.

- The IsUnique function operates in $O(|V|)$ time due to the flattened hash set.
- The CombGenerator function recursively explores valid labelings over a candidate label range of order $O(|V|^2\log|V|)$. In the worst case, it evaluates many such candidates at each recursion level and invokes IsUnique for each, leading to an exponential-time behaviour dominated by a factor of $(|V|^2\log|V|)^{|V|}$, multiplied by a polynomial overhead in $|V|$ and E.

- The DFS-Worker algorithm mirrors the search space of CombGenerator, but uses an explicit stack-based depth-first strategy while repeatedly calling IsUnique to prune infeasible labelings.
- The Complete-Graph-Labeling function coordinates the parallel execution by distributing disjoint portions of the search space across P processing units and aggregating the best solution found.

Considering the exponential nature of the solution space and the use of parallelism, a worst-case upper bound on the overall time complexity of the complete framework can be expressed as

$$O\left(\frac{(|V|^2\log|V|)^{|V|}\cdot(|V|)}{P}\right), \tag{1}$$

where

- $|V|$ is the number of vertices,
- P denotes the number of parallel processing units.

4. Conclusion and future work

This research presents a comprehensive investigation into the edge irregular k-labeling of complete graphs K_n, focusing on addressing previously observed edge deficiencies and enhancing the mathematical precision of labeling strategies. Through the development of a novel branch-and-bound algorithmic framework, we have achieved a notable improvement in the upper bound of edge irregularity strength. Specifically, the proposed method suggests a better upper bound than was previously proposed from $E\log|V|$ to a more efficient expression $E\log\left(\frac{|V|}{2}\right)$, representing a significant advancement in the labeling methodology for complete graphs.

The mathematical refinement achieved in this work can be summarized by the following relation:

$$E\cdot\log_2\left(\frac{|V|}{2}\right) < E\cdot\log_2|V| < F_n. \tag{2}$$

The proposed approach consistently produces tighter upper bounds for the edge irregularity strength $es(K_n)$ as the graph size $|V|$ increases. This result shows the effectiveness of the algorithm to reduce duplicate edge weights and eliminate edge deficiencies, outperforming conventional iterative methods. The close alignment between empirical outcomes and theoretical predictions shows the mathematical soundness and robustness of the proposed solution.

A good contribution of this work is the integration of parallel processing to mitigate the exponential computational complexity inherent in recursive labeling algorithms. By distributing the search space across P processing units, the effective runtime is reduced from $O(|V|^2\log|V|)^{|V|}\times|V|$ to:

$$O\left(\frac{(|V|^2\log|V|)^{|V|}\times|V|}{P}\right),$$

thereby ensuring scalability without compromising the correctness or precision of the labeling process. A future research work include extending this algorithmic framework to different classes of graphs which includes the circulant graphs, bipartite graphs, and strong product graphs, to evaluate its adaptability and generalization capabilities. Further enhancements may incorporate dynamic pruning strategies, heuristic-driven labeling methods, and GPU-accelerated parallelism to efficiently address increasingly complex graph instances.

Overall, this research contributes a mathematically rigorous and computationally scalable solution to the edge irregular k-labeling problem. It provides both theoretical insights and practical methodologies, establishing a foundation for future advances in graph labeling, combinatorial optimization, and high-performance algorithm design.

References

[1] E. Aarts and J. Lenstra. *Local Search in Combinatorial Optimization*. Princeton University Press, 2003.

[2] A. Ahmad, M. Asim, M. Bača, and R. Hasni. Computing edge irregularity strength of complete m-ary trees using algorithmic approach. *U.P.B. Scientific Bulletin, Series A: Applied Mathematics and Physics*, 80(3):145–152, 2018.

[3] A. Ahmad, M. Bača, Y. Bashir, and M. Siddiqui. Total edge irregularity strength of strong product of two paths. *Ars Combinatoria*, 106:449–459, 2012.

[4] A. Ahmad, M. Bača, and M. Siddiqui. On edge irregular total labeling of categorical product of two cycles. *Theory of Computing Systems*, 54:1–12, 2014. https://doi.org/10.1007/s00224-013-9470-3.

[5] A. Ahmad, E. Baskoro, and M. Imran. Total vertex irregularity strength of disjoint union of helm graphs. *Discussiones Mathematicae Graph Theory*, 32(3):427–434, 2012. https://doi.org/10.7151/dmgt.1619.

[6] A. Ahmad, O. Al-Mushayt, and M. Baca. On the edge irregularity strength of graphs. *Applied Mathematics and Computation*, 243:607–610, 2014. https://doi.org/10.1016/j.amc.2014.06.028.

[7] M. Aigner and E. Triesch. Irregular assignments of trees and forests. *SIAM Journal on Discrete Mathematics*, 3:439–449, 1990. https://doi.org/10.1137/0403038.

[8] D. Amar and O. Togni. Irregularity strength of trees. *Discrete Mathematics*, 190:15–38, 1998. https://doi.org/10.1016/S0012-365X(98)00112-5.

[9] M. Asim, A. Ahmad, and R. Hasni. Iterative algorithm for computing irregularity strength of complete graph. *Ars Combinatoria*, 138:17–24, 2018.

[10] M. Asim, A. Ahmad, and R. Hasni. Edge irregular k-labeling for several classes of trees. *Utilitas Mathematica*, 111:75–83, 2019.

[11] M. Asim, R. Hasni, and A. Ahmad. Edge irregular k-labeling for several classes of trees. *Utilitas Mathematica*, 111:75–83, 2019.

[12] M. Asim, R. Hasni, A. Ahmad, B. Assiri, and A. Fenovcikova. Irregularity strength of circulant graphs using algorithmic approach. *IEEE Access*, 9:54401–54406, 2021. https://doi.org/10.1109/ACCESS.2021.3058786.

[13] M. Bača, S. Jendroľ, M. Miller, and J. Ryan. On irregular total labellings. *Discrete Mathematics*, 307:1378–1388, 2007. https://doi.org/10.1016/j.disc.2005.11.075.

[14] T. Bohman and D. Kravitz. On the irregularity strength of trees. *Journal of Graph Theory*, 45:241–254, 2004. https://doi.org/10.1002/jgt.10158.

[15] G. Chartrand, M. Jacobson, J. Lehel, O. Oellermann, S. Ruiz, and F. Saba. Irregular networks. *Congressus Numerantium*, 64:187–192, 1988.

[16] S. Cook and C. Dwork. Bounds on the time for parallel rams to compute simple functions. *Journal of Computer and System Sciences*, 25(3):225–237, 1982.

[17] R. Faudree, M. Jacobson, J. Lehel, and R. Schelp. Irregular networks, regular graphs and integer matrices with distinct row and column sums. *Discrete Mathematics*, 76:223–240, 1989. https://doi.org/10.1016/0012-365X(89)90321-X.

[18] A. Frieze, R. Gould, M. Karonski, and F. Pfender. On graph irregularity strength. *Journal of Graph Theory*, 41:120–137, 2002. https://doi.org/10.1002/jgt.10056.

[19] J. Gallian. A dynamic survey of graph labeling. *Electronic Journal of Combinatorics*, 25:DS6, 2022. https://doi.org/10.37236/11668.

[20] J. Ivančo and S. Jendroľ. Total edge irregularity strength of trees. *Discussiones Mathematicae Graph Theory*, 26:449–456, 2006. https://doi.org/10.7151/dmgt.1337.

[21] S. Jendroľ, J. Miškuf, and R. Sot'ak. Total edge irregularity strength of complete graphs and complete bipartite graphs. *Discrete Mathematics*, 310:400–407, 2010. https://doi.org/10.1016/j.disc.2009.03.006.

[22] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin/Cummings Publishing Company, 1994.

[23] M. Nadeem, M. Cancan, M. Imran, and Y. Ali. On edge irregularity strength of certain families of snake graph. *Journal of Prime Research in Mathematics*, 19(1):92–101, 2023.

[24] Nurdin, E. Baskoro, A. Salman, and N. Gaos. On the total vertex irregularity strength of trees. *Discrete Mathematics*, 310:3043–3048, 2010. https://doi.org/10.1016/j.disc.2010.06.041.

[25] O. Omotehinwa and S. Ramon. Fibonacci numbers and golden ratio in mathematics and science. *International Journal of Computer and Information Technology*, 2(4):630, 2013. Available: www.ijcit.com. ISSN: 2279–0764.

[26] J. Przybylo. Linear bound on the irregularity strength and the total vertex irregularity strength of graphs. *SIAM Journal on Discrete Mathematics*, 23:511–516, 2009. https://doi.org/10.1137/070707385.

[27] M. Shahzad, M. Asim, R. Hasni, and A. Ahmad. Computing edge irregularity strength of star and banana trees using algorithmic approach. *Ars Combinatoria*, 159(1):11–20, 2024. https://doi.org/10.61091/ars159-02.

[28] M. Shahzad, R. Hasni, I. Tarawneh, and M. Asim. Computing the edge irregularity strength of some classes of grid graphs. *Malaysian Journal of Mathematical Sciences*, 18(3):617–630, 2024. https://doi.org/10.47836/mjms.18.3.10.

[29] I. Tarawneh, R. Hasni, A. Ahmad, and M. Asim. On the edge irregularity strength for some classes of plane graphs. *AIMS Mathematics*, 6(3):2724–2731, 2021. https://doi.org/10.3934/math.2021166.

[30] I. Tarawneh, R. Hasni, M. Siddiqui, and M. Asim. On the edge irregularity strength of disjoint union of graphs. *Ars Combinatoria*, 143:239–249, 2019.

Muhammad Shahzad
Department of Computing Sciences, Gulf College, Muscat, 133, Oman
E-mail shahzad@gulfcollege.edu.om

Muhammad Ahsan Asim
Division of Computing, Analytics and Mathematics, School of Science and Engineering
University of Missouri-Kansas City, MO 64110, USA
E-mail mr.ahsan.asim@gmail.com

Roslan Hasni
Special Interest Group on Modeling and Data Analytics (SIGMDA)
Faculty of Computer Science and Mathematics
Universiti Malaysia Terengganu, 21030 Kuala Nerus, Terengganu, Malaysia
E-mail hroslan@umt.edu.my

Ali Ahmad
Department of Computer Science, College of Engineering and Computer Science
Jazan University, Jazan, Saudi Arabia
E-mail hmadsms@gmail.com