Split-by-edges trees

Asbjørn Brændeland

Abstract
A split-by-edges tree of a graph \( G = (V, E) \) is a set of vertices no two of which are adjacent, and if \( G \) has no larger independent set then \( I \) is a maximum independent set of \( G \). Tarjan and Trojanowski point out that, given a vertex \( v \in V \), any maximum independent set of \( G \) must be a subset of either \( V - N(v) \) or \( V - v \), using that as the starting point for an algorithm that finds a maximum independent set in less than \( 2^{|V|} \) time [1]. Relatedly, given an edge \( uv \in E \) and a maximum independent set \( M \subseteq V \), either \( M \subseteq V \setminus \{ u \} \) or \( M \subseteq V \setminus \{ v \} \). This gives rise to the following definition.

Definition 1: Let \( G \) be graph and let \( T \) be a binary tree of subsets of \( V(G) \). Then \( T \) is a split-by-edges tree, or SBE-tree, of \( G \) if and only if the root of \( T = V(G) \), every leaf in \( T \) is an independent set of \( G \), and for every other node \( N \) in \( T \) with children \( L \) and \( R \) there is a pair of vertices \{\( u, v \}\} \subseteq N \) such that \( L = N \setminus \{ v \} \) and \( R = N \setminus \{ u \} \), and \( u \) and \( v \) are adjacent in \( G \).

![Figure 1](image)

Figure 1. An SBE-tree of the graph at the upper left. The leaves have bold blue frames. The gray nodes are duplicates of others. The branching labels, which do not belong to the tree, show the splitting edges.

Theorem 1. Given a graph \( G \) and a split-by-edges tree \( T \) of \( G \), for every independent set \( X \) of \( G \) there is a leaf \( Y \) in \( T \) such that \( X \subseteq Y \).

Proof: Given an independent set \( I \) of \( G \), for every node \( N \) in \( T \), if \( I \subseteq N \) then \( I \subseteq L(N) \) or \( I \subseteq L(N) \). □

Corollary 1.1. Every maximal independent set of \( G \) is a leaf in \( T \). □
The number of possible child pairs of a node in an SBE-tree equals the number of neighbor pairs in the node. If \( G \) is the \( n \)-complete graph a \( k \)-vertex node in the SBE-tree of \( G \) contains \( \binom{k}{2} \) neighbor pairs, the number of possible quadruples of grandchildren is \( \binom{k}{2}(k-1)^2 \), the number of octuplets of grand-grandchildren is \( \binom{k}{2}(k-2)^2 \), etc. E.g. \( K_6 \) has \( 15(10(6^2))^2 = 12,754,584,000 \) SBE-trees. However, by Theorem 1, for each \( l \), the contents of layer \( l \) is the same in all of these trees. If \( G \) is not complete, the contents, shapes and sizes of its SBE-trees vary, dependent on the order in which the edges are searched, as illustrated in Figure 2. But notice that for every layer \( L_l \) down to the one that contains the maximum independent sets, \( |L_l| = 2^l \).

With a given edge search order, an SBE-tree \( T \) of a graph \( G \) can be generated from any single node \( N \) in \( T \). I.e., if \( N \) is not the root of \( T \) there must be an edge uv found in a reverse edge search, with \( u \in N \) and \( v \not\in N \), and then \( N \cup \{v\} \) is the parent and \( (N \cup \{v\}) \setminus \{u\} \) is the sibling of \( N \) in \( T \).

**The uniquified SBE-tree**

For most graphs, every SBE-tree contains duplicate vertex sets, and the tree size can be many times the cardinality of the corresponding set.

**Definition 2:** An SBE-tree minus its duplicate nodes is a **uniquified SBE-tree**, or a USBE-tree.

Let \( T \) be an SBE-tree and \( T' \) the corresponding USBE-tree of \( G \). If \( G \) is a complete graph on \( n \) vertices the size of \( T \) is \( 2^n - 1 \) and the size of \( T' \) is \( \binom{n+1}{2} \). If \( G \) is not complete, the exclusion of duplicates has less effect, but this is to some extent outweighed by the occurrence of leaf nodes closer to the root, i.e., for each non-singleton leaf \( \phi \) in \( T \), the size of \( T \) is reduced by \( 2^{\phi-1} - 1 \) compared to the SBE-tree of a complete graph of the same order as \( G \). Below are SBE and USBE-tree sizes, etc., for some graphs.

| \( G \)               | \( n \) | \( m \) | \( \delta \) | \( \Delta \) | \( \alpha(G) \) | \( |\text{SBE}(G)| \) | \( |\text{USBE}(G)| \) |
|-----------------------|--------|--------|-------------|-------------|----------------|-------------------|-------------------|
| \( K_6 \)             | 48     | 1128   | 47          | 47          | 1              | 28147976710655    | 1176              |
| 18-regular graph      | 48     | 432    | 18          | 18          | 6              | 205624938644223   | 192146            |
| Apollonian network    | 48     | 138    | 3           | 6           | 12             | 54263808384247    | 17721342          |
| Möbius ladder         | 48     | 72     | 3           | 3           | 23             | 238972941719      | 153349985         |
| Path                  | 48     | 47     | 1           | 2           | 24             | 15557484097       | 15557484097       |
The SBE-tree of an edgeless graph has trivially (and by definition) just a single node. For connected graphs, the extreme cases, the complete graph, and the path, are determined as such by their density (disregarding heavily leafed graphs, such as stars, which rapidly break down to edgeless graphs).

In the USBE-tree of a complete graph, the width of layer \( l \) is \( l \), and every independent set is in the bottom layer. In the USBE-tree of a path on \( 2n \) vertices, the widths of the first \( n + 1 \) layers are \( 2^{l} \), and the combined size of these layers is \( 2^{n} + n \). The tree width continues to grow, at a slowing rate, until it reaches a maximum somewhere before three quarters from the root.

**Figure 3.** USBE-tree layer widths for 5 graphs on 48 vertices. The horizontal scale is \( 2 \log_{2} w \) (= layer width).

**Claim 2.** The SBE-tree of a path contains no duplicates.

**Proof:** Let \( P = (\{p_{1}, \ldots, p_{n}\}, \{p_{1}p_{2}, \ldots, p_{n-1}p_{n}\}) \), let \( T \) be the SBE-tree of \( P \) defined by the edge search order \((p_{1}p_{2}, p_{3}p_{4}, \ldots)\), let \( N \) be a node in \( T \) with a neighbor pair \((u, v)\), \( u < v \), that splits \( N \) into \( L \) and \( R \), and let \( M \) be the vertices \( p_{1} \) through \( u \) in \((p_{1}, \ldots, u, \ldots)\). Then \( M \) is an independent set in \( P \) and must occur in every node below \( L \) and, since \( u \) is not in \( R \), \( M \) cannot occur in any node below \( R \), thus the SBE-tree of \( R \) cannot contain a duplicate of any node in the SBE-tree of \( L \). □

The cardinality of a maximum independent set of an \( n \)-vertex path \( P = \lceil (n + 1)/2 \rceil \) and in the SBE-tree of \( P \), the MI sets are in layer number \( \lceil n/2 \rceil + 1 \), and the width of every layer \( k \) up to this is \( 2^{(k-1)} \).

The fact that every split in the SBE-tree of a path gives a pair of unique nodes, makes it possible to compute the layer widths of the tree in linear time, without constructing any part of the tree.
**USBE-trees of random graphs**

The graphs giving the numbers and curves in Figure 3 are structurally clear and well suited to illustrate the relation between graph density and USBE tree layer widths in general.

![Graphs](image)

Figure 4. 12-vertex versions of four of the 48-vertex graphs used as examples above.

Being without an inherent structure, a *random graph* is more malleable than the above graphs (when in a random graph \( (V, E) \) of a given order and size, the members of \( E \) have been selected at random from the handshake product of \( V \)), and a simple way to rearrange such a graph is to order its vertices by degree. Given the graph represented by the table

|   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| 2 | 1 | 4 | 5 | 9 | 11|
| 3 | 1 | 7 | 8 | 11|
| 4 | 1 | 2 | 6 | 9 | 11|
| 5 | 1 | 2 | 7 | 8 | 11|
| 6 | 1 | 2 | 7 | 10 | 12|
| 7 | 1 | 5 | 6 | 8 | 9 | 10 | 12|
| 8 | 1 | 3 | 4 | 5 | 7 | 9 | 10 | 12|
| 9 | 2 | 3 | 4 | 8 | 11|
| 10 | 6 | 8 | 11 | 12|
| 11 | 2 | 3 | 4 | 5 | 9 | 10 | 12|
| 12 | 5 | 6 | 8 | 10 | 11|

two by-degree arrangements give us two corresponding vertex mapping functions,

![Mappings](image)

and altogether this gives the three isomorphic graphs shown in Figure 5.

![Isomorphic Graphs](image)

Figure 5. The shapes below the graphs represent the corresponding USBE-trees. As we see, an ordering by descending degree gives a slimmer, and an ordering by ascending degree gives a wider tree than no ordering at all.
In a search for a split, we examine the edges in ascending order \((ab < cd \iff a < c \land (a = c \lor b < d))\).

In Figure 5 we see that in the graph ordered by descending degree (in the middle) the densest part contains the vertices 1 – 6, which are the first to be removed in a succession of split-by-edge operations, whereas in the graph ordered by ascending degree (to the right) the densest part contains the vertices 7 – 12, which are the last to be removed.

Finding a maximum independent set in a USBE-tree

Since the maximum independent sets of any graph are the leaves closest to the roots of its USBE-trees, such a set can be found in a layer-by-layer search. We construct each layer \(L_i\) from the one above, \(L_{i+1}\) in a succession of split operations. In order to avoid duplicates, we use a search tree, and since the sizes of the nodes are specific for each layer, we can use one search tree, \(S_i\), per layer. (The fact that \(L_i\) and \(S_i\) contain the same nodes, seems to indicate that we could have made do with \(S_i\) alone, but this has turned out to slow down the search process considerably, possibly as a result of the edge search order having been disrupted.) In a layer-by-layer search, an ordering of the graphs’ vertices by descending degree, more than halves the workload, as illustrated in Figure 6.

The complexity of the USBE-tree

We can describe the complexity of a USBE-tree in terms of the number of split operations, \(\mathcal{X}(n, m)\), required to find a maximum independent set in a graph of order \(n\) and size \(m\). Figure 6 shows the average values of \(\mathcal{X}(24, m)\) in 1000 runs for each \(m\) from 24 to 267. The maximum of the bottom curve \(\mathcal{X}(24, 83) = 1192,\) which is a little above \(2^{0.4n}\). As far as the tests go, the average of the sizes that give the highest split numbers lies a little above \(3n\) and the relative maximum number of splits falls from \(2^{0.47n}\) to \(2^{0.395n}\) for \(n = 12\) to 50.

For comparison, the number of split operations for Möbius ladders, for which \(m = 3n/2\), falls from \(2^{0.482n}\) to \(2^{0.38n}\) in the same interval, and since these graphs are not random we get a definite measure, which is \(2^{0.347120956815n+1.66485616037} - 2\) for \(n\) divisible by 4, and \(2^{0.347120956815n+1.74055665759} - 2\) for \(n\) divisible by 2 but not by 4. The values for random graphs are regular enough to indicate that the maximum split number for these are on the same form, \(2^{0.369425n+0.56325}\), to be specific, that is, the maximum number of split operations required to find a maximum independent set in a random graph on \(n\) vertices seems to be \(2^{O(0.369425n)}\).
Finding and utilising the set of independent sets of a graph

Let $I$ be the set of independent sets of a graph $G$, let $T$ be a USBE-tree of $G$, and let $F$ be the foliage of $T$. Then $F \subseteq I$, but since, by Theorem 1, every set in $I$ has a superset in $F$, $I$ can be generated from $F$. In principle, $I$ is the union of the powersets of the sets in $F$, but rather than generating all these power-sets, we iterate over $F^0(i)$ and $f^0(i)$ as follows, when $F^0(i) = F(i) = F$.

Let

$$F^0(i) = \{X \mid \exists Y \in F(i-1). X \subseteq Y \land |Y| > 1 \land |X| = |Y| - 1 \land X \notin F(i-1)\}$$

and let

$$f^0(i) = f(i-1) \cup F^0(i).$$

There must then be a $j$ such that $F^0(j) \neq \emptyset$ and $F^0(j+1) = \emptyset$, and then $f^0(i) = I$.

$I$ can be used to find the chromatic number $\chi(G)$ and, knowing $\chi(G)$, to compute all colorings of $G$ (which was what inspired the split-by-edges approach in the first place). We start with the latter.

Knowing $k = \chi(G)$ and $\mu = \alpha(G)$, we compute the set $\mathcal{A}$ of sets of $k$ numbers in $(1, \mu)$ that add up to the order of $G$. We then organize $I$ into a set of lists $L = \{L_1, L_2, \ldots, L_\mu\}$ of independent sets of uniform cardinalities, and for each set in $\mathcal{A}$ we find the sets of $k$ non-intersecting sets from the corresponding lists in $L$, and the union of all of this is then the set of colorings of $G$.

To find $k = \chi(G)$, we use $L$ as above and iterate over $k$, computing the successive sets of add-up sets $\mathcal{A}_k$, until a coloring is found. For each set $A_i = (a_{i,1}, \ldots, a_{i,k})$ in $\mathcal{A}_k$ we search through the corresponding lists in $L$, building a set $C$ of non-intersecting independent sets along the way. At some point we have $C = \{C_1, \ldots, C_{\mu+1}\}$, with one element from each of the respective lists $L_{a_{i,1}}, \ldots, L_{a_{i,k}}$, $i > 1$, and we then search through $L_{a_{i,j}}$.

• A match, if any, is a set in $L_{a_{i,j}}$ that does not intersect any of the sets in $C$.
  - If we find a match and $i = k$, we have a $k$-coloring of $G$, and $\chi(G) = k$.
  - If we find a match and $i < k$ we add the match to $C$ and proceed to examine $L_{a_{i,j+1}}$.
  - If we did not find a match we keep looking through $L_{a_{i,j+1}}$ with $C = \{C_1, \ldots, C_{\mu+2}\}$, or $C = \emptyset$, if $i = 2$.

• If we did not find any match for $A_i$ then
  - if $j < |\mathcal{A}|$ we proceed to $A_{i+1}$, and
  - otherwise, we proceed to $k + 1$.

Given a set $\{L_1, \ldots, L_k\}$ of lists of independent sets of uniform cardinalities, if there is a set of sets $\{\{C_i, \ldots, C_k\} | C_i \in L_i, i \in (1, k)\}$, this algorithm will give us one of these.

(Of course, operations like these are only practical for relatively small graphs. The graphs of interests are the ones that are too large for naïve coloring algorithms, but small enough for the algorithms described above to work in reasonable time—i.e. seconds and minutes rather than days and weeks.)

Reference

[1] R. E. Tarjan and A. E. Trojanowski, "Finding a maximum independent set," SIAM J. Comput., vol. 3, pp. 537-546, 1977.