Speeding up compact planar graphs by using shallower trees†

Alexander Irribarra-Cortés [1,2] José Fuentes-Sepúlveda [1]
Diego Seco [1,2] Roberto Asín [1]

[1]Department of Computer Science, Universidad de Concepción, Chile

[2]Millennium Institute for Foundational Research on Data, Chile

March 24, 2022

†This work was funded by ANID Millennium Science Initiative Program - Code ICN17_002 (1st and 3rd authors), PFCHA/Doctorado Nacional/2021-21211768 (1st author), PAI grant 77190038 (2nd author), and CYTED grant 519RT0579 (3rd author).

1. Introduction
2. Related work
   - Compact data structures
   - Succinct ordinal trees
   - Planar embeddings
3. Our contribution
   - Effect of the trees' topology
   - Computing shallower spanning trees
   - Alternative data structures
4. Experimental analysis
5. Conclusions and future work

1 Introduction

2 Related work
- Compact data structures
- Succinct ordinal trees
- Planar embeddings

3 Our contribution
- Effect of the trees' topology
- Computing shallower spanning trees
- Alternative data structures

4 Experimental analysis

5 Conclusions and future work

Problem: Compact representation of planar graphs.

- A planar graph can be embedded into the plane with no edges crossing each other.
- A planar graph may have more than one planar embedding.
- Two planar embeddings of the same graph are differentiated by the orientation or order of their edges.

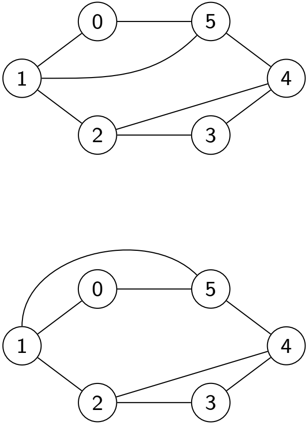
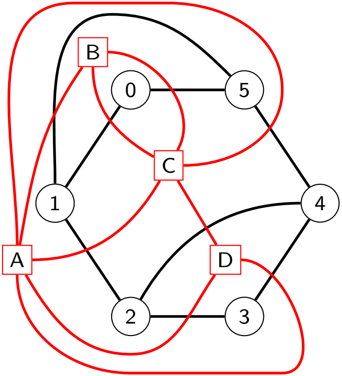


Figure: Two planar embeddings of the same planar graph.

Problem

- In some applications a specific embedding may be of special interest.
- A planar embedding is associated with a primal graph **(vertices)** and a dual graph (faces).
- It is wanted to be able to support queries of the primal graph and dual graph.



1 Introduction

2 Related work
- Compact data structures
- Succinct ordinal trees
- Planar embeddings

3 Our contribution
- Effect of the trees' topology
- Computing shallower spanning trees
- Alternative data structures

4 Experimental analysis

5 Conclusions and future work

Bitvectors

Sequence of bits B supporting the following operations:

- $access(B, i)$: Return the value of the i-th bit.
- $rank_v(B, i)$: Return the amount of bits with value v until position i.
- $select_v(B, i)$: Return the position of the i-th occurrence of v.

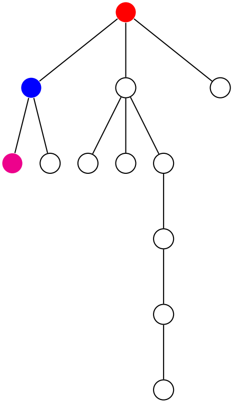
$$B = \underline{0110\ 1}101\ 1010\ 1100$$
$$rank_1(B, 4) = 3$$
$$select_0(B, 3) = 6$$

Trees as balanced parentheses sequence

An ordinal tree can be encoded as a balanced parentheses sequence (Jacobson, 1989) obtained with a depth first traversal:

1. The symbol '(' is written the first time an edge is visited.
2. The children nodes are processed.
3. The symbol ')' is written the second time an edge is visited.

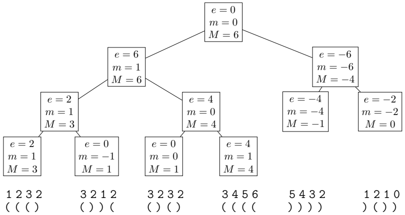
2 bits per node.

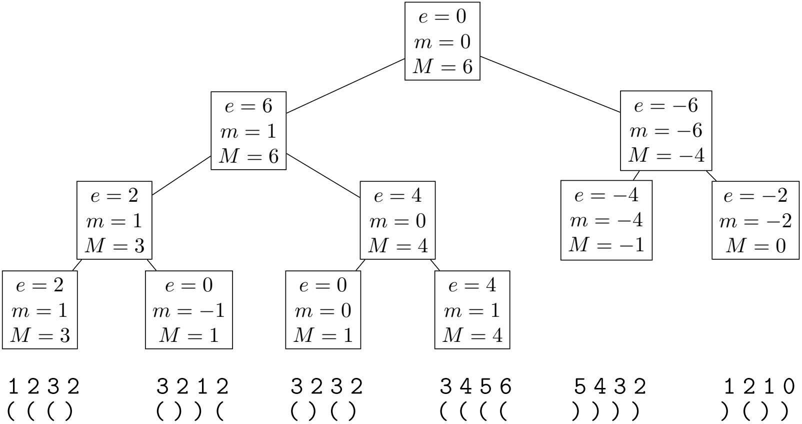


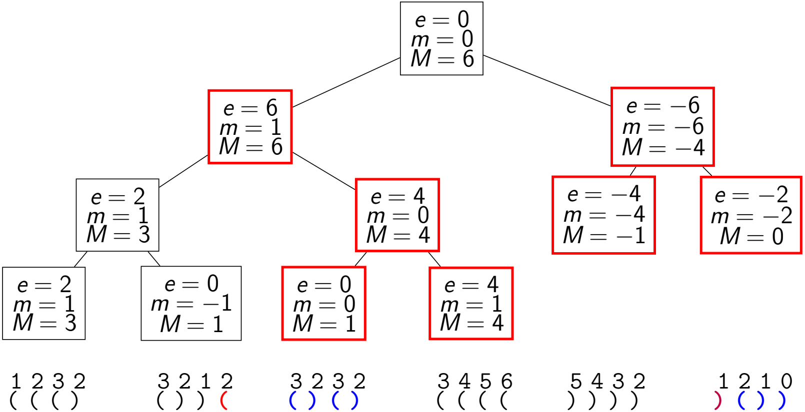
((()())(()()(((())))()))
111010011010111100000100

Fully Functional (Navarro and Sadakane, 2014)

- Supports multiple operations (*e.g. match, enclose, height, etc*).
- Uses the auxiliary data structure *Range min-Max Tree*. It divides the sequence into blocks.
- Leaves store information about the blocks, while internal nodes store aggregated information about their children.




e = 0
m = 0
M = 6
e = 6
m = 1
M = 6
e = −6
m = −6
M = −4
e = 2
m = 1
M = 3
e = 4
m = 0
M = 4
e = −4
m = −4
M = −1
e = −2
m = −2
M = 0
e = 2
m = 1
M = 3
e = 0
m = −1
M = 1
e = 0
m = 0
M = 1
e = 4
m = 1
M = 4
1 2 3 2
((( )
3 2 1 2
( )) (
3 2 3 2
( ) ( )
3 4 5 6
((((
5 4 3 2
))))
1 2 1 0
) ( ))


e = 0 m = 0 M = 6
e = 6 m = 1 M = 6
e = −6 m = −6 M = −4
e = 2 m = 1 M = 3
e = 4 m = 0 M = 4
e = −4 m = −4 M = −1
e = −2 m = −2 M = 0
e = 2 m = 1 M = 3
e = 0 m = −1 M = 1
e = 0 m = 0 M = 1
e = 4 m = 1 M = 4
1 2 3 2
((( )
3 2 1 2
( )) (
3 2 3 2
( ) ( )
3 4 5 6
((((
5 4 3 2
))))
1 2 1 0
) ( ))

Range min Tree (Grossi and Ottaviano, 2015)

- Variation of the *Range min-Max Tree* in which the maximum is not stored.
- It uses other precomputed tables to speed up the search inside a block.
- Faster than the *Range min-Max Tree* in practice.

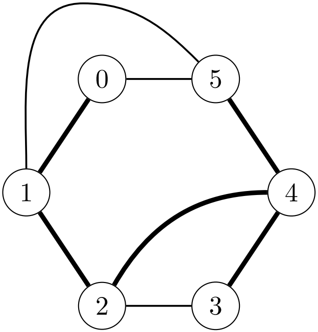
Compact planar embeddings

Year, Author(s)	Memory (bits)	Supports operations
1984, Turán	$4m$	No
1989, Jacobson	$O(m)$	Yes
1995, Keeler and Westbrook	$3.58m + O(1)$	No
2001, Munro and Raman	$2m + 8n + o(n)$	Yes
2010, Blelloch and Farzan	$3.58m + o(m)$	Yes
2020, Ferres *et al.*	$4m + o(m)$	Yes

State of the art

Ferres *et al.* representation.

- Based on Turán's representation.
- Built with a spanning tree T of the primal graph.

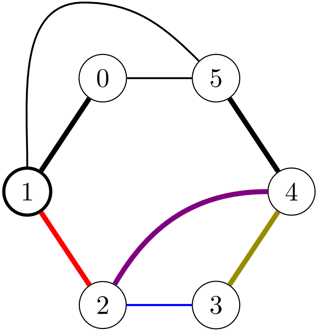


Turán's representation

A counterclockwise DFS traversal over G is performed:

- When an edge $e \in T$ is visited, the symbol '(' is written the first time, and ')' the second time. This path is followed.
- When an edge $e \notin T$ is visited, the symbol '[' is written the first time, and ']' the second time.

4 bits per edge.

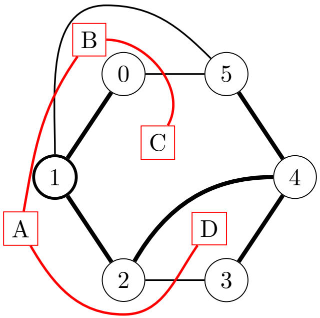


([(( ]) ([[ ))) ( ])]

Ferres *et al.* representation

The sequence is decomposed in three parts, exploiting the fact that $G - T$ is homologous to a spanning tree over the dual T'.

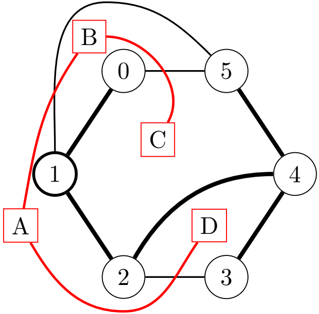
- A: A bitvector encoding the interleaving between T and T'.
- B: The tree T encoded as a balanced parentheses sequence.
- B^*: The tree T' encoded as a balanced parentheses sequence.
- The spanning tree T used in the representation is computed with a DFS traversal.



	(	[	(	(	]	)	(	[	[	)	)	)	(	]	)	]
A	1	0	1	1	0	1	1	0	0	1	1	1	1	0	1	0
B	(		(	(		)	(			)	)	)	(		)	
B*		[			]			[	[					]		]

Spanning trees T and T'

- The spanning tree T is mostly used for queries on the primal graph.
- The spanning tree T' is mostly used for queries on the dual graph.
- Changing the topology of one of the trees will change the topology of the other tree.




```
      (  [  (  (  ]  )  (  [  [  )  )  )  (  ]  )  ]
A     1  0  1  1  0  1  1  0  0  1  1  1  1  0  1  0
B     (     (  (     )  (        )  )  )  (     )
B*       [        ]        [  [              ]     ]
```

1 Introduction

2 Related work
- Compact data structures
- Succinct ordinal trees
- Planar embeddings

3 Our contribution
- Effect of the trees' topology
- Computing shallower spanning trees
- Alternative data structures

4 Experimental analysis

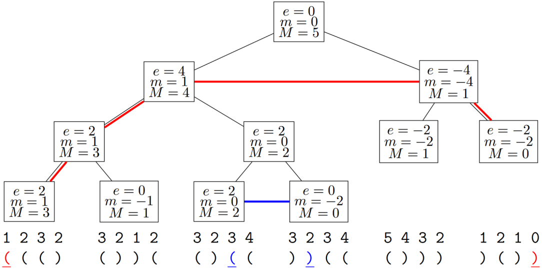
5 Conclusions and future work

Effect of the trees' topology

- Time complexity of *matching* query is $O(\log \frac{b-a}{B})$ when the matching parentheses are in positions a and b (Navarro, 2016, Section 7.2).

Effect of the trees' topology

- Time complexity of *matching* query is $O(\log \frac{b-a}{B})$ when the matching parentheses are in positions a and b (Navarro, 2016, Section 7.2).
- Therefore, to speed up the operations we must reduce the distances.



Computing shallower spanning trees

We evaluated the following methods to compute the spanning trees

Computing shallower spanning trees

We evaluated the following methods to compute the spanning trees

- $\mathtt{DFS}_{\mathtt{primal}}$: DFS traversal over the vertices of the primal graph.

Computing shallower spanning trees

We evaluated the following methods to compute the spanning trees

- $\texttt{DFS}_{\texttt{primal}}$: DFS traversal over the vertices of the primal graph.
- $\texttt{BFS}_{\texttt{primal}}$: BFS traversal over the vertices of the primal graph.

Computing shallower spanning trees

We evaluated the following methods to compute the spanning trees

- $\texttt{DFS}_{\texttt{primal}}$: DFS traversal over the vertices of the primal graph.
- $\texttt{BFS}_{\texttt{primal}}$: BFS traversal over the vertices of the primal graph.
- $\texttt{BFS}_{\texttt{dual}}$: BFS traversal over the faces of the dual graph.

Computing shallower spanning trees

We evaluated the following methods to compute the spanning trees

- $\texttt{DFS}_{\texttt{primal}}$: DFS traversal over the vertices of the primal graph.
- $\texttt{BFS}_{\texttt{primal}}$: BFS traversal over the vertices of the primal graph.
- $\texttt{BFS}_{\texttt{dual}}$: BFS traversal over the faces of the dual graph.
- $\texttt{DFS}_{\texttt{x}}$: DFS traversal over the vertices of the primal graph, limiting the height to x times the height obtained with the method $\texttt{BFS}_{\texttt{primal}}$.

Computing shallower spanning trees

We evaluated the following methods to compute the spanning trees

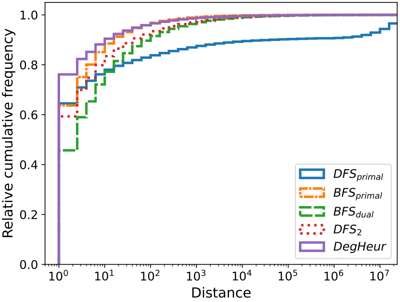
- $\texttt{DFS}_{\texttt{primal}}$: DFS traversal over the vertices of the primal graph.
- $\texttt{BFS}_{\texttt{primal}}$: BFS traversal over the vertices of the primal graph.
- $\texttt{BFS}_{\texttt{dual}}$: BFS traversal over the faces of the dual graph.
- $\texttt{DFS}_{\texttt{x}}$: DFS traversal over the vertices of the primal graph, limiting the height to x times the height obtained with the method $\texttt{BFS}_{\texttt{primal}}$.
- `Alt`: Alternate computation of BFS traversal over the primal graph and the dual graph.

Computing shallower spanning trees

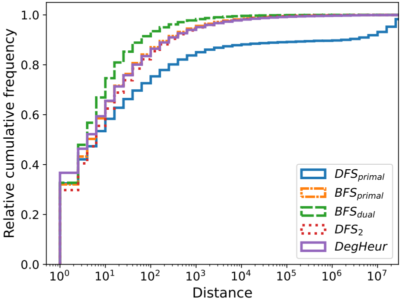
We evaluated the following methods to compute the spanning trees

- $\texttt{DFS}_{\texttt{primal}}$: DFS traversal over the vertices of the primal graph.
- $\texttt{BFS}_{\texttt{primal}}$: BFS traversal over the vertices of the primal graph.
- $\texttt{BFS}_{\texttt{dual}}$: BFS traversal over the faces of the dual graph.
- $\texttt{DFS}_{\texttt{x}}$: DFS traversal over the vertices of the primal graph, limiting the height to x times the height obtained with the method $\texttt{BFS}_{\texttt{primal}}$.
- `Alt`: Alternate computation of BFS traversal over the primal graph and the dual graph.
- `DegHeur`: Heuristical traversal using a degree ordered heap.

Computing shallower spanning trees



Distances in B



Distances in B^*

Figure: Distance histograms using different methods on the dataset *tiger_map*.

Average distances

Dataset	$\texttt{DFS}_{\texttt{primal}}$		$\texttt{BFS}_{\texttt{primal}}$		$\texttt{BFS}_{\texttt{dual}}$		`Alt`		$\texttt{DFS}_2$		`DegHeur`	
	B	B^*	B	B^*	B	B^*	B	B^*	B	B^*	B	B^*
PE1M	263.96	345.84	**0.57**	5.65	3.11	**1.30**	**0.57**	5.65	2.34	6.42	4.78	12.30
PE5M	1261.52	1612.13	**1.17**	12.07	6.03	**2.68**	**1.17**	12.07	5.12	13.22	7.39	33.49
PE10M	2547.27	3265.26	**1.86**	20.55	9.69	**4.29**	**1.86**	20.55	7.01	16.41	11.88	47.00
PE25M	6440.49	8246.12	**2.87**	32.22	15.16	**6.66**	**2.87**	32.22	11.23	25.67	19.39	102.99
tiger_map	1882.50	2482.99	**0.27**	26.06	5.80	**1.96**	**0.27**	26.06	2.03	28.15	2.07	44.82

Table: Average parentheses distance for B and B^*. Values are multiplied by 10^{-3}.

Using alternative data structures

- We tested using newer CDS for balanced parentheses sequences, specifically the range min tree (Grossi and Ottaviano, 2015).
- We used both the implementation available in the `SUCCINCT` library and a variation of the range min-max tree of the SDSL library which doesn't store the maximums*.

*This is not a full implementation of the range min tree

1 Introduction

2 Related work
- Compact data structures
- Succinct ordinal trees
- Planar embeddings

3 Our contribution
- Effect of the trees' topology
- Computing shallower spanning trees
- Alternative data structures

4 Experimental analysis

5 Conclusions and future work

Setup

- We modified Ferres *et al.* implementation to plug in the approaches already described.
- The base implementation uses the C++ SDSL library (Gog et al., 2014).
- We also tested using the range min tree implementation in the `SUCCINCT` library.

Setup

- The code was compiled using GCC 4.8.4 with the optimization flag -O3.
- We measured time using the `clock` function.
- The experiments were carried out on a machine with Intel Core i7-3820 processor clocked at 3.60 GHz. This processor has per–core L1 and L2 cachesof 32KB and 256KB respectively, and a shared cache of 10MB. The machine runs Linux 3.13.0-86-generic and has 32GB of DDR3 RAM.

Datasets

We used both synthetic and real datasets with different numbers of nodes, which are available at
http://www.inf.udec.cl/~jfuentess/datasets/graphs.php

Dataset	Vertices	Edges
PE1M	1.000.000	2.999.978
PE5M	5.000.000	14.999.983
PE10M	10.000.000	29.999.979
PE25M	25.000.000	74.999.979
tiger_map	19.785.187	43.903.023

Datasets used in the experimental analysis

Implementations

Implementation	Description
succinct	Uses only CDS from succinct
SDSL	Uses only CDS from SDSL
minTree	Uses the *range min tree* from succinct, and SDSL on the remaining CDS
v5	Variation that uses *rank_support_v5* from SDSL for *rank* support
$\mathtt{SDSL}_{\mathtt{min}}$	Modification of the *range min max tree* from SDSL which doesn't store the maximums

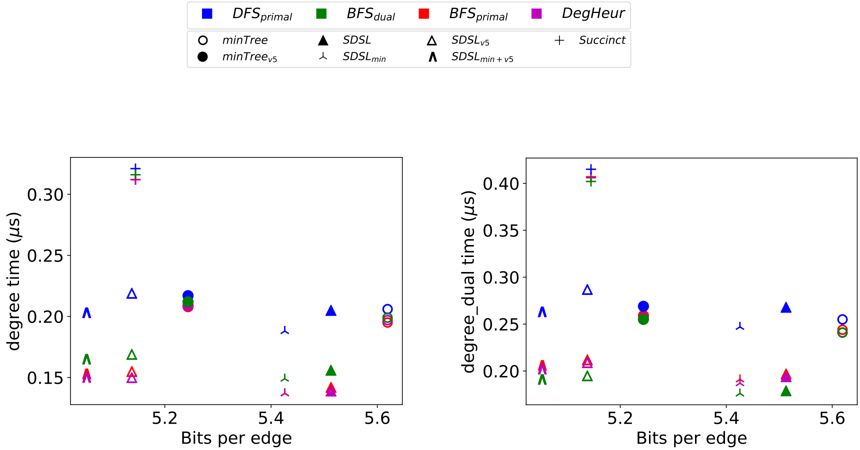
Results

- We evaluated operations *degree*, *listing*, *face* and *dfs*.
- All the operations were performed on the primal graph and dual graph.
- We averaged 15 repetitions over all vertices/edges except for operation *dfs*, in which we averaged over 5 starting vertices.
- We measured using three starting edges for the construction, and we report the median of the averages.

Results (tiger_map)

Method	Queries on the primal				Queries on the dual			Space
	dfs	degree	neighbors	face	dfs	degree	neighbors	
$\texttt{BFS}_{\texttt{primal}}$ `minTree`	13.00	0.195	0.540	0.608	14.09	0.244	0.610	30.83
$\texttt{BFS}_{\texttt{primal}}$ `SDSL`	16.59	0.142	0.621	0.718	18.89	0.197	0.724	30.25
$\texttt{BFS}_{\texttt{primal}}$ $\texttt{SDSL}_{\texttt{min}}$	16.10	0.137	0.594	0.688	18.05	0.191	0.688	29.78
$\texttt{DFS}_{\texttt{primal}}$ `minTree`	14.44	0.206	0.617	0.687	15.79	0.255	0.689	30.83
$\texttt{DFS}_{\texttt{primal}}$ `SDSL`	27.43	0.205	1.177	1.309	29.67	0.268	1.265	30.25
$\texttt{DFS}_{\texttt{primal}}$ $\texttt{SDSL}_{\texttt{min}}$	24.29	0.188	1.013	1.133	26.39	0.247	1.102	29.78
$\texttt{DFS}_{\texttt{primal}}$ $\texttt{SDSL}_{\texttt{min+v5}}$	26.11	0.203	1.064	1.198	28.39	0.263	1.158	27.72
$\texttt{BFS}_{\texttt{dual}}$ `minTree`	13.46	0.199	0.561	0.620	13.98	0.241	0.598	30.83
$\texttt{BFS}_{\texttt{dual}}$ `SDSL`	18.18	0.156	0.677	0.767	17.89	0.179	0.664	30.25
$\texttt{BFS}_{\texttt{dual}}$ $\texttt{SDSL}_{\texttt{min}}$	17.41	0.149	0.638	0.728	17.26	0.176	0.639	29.78
`DegHeur minTree`	12.77	0.197	0.538	0.608	14.06	0.241	0.601	30.83
`DegHeur SDSL`	16.59	0.139	0.640	0.763	18.90	0.194	0.729	30.25
`DegHeur` $\texttt{SDSL}_{\texttt{min}}$	15.93	0.137	0.612	0.719	18.05	0.187	0.689	29.78

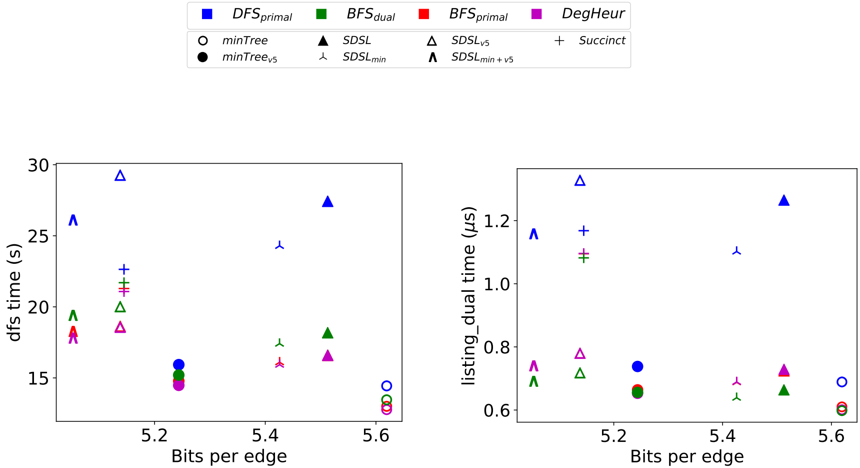
Trade–off (tiger_map)



Trade–off *degree* primal

Trade–off *degree* dual

Trade–off (tiger_map)



Trade–off *dfs* primal

Trade–off *neighbors* dual

1 Introduction

2 Related work
- Compact data structures
- Succinct ordinal trees
- Planar embeddings

3 Our contribution
- Effect of the trees' topology
- Computing shallower spanning trees
- Alternative data structures

4 Experimental analysis

5 Conclusions and future work

Conclusions

- The topology of the spanning trees used in compact representations of planar graphs may have an important impact in the performance of the CDS in practice.

Conclusions

- The topology of the spanning trees used in compact representations of planar graphs may have an important impact in the performance of the CDS in practice.
- Using shallower trees reduces the average distance between the pair of parentheses encoding a node in a BPS. Therefore, CDS such as the *range min-max tree* can perform faster.

Conclusions

- The topology of the spanning trees used in compact representations of planar graphs may have an important impact in the performance of the CDS in practice.
- Using shallower trees reduces the average distance between the pair of parentheses encoding a node in a BPS. Therefore, CDS such as the *range min-max tree* can perform faster.
- Using new implementations of CDS for BPS may allow further improvements.
- Overall, our new methods can save up to 8% of space while queries may be twice as fast.

Conclusions

- The topology of the spanning trees used in compact representations of planar graphs may have an important impact in the performance of the CDS in practice.
- Using shallower trees reduces the average distance between the pair of parentheses encoding a node in a BPS. Therefore, CDS such as the *range min-max tree* can perform faster.
- Using new implementations of CDS for BPS may allow further improvements.
- Overall, our new methods can save up to 8% of space while queries may be twice as fast.
- In general, queries on the primal are faster when we use the degree heuristic or a BFS traversal on the primal graph.

Conclusions

- The topology of the spanning trees used in compact representations of planar graphs may have an important impact in the performance of the CDS in practice.
- Using shallower trees reduces the average distance between the pair of parentheses encoding a node in a BPS. Therefore, CDS such as the *range min-max tree* can perform faster.
- Using new implementations of CDS for BPS may allow further improvements.
- Overall, our new methods can save up to 8% of space while queries may be twice as fast.
- In general, queries on the primal are faster when we use the degree heuristic or a BFS traversal on the primal graph.
- Similarly, queries on the dual are faster when the BFS is performed on the dual graph.

Future work

- To find a method which gives a balanced point for queries on the primal and dual graph.

Future work

- To find a method which gives a balanced point for queries on the primal and dual graph.
- To identify new parameters that affect the performance of the data structure.

Future work

- To find a method which gives a balanced point for queries on the primal and dual graph.
- To identify new parameters that affect the performance of the data structure.
- To study the effect of the topology on balanced parentheses representations that are not based on trees.

DEPARTAMENTO INGENIERÍA INFORMÁTICA Y CIENCIAS DE LA COMPUTACIÓN
FACULTAD DE INGENIERÍA UNIVERSIDAD DE CONCEPCIÓN

Speeding up compact planar graphs by using shallower trees[†]

Alexander Irribarra-Cortés [1,2] José Fuentes-Sepúlveda [1]
Diego Seco [1,2] Roberto Asín [1]

[1]Department of Computer Science, Universidad de Concepción, Chile

[2]Millennium Institute for Foundational Research on Data, Chile

March 24, 2022

[†]This work was funded by ANID Millennium Science Initiative Program - Code ICN17_002 (1st and 3rd authors), PFCHA/Doctorado Nacional/2021-21211768 (1st author), PAI grant 77190038 (2nd author), and CYTED grant 519RT0579 (3rd author).