Drawing Binary Tanglegrams: An Experimental Evaluation

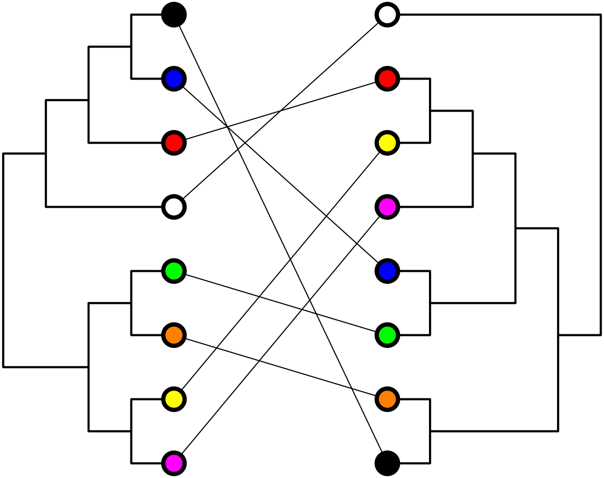
Martin Nöllenburg[1] Markus Völker[1]
Alexander Wolff[2] Danny Holten[2]

[1]Karlsruhe University, Germany

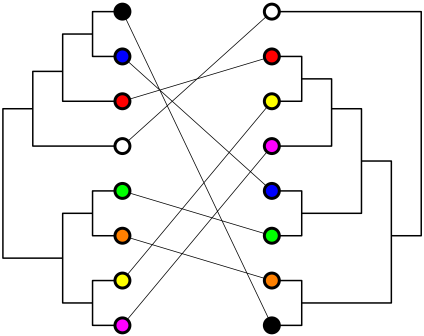
[2]TU Eindhoven, The Netherlands

Workshop on Algorithm Engineering and Experiments, New York, January 3, 2009

A Tanglegram



A Tanglegram



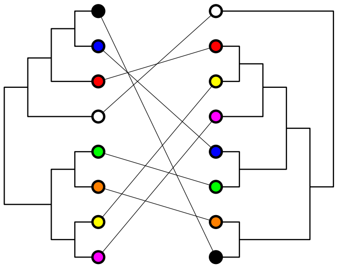
- face-to-face drawing of two binary trees
- trees have identical leaves
- leaves are connected by straight *inter-tree edges*
- visual tool for exploring hierarchical data
  - phylogenetic trees
  - clustering dendrograms
  - ...

The Binary Tanglegram Layout Problem

Problem: Binary Tanglegram Layout (TL)

Input: binary trees S and T with the same sets of n leaves

Output: plane face-to-face drawings of S and T that minimize the number of inter-tree edge crossings

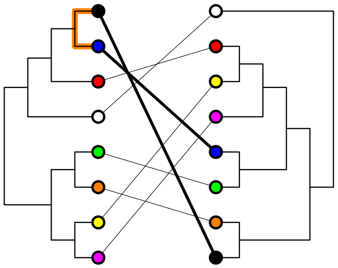


The Binary Tanglegram Layout Problem

Problem: Binary Tanglegram Layout (TL)

Input: binary trees S and T with the same sets of n leaves

Output: plane face-to-face drawings of S and T that minimize the number of inter-tree edge crossings



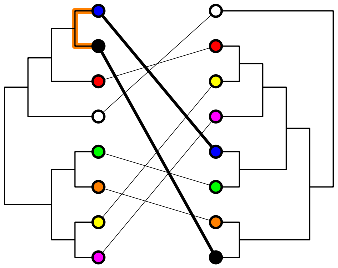
16 crossings

The Binary Tanglegram Layout Problem

Problem: Binary Tanglegram Layout (TL)

Input: binary trees S and T with the same sets of n leaves

Output: plane face-to-face drawings of S and T that minimize the number of inter-tree edge crossings



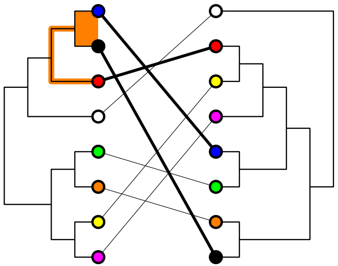
15 crossings

The Binary Tanglegram Layout Problem

Problem: Binary Tanglegram Layout (TL)

Input: binary trees S and T with the same sets of n leaves

Output: plane face-to-face drawings of S and T that minimize the number of inter-tree edge crossings



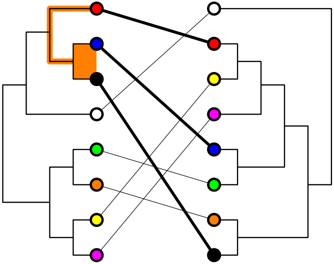
15 crossings

The Binary Tanglegram Layout Problem

Problem: Binary Tanglegram Layout (TL)

Input: binary trees S and T with the same sets of n leaves

Output: plane face-to-face drawings of S and T that minimize the number of inter-tree edge crossings



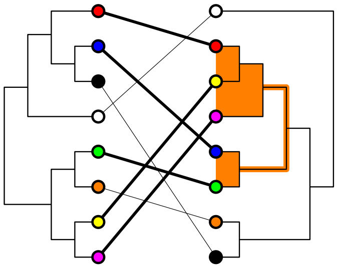
13 crossings

The Binary Tanglegram Layout Problem

Problem: Binary Tanglegram Layout (TL)

Input: binary trees S and T with the same sets of n leaves

Output: plane face-to-face drawings of S and T that minimize the number of inter-tree edge crossings



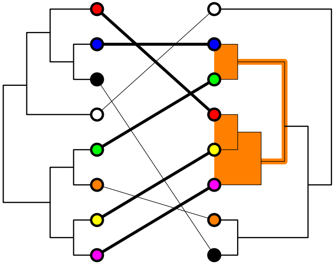
13 crossings

The Binary Tanglegram Layout Problem

Problem: Binary Tanglegram Layout (TL)

Input: binary trees S and T with the same sets of n leaves

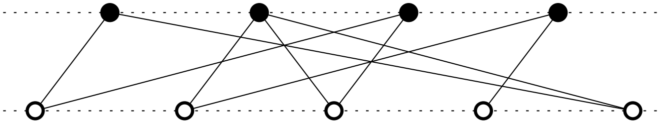
Output: plane face-to-face drawings of S and T that minimize the number of inter-tree edge crossings



11 crossings

A Related Problem

Two-layer crossing minimization [Sugiyama, Tagawa, Toda '81]



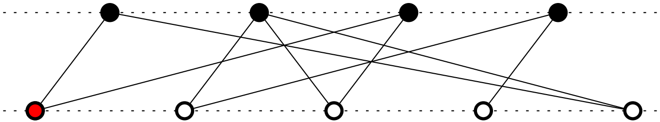
- NP-hard even if one layer is fixed [Eades, Wormald '94]
- (variant of) barycenter heuristic yields 3-approximation

Differences to TL:

- arbitrary vertex degree
- vertex orders not restricted by underlying trees

A Related Problem

Two-layer crossing minimization [Sugiyama, Tagawa, Toda '81]



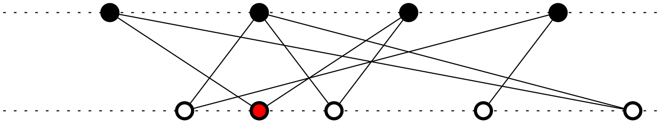
- NP-hard even if one layer is fixed [Eades, Wormald '94]
- (variant of) barycenter heuristic yields 3-approximation

Differences to TL:

- arbitrary vertex degree
- vertex orders not restricted by underlying trees

A Related Problem

Two-layer crossing minimization [Sugiyama, Tagawa, Toda ’81]



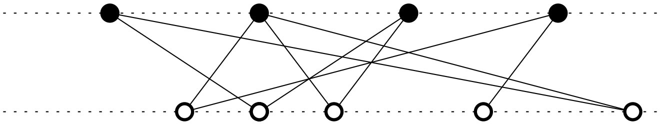
- NP-hard even if one layer is fixed [Eades, Wormald ’94]
- (variant of) barycenter heuristic yields 3-approximation

Differences to TL:

- arbitrary vertex degree
- vertex orders not restricted by underlying trees

A Related Problem

Two-layer crossing minimization [Sugiyama, Tagawa, Toda '81]



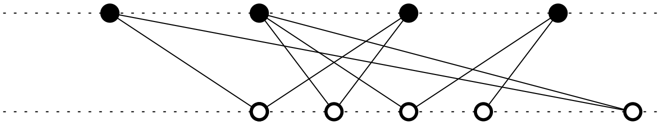
- NP-hard even if one layer is fixed [Eades, Wormald '94]
- (variant of) barycenter heuristic yields 3-approximation

Differences to TL:

- arbitrary vertex degree
- vertex orders not restricted by underlying trees

A Related Problem

Two-layer crossing minimization [Sugiyama, Tagawa, Toda ’81]



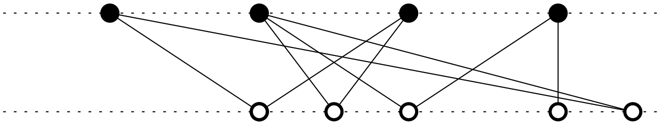
- NP-hard even if one layer is fixed [Eades, Wormald ’94]
- (variant of) barycenter heuristic yields 3-approximation

Differences to TL:

- arbitrary vertex degree
- vertex orders not restricted by underlying trees

A Related Problem

Two-layer crossing minimization [Sugiyama, Tagawa, Toda '81]



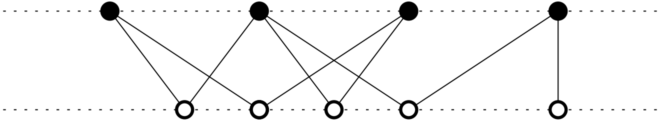
- NP-hard even if one layer is fixed [Eades, Wormald '94]
- (variant of) barycenter heuristic yields 3-approximation

Differences to TL:

- arbitrary vertex degree
- vertex orders not restricted by underlying trees

A Related Problem

Two-layer crossing minimization [Sugiyama, Tagawa, Toda '81]



- NP-hard even if one layer is fixed [Eades, Wormald '94]
- (variant of) barycenter heuristic yields 3-approximation

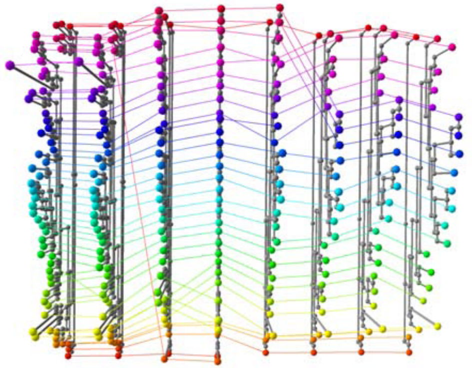
Differences to TL:

- arbitrary vertex degree
- vertex orders not restricted by underlying trees

Previous Work

[Dwyer, Schreiber '04]

- stacked tanglegram layout of ≥ 2 trees
- one-sided TL in $O(n^2 \log n)$ time



Previous Work

[Dwyer, Schreiber '04]

- stacked tanglegram layout of ≥ 2 trees
- one-sided TL in $O(n^2 \log n)$ time

[Fernau, Kaufmann, Poths '05]

- TL is NP-hard
- one-sided TL in $O(n \log^2 n)$ time
- FPT algorithm for TL

Previous Work

[Dwyer, Schreiber '04]

- stacked tanglegram layout of ≥ 2 trees
- one-sided TL in $O(n^2 \log n)$ time

[Fernau, Kaufmann, Poths '05]

- TL is NP-hard
- one-sided TL in $O(n \log^2 n)$ time
- FPT algorithm for TL

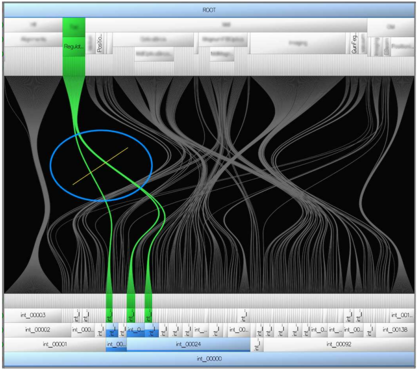
[Zainon, Calder '06]

- interactive tree comparison tool
- no explicit crossing minimization

Previous Work (cont'd)

[Holten, van Wijk '08]

- tanglegram visualization tool for arbitrary (large) trees
- crossing reduction heuristic based on barycentric method



Previous Work (cont'd)

[Holten, van Wijk '08]

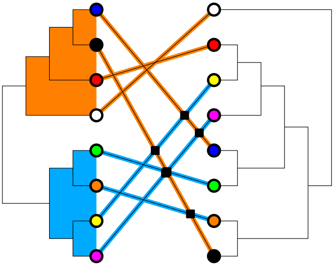
- tanglegram visualization tool for arbitrary (large) trees
- crossing reduction heuristic based on barycentric method

[Buchin^2, Byrka, Nöllenburg, Okamoto, Silveira, Wolff '08]

- TL remains NP-hard for *complete* binary trees
- 2-approximation and FPT algorithm for this case
- TL is hard to approximate to any constant factor
  [under a widely accepted assumption]
- max-version of dual of TL has 0.878-approximation

Algorithms

One-Sided TL (1STL) [Dwyer, Schreiber ’04]



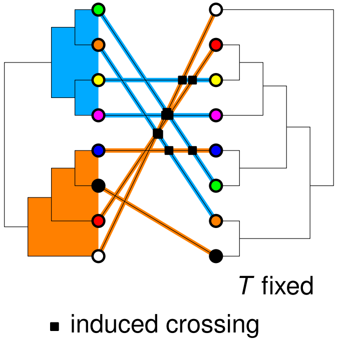
T fixed

▪ induced crossing

`iterated 1STL`(S, T)

while *layout improves* **do**
- fix leaf order of T
- **foreach** *internal node v of S* **do**
  - decide whether to swap v
- fix leaf order of S
- **foreach** *internal node w of T* **do**
  - decide whether to swap w

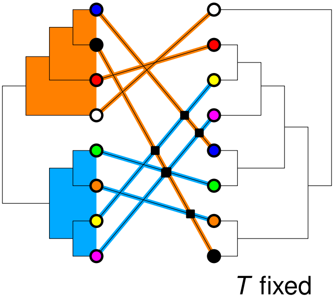
One-Sided TL (1STL) [Dwyer, Schreiber '04]



`iterated 1STL`(S, T)

while *layout improves* **do**
- fix leaf order of T
- **foreach** *internal node v of S* **do**
  - decide whether to swap v
- fix leaf order of S
- **foreach** *internal node w of T* **do**
  - decide whether to swap w

One-Sided TL (1STL) [Dwyer, Schreiber '04]

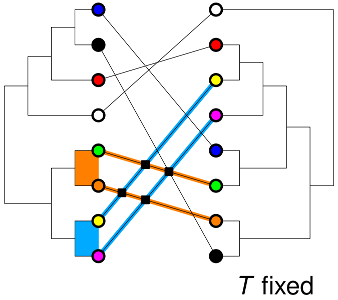


■ induced crossing

`iterated 1STL`(S, T)

while *layout improves* **do**
- fix leaf order of T
- **foreach** *internal node v of S* **do**
  - decide whether to swap v
- fix leaf order of S
- **foreach** *internal node w of T* **do**
  - decide whether to swap w

One-Sided TL (1STL) [Dwyer, Schreiber '04]



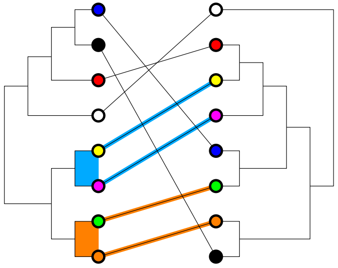
▪ induced crossing

`iterated 1STL`(S, T)

while *layout improves* **do**

- fix leaf order of T
- **foreach** *internal node v of S* **do**
  - decide whether to swap v
- fix leaf order of S
- **foreach** *internal node w of T* **do**
  - decide whether to swap w

One-Sided TL (1STL) [Dwyer, Schreiber '04]

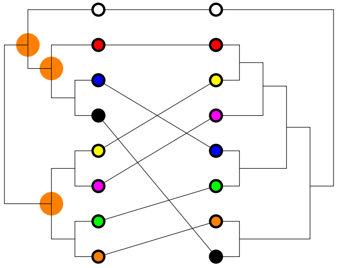


- induced crossing

`iterated 1STL`(S, T)

while *layout improves* **do**
  fix leaf order of T
  foreach *internal node v of S* **do**
    decide whether to swap v
  fix leaf order of S
  foreach *internal node w of T* **do**
    decide whether to swap w

One-Sided TL (1STL) [Dwyer, Schreiber '04]

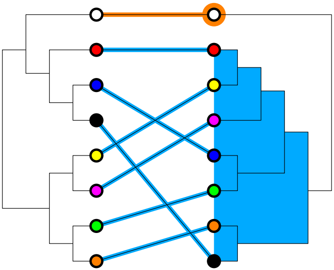


- induced crossing

`iterated 1STL`(S, T)

while *layout improves* **do**
- fix leaf order of T
- **foreach** *internal node* v *of* S **do**
  - decide whether to swap v
- fix leaf order of S
- **foreach** *internal node* w *of* T **do**
  - decide whether to swap w

One-Sided TL (1STL) [Dwyer, Schreiber '04]



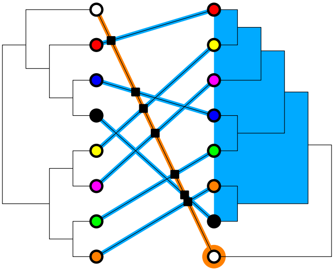
S fixed

- induced crossing

`iterated 1STL`(S, T)

while *layout improves* **do**
- fix leaf order of T
- **foreach** *internal node v of S* **do**
  - decide whether to swap v
- fix leaf order of S
- **foreach** *internal node w of T* **do**
  - decide whether to swap w

One-Sided TL (1STL) [Dwyer, Schreiber ’04]



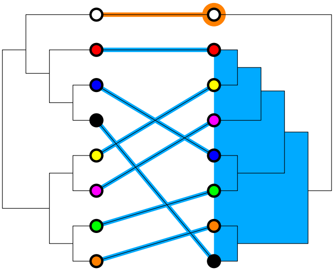
S fixed

▪ induced crossing

`iterated 1STL`(S, T)

while *layout improves* **do**
- fix leaf order of T
- **foreach** *internal node v of S* **do**
  - decide whether to swap v
- fix leaf order of S
- **foreach** *internal node w of T* **do**
  - decide whether to swap w

One-Sided TL (1STL) [Dwyer, Schreiber '04]



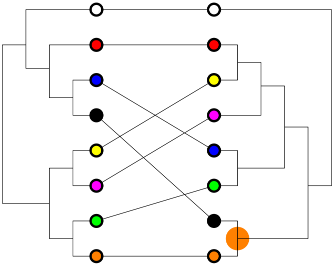
S fixed

- induced crossing

`iterated 1STL`(S, T)

while *layout improves* **do**
- fix leaf order of T
- **foreach** *internal node* v *of* S **do**
  - decide whether to swap v
- fix leaf order of S
- **foreach** *internal node* w *of* T **do**
  - decide whether to swap w

One-Sided TL (1STL) [Dwyer, Schreiber '04]



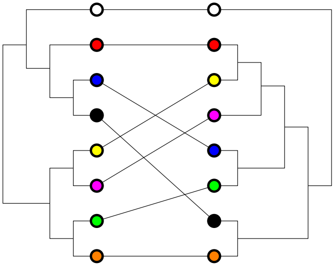
S fixed

- induced crossing

`iterated 1STL`(S, T)

while *layout improves* **do**
  fix leaf order of T
  foreach *internal node v of S* **do**
    decide whether to swap v
  fix leaf order of S
  foreach *internal node w of T* **do**
    decide whether to swap w

One-Sided TL (1STL) [Dwyer, Schreiber '04]

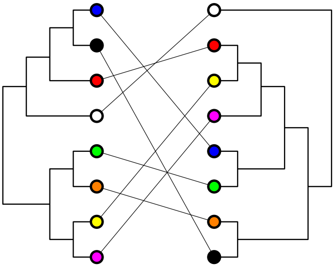


`iterated 1STL`(S, T)

while *layout improves* **do**
- fix leaf order of T
- **foreach** *internal node v of S* **do**
  - decide whether to swap v
- fix leaf order of S
- **foreach** *internal node w of T* **do**
  - decide whether to swap w

- no quality guarantee
- originally $O(n^2 \log n)$ time
- improved to $O(n \log^2 n)$ time [Fernau et al. '05]

Hierarchy Sort [Holten, van Wijk ’08]



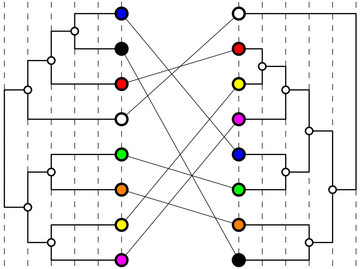
`hierarchy-sort`(S, T)

augment trees to equal height h

while *layout improves* **do**

- **for** $i = 2$ **to** h **do**
  - apply barycenter heuristic to level i of S and T
  - collapse level $i - 1$
- **for** $i = h$ **downto** 2 **do**
  - apply barycenter heuristic to level i of S and T
  - expand level $i - 1$

Hierarchy Sort [Holten, van Wijk ’08]

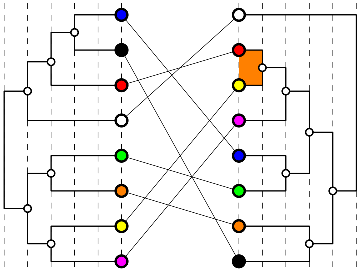


`hierarchy-sort`(S, T)

augment trees to equal height h

while *layout improves* **do**
- **for** $i = 2$ **to** h **do**
  - apply barycenter heuristic to level i of S and T
  - collapse level $i - 1$
- **for** $i = h$ **downto** 2 **do**
  - apply barycenter heuristic to level i of S and T
  - expand level $i - 1$

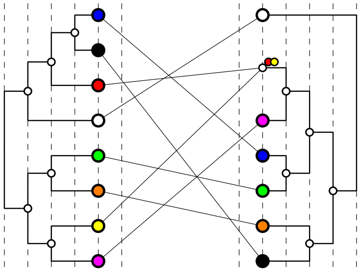
Hierarchy Sort [Holten, van Wijk '08]



`hierarchy-sort`(S, T)

augment trees to equal height h
while *layout improves* **do**
- **for** $i = 2$ **to** h **do**
  - apply barycenter heuristic to level i of S and T
  - collapse level $i - 1$
- **for** $i = h$ **downto** 2 **do**
  - apply barycenter heuristic to level i of S and T
  - expand level $i - 1$

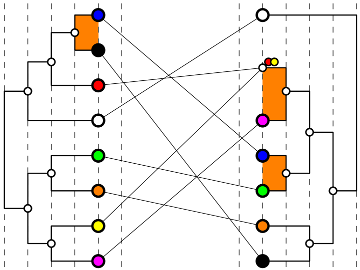
Hierarchy Sort [Holten, van Wijk '08]



`hierarchy-sort`(S, T)

augment trees to equal height h
while *layout improves* **do**
  for $i = 2$ **to** h **do**
    apply barycenter heuristic
      to level i of S and T
    collapse level $i - 1$
  for $i = h$ **downto** 2 **do**
    apply barycenter heuristic
      to level i of S and T
    expand level $i - 1$

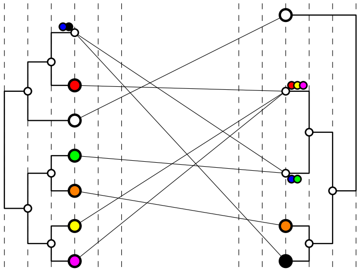
Hierarchy Sort [Holten, van Wijk '08]



`hierarchy-sort`(S, T)

augment trees to equal height h
while *layout improves* **do**
    for $i = 2$ **to** h **do**
        apply barycenter heuristic to level i of S and T
        collapse level $i - 1$
    for $i = h$ **downto** 2 **do**
        apply barycenter heuristic to level i of S and T
        expand level $i - 1$

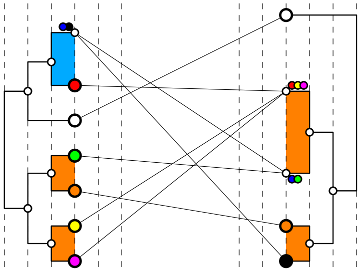
Hierarchy Sort [Holten, van Wijk '08]



`hierarchy-sort`(S, T)

augment trees to equal height h
while *layout improves* **do**
  for $i = 2$ **to** h **do**
    apply barycenter heuristic to level i of S and T
    collapse level $i - 1$
  for $i = h$ **downto** 2 **do**
    apply barycenter heuristic to level i of S and T
    expand level $i - 1$

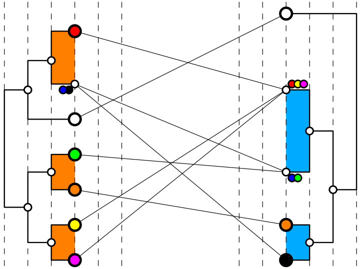
Hierarchy Sort [Holten, van Wijk '08]



`hierarchy-sort`(S, T)

augment trees to equal height h
while *layout improves* **do**
  for $i = 2$ **to** h **do**
    apply barycenter heuristic
      to level i of S and T
    collapse level $i - 1$
  for $i = h$ **downto** 2 **do**
    apply barycenter heuristic
      to level i of S and T
    expand level $i - 1$

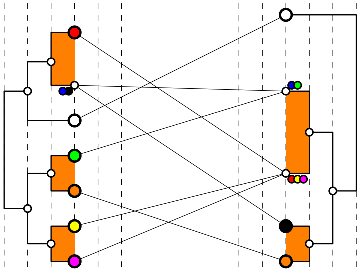
Hierarchy Sort [Holten, van Wijk '08]



`hierarchy-sort`(S, T)

augment trees to equal height h
while *layout improves* **do**
  for $i = 2$ **to** h **do**
    apply barycenter heuristic to level i of S and T
    collapse level $i - 1$
  for $i = h$ **downto** 2 **do**
    apply barycenter heuristic to level i of S and T
    expand level $i - 1$

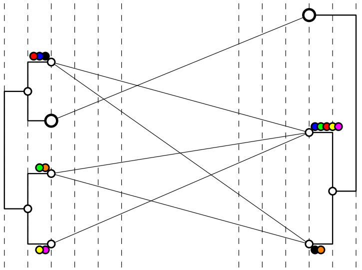
Hierarchy Sort [Holten, van Wijk '08]



hierarchy-sort(S, T)

augment trees to equal height h
while *layout improves* **do**
  for $i = 2$ **to** h **do**
    apply barycenter heuristic to level i of S and T
    collapse level $i - 1$
  for $i = h$ **downto** 2 **do**
    apply barycenter heuristic to level i of S and T
    expand level $i - 1$

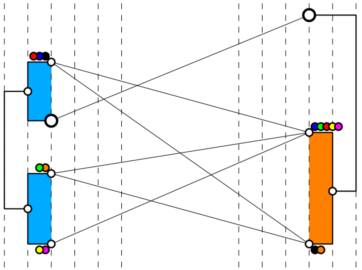
Hierarchy Sort [Holten, van Wijk '08]



`hierarchy-sort`(S, T)

augment trees to equal height h
while *layout improves* **do**
- **for** $i = 2$ **to** h **do**
  - apply barycenter heuristic to level i of S and T
  - collapse level $i - 1$
- **for** $i = h$ **downto** 2 **do**
  - apply barycenter heuristic to level i of S and T
  - expand level $i - 1$

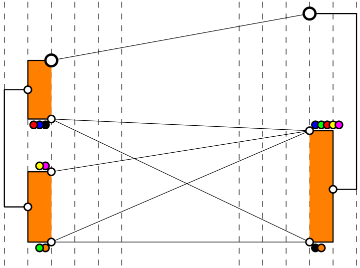
Hierarchy Sort [Holten, van Wijk '08]



`hierarchy-sort`(S, T)

augment trees to equal height h
while *layout improves* **do**
- **for** $i = 2$ **to** h **do**
  - apply barycenter heuristic to level i of S and T
  - collapse level $i - 1$
- **for** $i = h$ **downto** 2 **do**
  - apply barycenter heuristic to level i of S and T
  - expand level $i - 1$

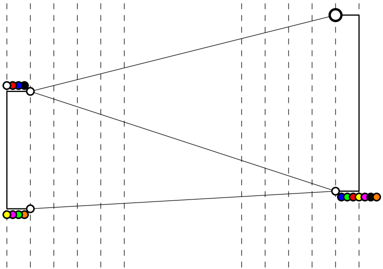
Hierarchy Sort [Holten, van Wijk '08]



`hierarchy-sort`(S, T)

augment trees to equal height h
while *layout improves* **do**
  for $i = 2$ **to** h **do**
    apply barycenter heuristic
      to level i of S and T
    collapse level $i - 1$
  for $i = h$ **downto** 2 **do**
    apply barycenter heuristic
      to level i of S and T
    expand level $i - 1$

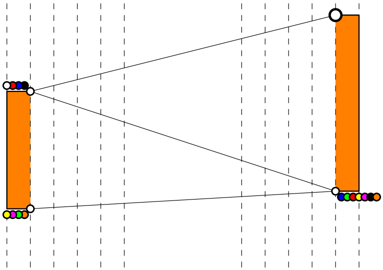
Hierarchy Sort [Holten, van Wijk ’08]



`hierarchy-sort`(S, T)

augment trees to equal height h
while *layout improves* **do**
  for $i = 2$ **to** h **do**
    apply barycenter heuristic to level i of S and T
    collapse level $i - 1$
  for $i = h$ **downto** 2 **do**
    apply barycenter heuristic to level i of S and T
    expand level $i - 1$

Hierarchy Sort [Holten, van Wijk '08]



`hierarchy-sort`(S, T)

augment trees to equal height h

while *layout improves* **do**

  for $i = 2$ **to** h **do**

    apply barycenter heuristic to level i of S and T

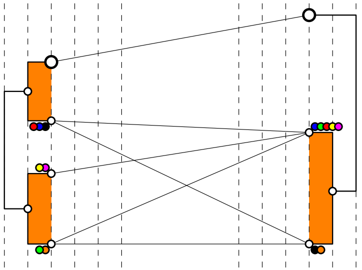
    collapse level $i - 1$

  for $i = h$ **downto** 2 **do**

    apply barycenter heuristic to level i of S and T

    expand level $i - 1$

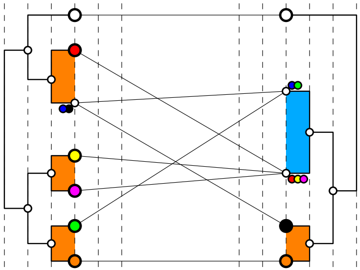
Hierarchy Sort [Holten, van Wijk '08]



`hierarchy-sort`(S, T)

augment trees to equal height h
while *layout improves* **do**
  for $i = 2$ **to** h **do**
    apply barycenter heuristic
      to level i of S and T
    collapse level $i - 1$
  for $i = h$ **downto** 2 **do**
    apply barycenter heuristic
      to level i of S and T
    expand level $i - 1$

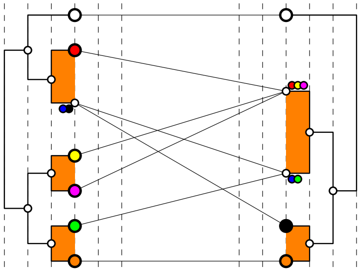
Hierarchy Sort [Holten, van Wijk '08]



`hierarchy-sort`(S, T)

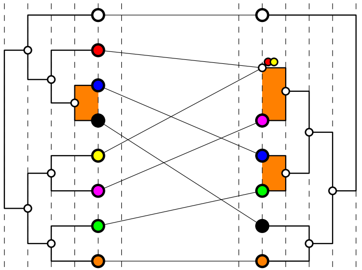
augment trees to equal height h
while *layout improves* **do**
  for $i = 2$ **to** h **do**
    apply barycenter heuristic
      to level i of S and T
    collapse level $i - 1$
  for $i = h$ **downto** 2 **do**
    apply barycenter heuristic
      to level i of S and T
    expand level $i - 1$

Hierarchy Sort [Holten, van Wijk '08]



```
hierarchy-sort(S, T)
augment trees to equal height h
while layout improves do
    for i = 2 to h do
        apply barycenter heuristic
            to level i of S and T
        collapse level i − 1
    for i = h downto 2 do
        apply barycenter heuristic
            to level i of S and T
        expand level i − 1
```

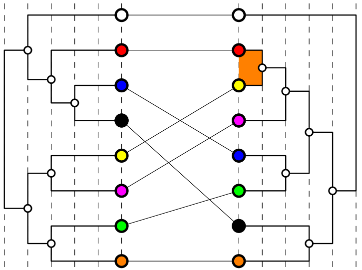
Hierarchy Sort [Holten, van Wijk '08]



`hierarchy-sort`(S, T)

augment trees to equal height h
while *layout improves* **do**
  for $i = 2$ **to** h **do**
    apply barycenter heuristic to level i of S and T
    collapse level $i - 1$
  for $i = h$ **downto** 2 **do**
    apply barycenter heuristic to level i of S and T
    expand level $i - 1$

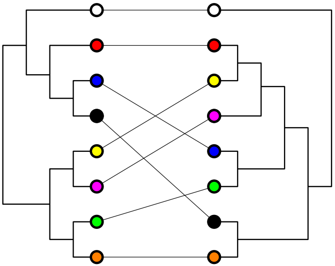
Hierarchy Sort [Holten, van Wijk '08]



`hierarchy-sort`(S, T)

augment trees to equal height h
while *layout improves* **do**
- **for** $i = 2$ **to** h **do**
  - apply barycenter heuristic to level i of S and T
  - collapse level $i - 1$
- **for** $i = h$ **downto** 2 **do**
  - apply barycenter heuristic to level i of S and T
  - expand level $i - 1$

Hierarchy Sort [Holten, van Wijk '08]



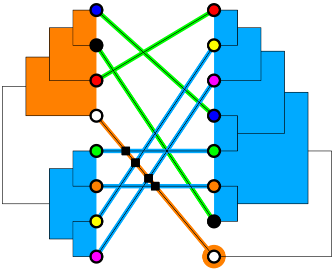
`hierarchy-sort`(S, T)

augment trees to equal height h
while *layout improves* **do**
- **for** $i = 2$ **to** h **do**
  - apply barycenter heuristic to level i of S and T
  - collapse level $i - 1$
- **for** $i = h$ **downto** 2 **do**
  - apply barycenter heuristic to level i of S and T
  - expand level $i - 1$

Hierarchy Sort [Holten, van Wijk '08]

- no quality guarantee
- implemented as hier-sort, running in $O(n \cdot h)$ time:
  - barycenter heuristic at most four times per level
  - outer loop at most twice
- improvement using edge weights: hier-sort++
- barycentric method *not* restricted to binary trees

Recursive Splitting [Buchin et al. '08]



■ induced crossing

$\texttt{RecSplit}(S = (S_1, S_2),\ T = (T_1, T_2))$

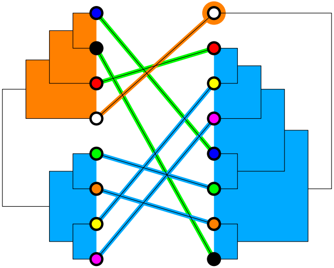
$\mathrm{cr}_{ST} \leftarrow \infty$

foreach $(\alpha, \beta) \in \{0, 1\}^2$ **do**

- $\mathrm{cr}_0 \leftarrow$ crossings induced by (α, β)
- $\mathrm{cr}_1 \leftarrow \texttt{RecSplit}(S_{1+\alpha}, T_{1+\beta})$
- $\mathrm{cr}_2 \leftarrow \texttt{RecSplit}(S_{2-\alpha}, T_{2-\beta})$
- $\mathrm{cr} \leftarrow \mathrm{cr}_0 + \mathrm{cr}_1 + \mathrm{cr}_2$
- **if** $cr < cr_{ST}$ **then**
  - $\mathrm{cr}_{ST} \leftarrow \mathrm{cr}$

return cr_{ST}

Recursive Splitting [Buchin et al. '08]



- induced crossing

$\texttt{RecSplit}(S = (S_1, S_2),\ T = (T_1, T_2))$

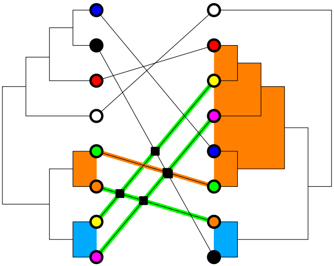
$\mathrm{cr}_{ST} \leftarrow \infty$

foreach $(\alpha, \beta) \in \{0, 1\}^2$ **do**

- $\mathrm{cr}_0 \leftarrow$ crossings induced by (α, β)
- $\mathrm{cr}_1 \leftarrow \texttt{RecSplit}(S_{1+\alpha}, T_{1+\beta})$
- $\mathrm{cr}_2 \leftarrow \texttt{RecSplit}(S_{2-\alpha}, T_{2-\beta})$
- $\mathrm{cr} \leftarrow \mathrm{cr}_0 + \mathrm{cr}_1 + \mathrm{cr}_2$
- **if** $cr < cr_{ST}$ **then**
  - $\mathrm{cr}_{ST} \leftarrow \mathrm{cr}$

return cr_{ST}

Recursive Splitting [Buchin et al. '08]



■ induced crossing

```
RecSplit(S = (S1, S2), T = (T1, T2))
```

$\texttt{RecSplit}(S = (S_1, S_2),\ T = (T_1, T_2))$

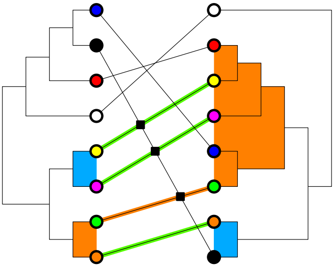
$\mathrm{cr}_{ST} \leftarrow \infty$

foreach $(\alpha, \beta) \in \{0, 1\}^2$ **do**

- $\mathrm{cr}_0 \leftarrow$ crossings induced by (α, β)
- $\mathrm{cr}_1 \leftarrow \texttt{RecSplit}(S_{1+\alpha}, T_{1+\beta})$
- $\mathrm{cr}_2 \leftarrow \texttt{RecSplit}(S_{2-\alpha}, T_{2-\beta})$
- $\mathrm{cr} \leftarrow \mathrm{cr}_0 + \mathrm{cr}_1 + \mathrm{cr}_2$
- **if** $cr < cr_{ST}$ **then**
  - $\mathrm{cr}_{ST} \leftarrow \mathrm{cr}$

return cr_{ST}

Recursive Splitting [Buchin et al. '08]



■ induced crossing

$\texttt{RecSplit}(S = (S_1, S_2), T = (T_1, T_2))$

$\mathrm{cr}_{ST} \leftarrow \infty$

foreach $(\alpha, \beta) \in \{0, 1\}^2$ **do**

$\quad \mathrm{cr}_0 \leftarrow$ crossings induced by (α, β)

$\quad \mathrm{cr}_1 \leftarrow \texttt{RecSplit}(S_{1+\alpha}, T_{1+\beta})$

$\quad \mathrm{cr}_2 \leftarrow \texttt{RecSplit}(S_{2-\alpha}, T_{2-\beta})$

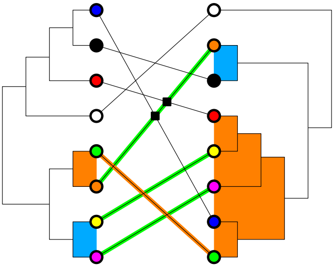
$\quad \mathrm{cr} \leftarrow \mathrm{cr}_0 + \mathrm{cr}_1 + \mathrm{cr}_2$

$\quad$ **if** $cr < cr_{ST}$ **then**

$\quad\quad \mathrm{cr}_{ST} \leftarrow \mathrm{cr}$

return cr_{ST}

Recursive Splitting [Buchin et al. '08]



▪ induced crossing

$\texttt{RecSplit}(S = (S_1, S_2),\ T = (T_1, T_2))$

$\mathrm{cr}_{ST} \leftarrow \infty$

foreach $(\alpha, \beta) \in \{0, 1\}^2$ **do**

$\quad \mathrm{cr}_0 \leftarrow$ crossings induced by (α, β)

$\quad \mathrm{cr}_1 \leftarrow \texttt{RecSplit}(S_{1+\alpha}, T_{1+\beta})$

$\quad \mathrm{cr}_2 \leftarrow \texttt{RecSplit}(S_{2-\alpha}, T_{2-\beta})$

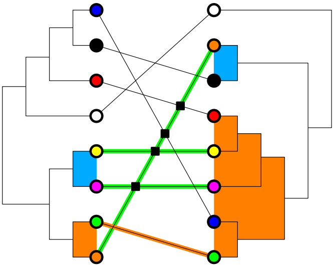
$\quad \mathrm{cr} \leftarrow \mathrm{cr}_0 + \mathrm{cr}_1 + \mathrm{cr}_2$

$\quad$ **if** $cr < cr_{ST}$ **then**

$\quad\quad \mathrm{cr}_{ST} \leftarrow \mathrm{cr}$

return cr_{ST}

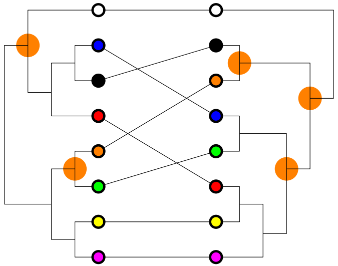
Recursive Splitting [Buchin et al. '08]



■ induced crossing

```
RecSplit(S = (S1, S2), T = (T1, T2))
crST ← ∞
foreach (α, β) ∈ {0,1}^2 do
    cr0 ← crossings induced by (α, β)
    cr1 ← RecSplit(S_{1+α}, T_{1+β})
    cr2 ← RecSplit(S_{2−α}, T_{2−β})
    cr ← cr0+cr1+cr2
    if cr < crST then
        crST ← cr
return crST
```

Recursive Splitting [Buchin et al. '08]



- induced crossing

$\texttt{RecSplit}(S = (S_1, S_2), T = (T_1, T_2))$

$\mathrm{cr}_{ST} \leftarrow \infty$

foreach $(\alpha, \beta) \in \{0, 1\}^2$ **do**

  $\mathrm{cr}_0 \leftarrow$ crossings induced by (α, β)

  $\mathrm{cr}_1 \leftarrow \texttt{RecSplit}(S_{1+\alpha}, T_{1+\beta})$

  $\mathrm{cr}_2 \leftarrow \texttt{RecSplit}(S_{2-\alpha}, T_{2-\beta})$

  $\mathrm{cr} \leftarrow \mathrm{cr}_0 + \mathrm{cr}_1 + \mathrm{cr}_2$

  if $cr < cr_{ST}$ **then**

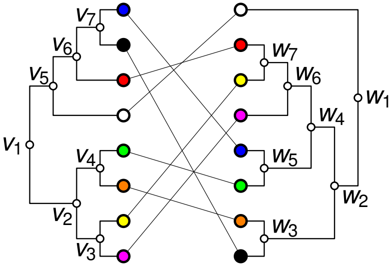
    $\mathrm{cr}_{ST} \leftarrow \mathrm{cr}$

return cr_{ST}

Recursive Splitting [Buchin et al. '08]

- 2-approximation for *complete* binary trees [$O(n^3)$ time]
- heuristic for general binary trees [$O(n \cdot 4^h)$ time] [$h =$ tree height]
- implemented as rec-split++
  - additional heuristic improvement for unbalanced trees
  - branch-and-bound for pruning the search tree

Exact Branch-and-Bound Algorithm

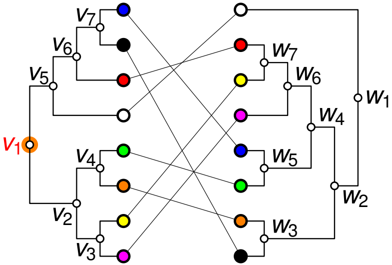


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	7
v_2	–	2/0	–	2/0	–	–	–	2
v_3	–	–	–	–	–	0/1	–	1
v_4	–	0/1	–	–	–	–	–	1
v_5	3/0	–	–	–	–	–	–	1
v_6	–	1/0	–	1/0	–	–	–	2
v_7	–	0/1	–	–	–	–	–	1
ic	2	5	1	3	1	2	1	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
  $u \leftarrow$ node maximizing cross. diff.
  cr$_1 \leftarrow$ lower bd. if swapping u
  cr$_2 \leftarrow$ lower bd. if keeping u
  if $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
    prune search tree branch
  if *all nodes fixed* **then**
    update cr { *new solution!* }
  else
    update ic-values

Exact Branch-and-Bound Algorithm



	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	7
v_2	–	2/0	–	2/0	–	–	–	2
v_3	–	–	–	–	–	0/1	–	1
v_4	–	0/1	–	–	–	–	–	1
v_5	3/0	–	–	–	–	–	–	1
v_6	–	1/0	–	1/0	–	–	–	2
v_7	–	0/1	–	–	–	–	–	1
ic	2	5	1	3	1	2	1	

`branch-and-bound`(S, T)

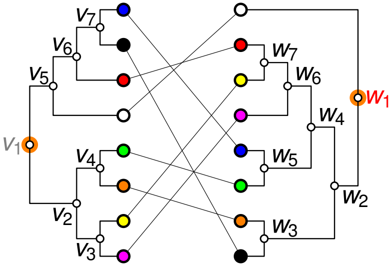
precompute crossing table

fix node $u^\star$ maximizing ic$(u^\star)$

cr $\leftarrow \infty$

while *search tree not traversed* **do**
- $u \leftarrow$ node maximizing cross. diff.
- $\text{cr}_1 \leftarrow$ lower bd. if swapping u
- $\text{cr}_2 \leftarrow$ lower bd. if keeping u
- **if** $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
  - prune search tree branch
- **if** *all nodes fixed* **then**
  - update cr { *new solution!* }
- **else**
  - update ic-values

Exact Branch-and-Bound Algorithm

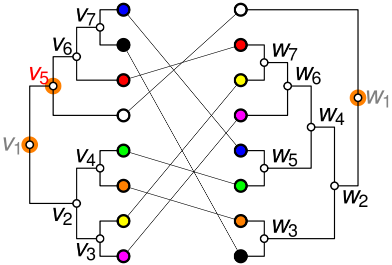


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	7
v_2	–	2/0	–	2/0	–	–	–	2
v_3	–	–	–	–	–	0/1	–	1
v_4	–	0/1	–	–	–	–	–	1
v_5	3/0	–	–	–	–	–	–	1
v_6	–	1/0	–	1/0	–	–	–	2
v_7	–	0/1	–	–	–	–	–	1
ic	1	4	0	2	0	1	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
- $u \leftarrow$ node maximizing cross. diff.
- $cr_1 \leftarrow$ lower bd. if swapping u
- $cr_2 \leftarrow$ lower bd. if keeping u
- **if** $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
  - prune search tree branch
- **if** *all nodes fixed* **then**
  - update cr { *new solution!* }
- **else**
  - update ic-values

Exact Branch-and-Bound Algorithm

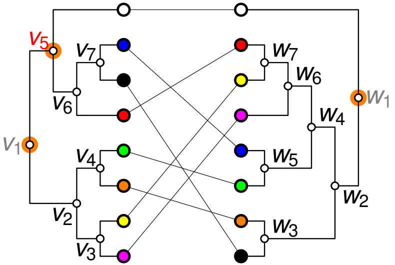


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	6
v_2	–	2/0	–	2/0	–	–	–	2
v_3	–	–	–	–	–	0/1	–	1
v_4	–	0/1	–	–	–	–	–	1
v_5	3/0	–	–	–	–	–	–	0
v_6	–	1/0	–	1/0	–	–	–	2
v_7	–	0/1	–	–	–	–	–	1
ic	1	4	0	2	0	1	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
  $u \leftarrow$ node maximizing cross. diff.
  $\text{cr}_1 \leftarrow$ lower bd. if swapping u
  $\text{cr}_2 \leftarrow$ lower bd. if keeping u
  if $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
    prune search tree branch
  if *all nodes fixed* **then**
    update cr { *new solution!* }
  else
    update ic-values

Exact Branch-and-Bound Algorithm

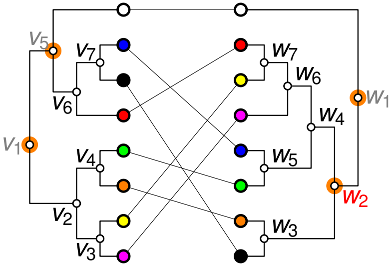


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	6
v_2	–	2/0	–	2/0	–	–	–	2
v_3	–	–	–	–	–	0/1	–	1
v_4	–	0/1	–	–	–	–	–	1
v_5	3/0	–	–	–	–	–	–	0
v_6	–	1/0	–	1/0	–	–	–	2
v_7	–	0/1	–	–	–	–	–	1
ic	1	4	0	2	0	1	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
- $u \leftarrow$ node maximizing cross. diff.
- $\mathrm{cr}_1 \leftarrow$ lower bd. if swapping u
- $\mathrm{cr}_2 \leftarrow$ lower bd. if keeping u
- **if** $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
  - prune search tree branch
- **if** *all nodes fixed* **then**
  - update cr { *new solution!* }
- **else**
  - update ic-values

Exact Branch-and-Bound Algorithm

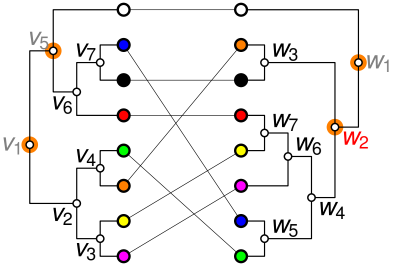


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	6
v_2	–	2/0	–	2/0	–	–	–	2
v_3	–	–	–	–	–	0/1	–	1
v_4	–	0/1	–	–	–	–	–	1
v_5	3/0	–	–	–	–	–	–	0
v_6	–	1/0	–	1/0	–	–	–	2
v_7	–	0/1	–	–	–	–	–	1
ic	0	4	0	2	0	1	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
- $u \leftarrow$ node maximizing cross. diff.
- $cr_1 \leftarrow$ lower bd. if swapping u
- $cr_2 \leftarrow$ lower bd. if keeping u
- **if** $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
  - prune search tree branch
- **if** *all nodes fixed* **then**
  - update cr { *new solution!* }
- **else**
  - update ic-values

Exact Branch-and-Bound Algorithm

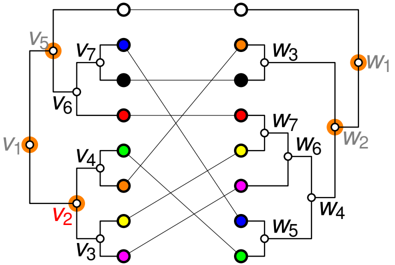


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	6
v_2	–	2/0	–	2/0	–	–	–	2
v_3	–	–	–	–	–	0/1	–	1
v_4	–	0/1	–	–	–	–	–	1
v_5	3/0	–	–	–	–	–	–	0
v_6	–	1/0	–	1/0	–	–	–	2
v_7	–	0/1	–	–	–	–	–	1
ic	0	4	0	2	0	1	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
- $u \leftarrow$ node maximizing cross. diff.
- $\mathrm{cr}_1 \leftarrow$ lower bd. if swapping u
- $\mathrm{cr}_2 \leftarrow$ lower bd. if keeping u
- **if** $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
  - prune search tree branch
- **if** *all nodes fixed* **then**
  - update cr { *new solution!* }
- **else**
  - update ic-values

Exact Branch-and-Bound Algorithm

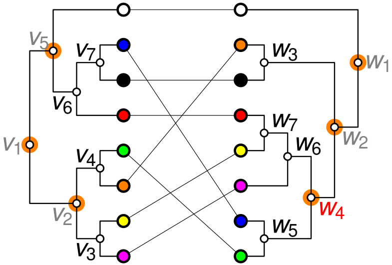


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	5
v_2	—	2/0	—	2/0	—	—	—	1
v_3	—	—	—	—	—	0/1	—	1
v_4	—	0/1	—	—	—	—	—	0
v_5	3/0	—	—	—	—	—	—	0
v_6	—	1/0	—	1/0	—	—	—	1
v_7	—	0/1	—	—	—	—	—	0
ic	0	4	0	2	0	1	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
- $u \leftarrow$ node maximizing cross. diff.
- $\text{cr}_1 \leftarrow$ lower bd. if swapping u
- $\text{cr}_2 \leftarrow$ lower bd. if keeping u
- **if** $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
  - prune search tree branch
- **if** *all nodes fixed* **then**
  - update cr { *new solution!* }
- **else**
  - update ic-values

Exact Branch-and-Bound Algorithm

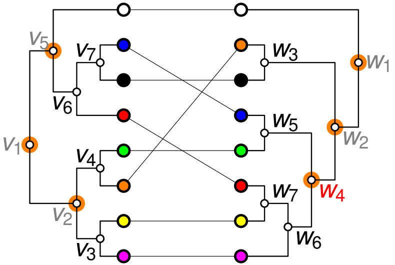


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	5
v_2	–	2/0	–	2/0	–	–	–	1
v_3	–	–	–	–	–	0/1	–	1
v_4	–	0/1	–	–	–	–	–	0
v_5	3/0	–	–	–	–	–	–	0
v_6	–	1/0	–	1/0	–	–	–	1
v_7	–	0/1	–	–	–	–	–	0
ic	0	3	0	1	0	1	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
- $u \leftarrow$ node maximizing cross. diff.
- $\mathrm{cr}_1 \leftarrow$ lower bd. if swapping u
- $\mathrm{cr}_2 \leftarrow$ lower bd. if keeping u
- **if** $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
  - prune search tree branch
- **if** *all nodes fixed* **then**
  - update cr { *new solution!* }
- **else**
  - update ic-values

Exact Branch-and-Bound Algorithm

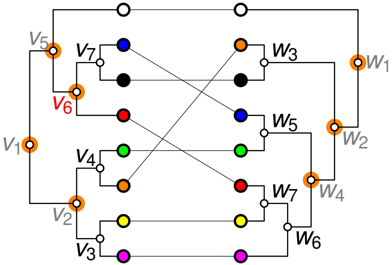


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	5
v_2	–	2/0	–	2/0	–	–	–	1
v_3	–	–	–	–	–	0/1	–	1
v_4	–	0/1	–	–	–	–	–	0
v_5	3/0	–	–	–	–	–	–	0
v_6	–	1/0	–	1/0	–	–	–	1
v_7	–	0/1	–	–	–	–	–	0
ic	0	3	0	1	0	1	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
- $u \leftarrow$ node maximizing cross. diff.
- cr$_1 \leftarrow$ lower bd. if swapping u
- cr$_2 \leftarrow$ lower bd. if keeping u
- **if** $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
  - prune search tree branch
- **if** *all nodes fixed* **then**
  - update cr { *new solution!* }
- **else**
  - update ic-values

Exact Branch-and-Bound Algorithm

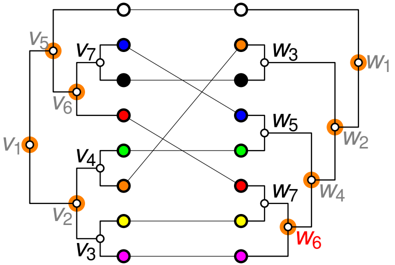


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	4
v_2	–	2/0	–	2/0	–	–	–	0
v_3	–	–	–	–	–	0/1	–	1
v_4	–	0/1	–	–	–	–	–	0
v_5	3/0	–	–	–	–	–	–	0
v_6	–	1/0	–	1/0	–	–	–	0
v_7	–	0/1	–	–	–	–	–	0
ic	0	3	0	1	0	1	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
- $u \leftarrow$ node maximizing cross. diff.
- $cr_1 \leftarrow$ lower bd. if swapping u
- $cr_2 \leftarrow$ lower bd. if keeping u
- **if** $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
  - prune search tree branch
- **if** *all nodes fixed* **then**
  - update cr { *new solution!* }
- **else**
  - update ic-values

Exact Branch-and-Bound Algorithm

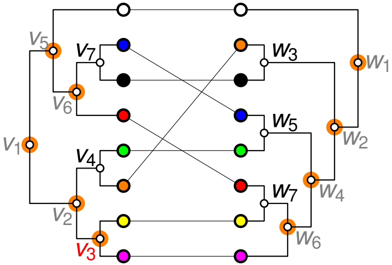


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	4
v_2	–	2/0	–	2/0	–	–	–	0
v_3	–	–	–	–	–	0/1	–	1
v_4	–	0/1	–	–	–	–	–	0
v_5	3/0	–	–	–	–	–	–	0
v_6	–	1/0	–	1/0	–	–	–	0
v_7	–	0/1	–	–	–	–	–	0
ic	0	2	0	0	0	1	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
- $u \leftarrow$ node maximizing cross. diff.
- $cr_1 \leftarrow$ lower bd. if swapping u
- $cr_2 \leftarrow$ lower bd. if keeping u
- **if** $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
  - prune search tree branch
- **if** *all nodes fixed* **then**
  - update cr { *new solution!* }
- **else**
  - update ic-values

Exact Branch-and-Bound Algorithm

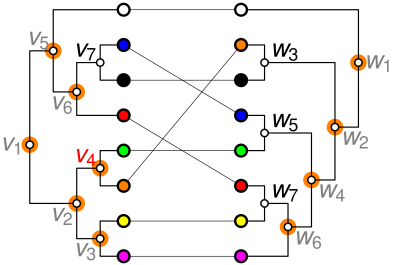


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	3
v_2	–	2/0	–	2/0	–	–	–	0
v_3	–	–	–	–	–	0/1	–	0
v_4	–	0/1	–	–	–	–	–	0
v_5	3/0	–	–	–	–	–	–	0
v_6	–	1/0	–	1/0	–	–	–	0
v_7	–	0/1	–	–	–	–	–	0
ic	0	2	0	0	0	1	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
  $u \leftarrow$ node maximizing cross. diff.
  $\text{cr}_1 \leftarrow$ lower bd. if swapping u
  $\text{cr}_2 \leftarrow$ lower bd. if keeping u
  if $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
    prune search tree branch
  if *all nodes fixed* **then**
    update cr { *new solution!* }
  else
    update ic-values

Exact Branch-and-Bound Algorithm

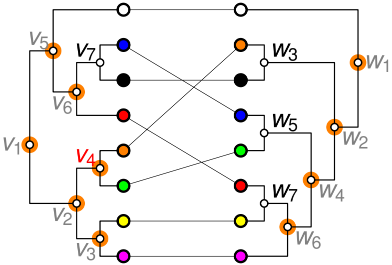


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	3
v_2	–	2/0	–	2/0	–	–	–	0
v_3	–	–	–	–	–	0/1	–	0
v_4	–	0/1	–	–	–	–	–	0
v_5	3/0	–	–	–	–	–	–	0
v_6	–	1/0	–	1/0	–	–	–	0
v_7	–	0/1	–	–	–	–	–	0
ic	0	2	0	0	0	0	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
- $u \leftarrow$ node maximizing cross. diff.
- $cr_1 \leftarrow$ lower bd. if swapping u
- $cr_2 \leftarrow$ lower bd. if keeping u
- **if** $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
  - prune search tree branch
- **if** *all nodes fixed* **then**
  - update cr { *new solution!* }
- **else**
  - update ic-values

Exact Branch-and-Bound Algorithm

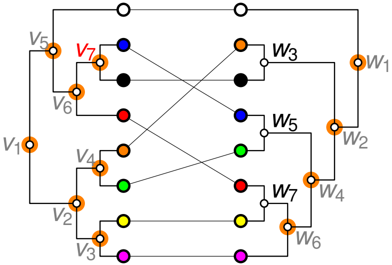


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	3
v_2	–	2/0	–	2/0	–	–	–	0
v_3	–	–	–	–	–	0/1	–	0
v_4	–	0/1	–	–	–	–	–	0
v_5	3/0	–	–	–	–	–	–	0
v_6	–	1/0	–	1/0	–	–	–	0
v_7	–	0/1	–	–	–	–	–	0
ic	0	2	0	0	0	0	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
- $u \leftarrow$ node maximizing cross. diff.
- $cr_1 \leftarrow$ lower bd. if swapping u
- $cr_2 \leftarrow$ lower bd. if keeping u
- **if** $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
  - prune search tree branch
- **if** *all nodes fixed* **then**
  - update cr { *new solution!* }
- **else**
  - update ic-values

Exact Branch-and-Bound Algorithm

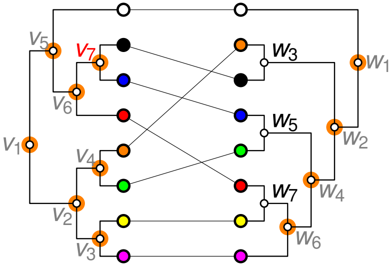


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	3
v_2	–	2/0	–	2/0	–	–	–	0
v_3	–	–	–	–	–	0/1	–	0
v_4	–	0/1	–	–	–	–	–	0
v_5	3/0	–	–	–	–	–	–	0
v_6	–	1/0	–	1/0	–	–	–	0
v_7	–	0/1	–	–	–	–	–	0
ic	0	1	0	0	0	0	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
  $u \leftarrow$ node maximizing cross. diff.
  $\text{cr}_1 \leftarrow$ lower bd. if swapping u
  $\text{cr}_2 \leftarrow$ lower bd. if keeping u
  if $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
    prune search tree branch
  if *all nodes fixed* **then**
    update cr { *new solution!* }
  else
    update ic-values

Exact Branch-and-Bound Algorithm

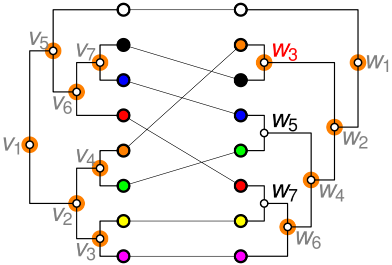


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	3
v_2	–	2/0	–	2/0	–	–	–	0
v_3	–	–	–	–	–	0/1	–	0
v_4	–	0/1	–	–	–	–	–	0
v_5	3/0	–	–	–	–	–	–	0
v_6	–	1/0	–	1/0	–	–	–	0
v_7	–	0/1	–	–	–	–	–	0
ic	0	1	0	0	0	0	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
- $u \leftarrow$ node maximizing cross. diff.
- $\text{cr}_1 \leftarrow$ lower bd. if swapping u
- $\text{cr}_2 \leftarrow$ lower bd. if keeping u
- **if** $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
  - prune search tree branch
- **if** *all nodes fixed* **then**
  - update cr { *new solution!* }
- **else**
  - update ic-values

Exact Branch-and-Bound Algorithm

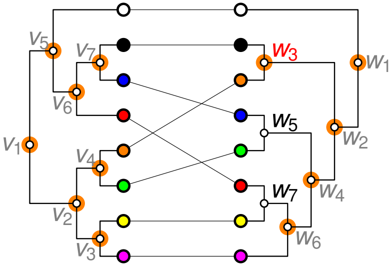


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	3
v_2	–	2/0	–	2/0	–	–	–	0
v_3	–	–	–	–	–	0/1	–	0
v_4	–	0/1	–	–	–	–	–	0
v_5	3/0	–	–	–	–	–	–	0
v_6	–	1/0	–	1/0	–	–	–	0
v_7	–	0/1	–	–	–	–	–	0
ic	0	0	0	0	0	0	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
- $u \leftarrow$ node maximizing cross. diff.
- $\text{cr}_1 \leftarrow$ lower bd. if swapping u
- $\text{cr}_2 \leftarrow$ lower bd. if keeping u
- **if** $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
  - prune search tree branch
- **if** *all nodes fixed* **then**
  - update cr { *new solution!* }
- **else**
  - update ic-values

Exact Branch-and-Bound Algorithm

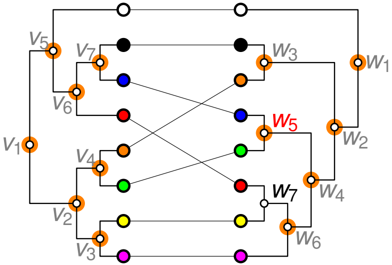


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	3
v_2	–	2/0	–	2/0	–	–	–	0
v_3	–	–	–	–	–	0/1	–	0
v_4	–	0/1	–	–	–	–	–	0
v_5	3/0	–	–	–	–	–	–	0
v_6	–	1/0	–	1/0	–	–	–	0
v_7	–	0/1	–	–	–	–	–	0
ic	0	0	0	0	0	0	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
  $u \leftarrow$ node maximizing cross. diff.
  $cr_1 \leftarrow$ lower bd. if swapping u
  $cr_2 \leftarrow$ lower bd. if keeping u
  if $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
    prune search tree branch
  if *all nodes fixed* **then**
    update cr { *new solution!* }
  else
    update ic-values

Exact Branch-and-Bound Algorithm

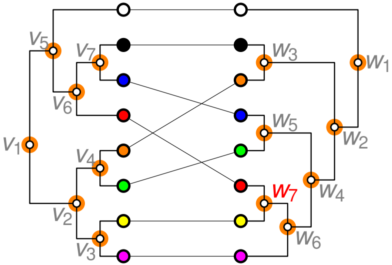


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	2
v_2	–	2/0	–	2/0	–	–	–	0
v_3	–	–	–	–	–	0/1	–	0
v_4	–	0/1	–	–	–	–	–	0
v_5	3/0	–	–	–	–	–	–	0
v_6	–	1/0	–	1/0	–	–	–	0
v_7	–	0/1	–	–	–	–	–	0
ic	0	0	0	0	0	0	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing $\text{ic}(u^\star)$
$\text{cr} \leftarrow \infty$
while *search tree not traversed* **do**
- $u \leftarrow$ node maximizing cross. diff.
- $\text{cr}_1 \leftarrow$ lower bd. if swapping u
- $\text{cr}_2 \leftarrow$ lower bd. if keeping u
- **if** $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
  - prune search tree branch
- **if** *all nodes fixed* **then**
  - update cr { *new solution!* }
- **else**
  - update ic-values

Exact Branch-and-Bound Algorithm

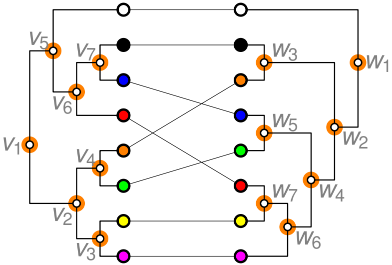


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	1
v_2	–	2/0	–	2/0	–	–	–	0
v_3	–	–	–	–	–	0/1	–	0
v_4	–	0/1	–	–	–	–	–	0
v_5	3/0	–	–	–	–	–	–	0
v_6	–	1/0	–	1/0	–	–	–	0
v_7	–	0/1	–	–	–	–	–	0
ic	0	0	0	0	0	0	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
  $u \leftarrow$ node maximizing cross. diff.
  $\text{cr}_1 \leftarrow$ lower bd. if swapping u
  $\text{cr}_2 \leftarrow$ lower bd. if keeping u
  if $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
    prune search tree branch
  if *all nodes fixed* **then**
    update cr { *new solution!* }
  else
    update ic-values

Exact Branch-and-Bound Algorithm

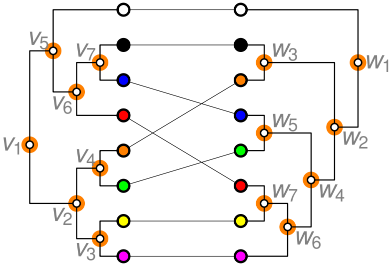


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	0
v_2	—	2/0	—	2/0	—	—	—	0
v_3	—	—	—	—	—	0/1	—	0
v_4	—	0/1	—	—	—	—	—	0
v_5	3/0	—	—	—	—	—	—	0
v_6	—	1/0	—	1/0	—	—	—	0
v_7	—	0/1	—	—	—	—	—	0
ic	0	0	0	0	0	0	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
- $u \leftarrow$ node maximizing cross. diff.
- $cr_1 \leftarrow$ lower bd. if swapping u
- $cr_2 \leftarrow$ lower bd. if keeping u
- **if** $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
  - prune search tree branch
- **if** *all nodes fixed* **then**
  - update cr { *new solution!* }
- **else**
  - update ic-values

Exact Branch-and-Bound Algorithm

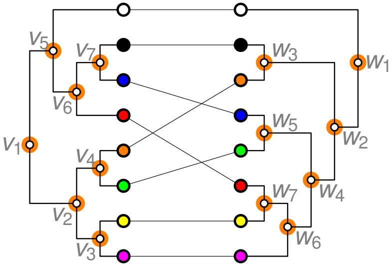


	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	0
v_2	–	2/0	–	2/0	–	–	–	0
v_3	–	–	–	–	–	0/1	–	0
v_4	–	0/1	–	–	–	–	–	0
v_5	3/0	–	–	–	–	–	–	0
v_6	–	1/0	–	1/0	–	–	–	0
v_7	–	0/1	–	–	–	–	–	0
ic	0	0	0	0	0	0	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
- $u \leftarrow$ node maximizing cross. diff.
- $\text{cr}_1 \leftarrow$ lower bd. if swapping u
- $\text{cr}_2 \leftarrow$ lower bd. if keeping u
- **if** $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
  - prune search tree branch
- **if** *all nodes fixed* **then**
  - update cr { *new solution!* }
- **else**
  - update ic-values

Exact Branch-and-Bound Algorithm



	w_1	w_2	w_3	w_4	w_5	w_6	w_7	ic
v_1	0/4	3/2	1/0	2/1	0/1	0/1	0/1	0
v_2	–	2/0	–	2/0	–	–	–	0
v_3	–	–	–	–	–	0/1	–	0
v_4	–	0/1	–	–	–	–	–	0
v_5	3/0	–	–	–	–	–	–	0
v_6	–	1/0	–	1/0	–	–	–	0
v_7	–	0/1	–	–	–	–	–	0
ic	0	0	0	0	0	0	0	

`branch-and-bound`(S, T)

precompute crossing table
fix node $u^\star$ maximizing ic$(u^\star)$
cr $\leftarrow \infty$
while *search tree not traversed* **do**
- $u \leftarrow$ node maximizing cross. diff.
- $\mathrm{cr}_1 \leftarrow$ lower bd. if swapping u
- $\mathrm{cr}_2 \leftarrow$ lower bd. if keeping u
- **if** $cr_1 \geq cr$ **or** $cr_2 \geq cr$ **then**
  - prune search tree branch
- **if** *all nodes fixed* **then**
  - update cr { *new solution!* }
- **else**
  - update ic-values

Exact Branch-and-Bound Algorithm

- precompute crossing table [$O(n^2)$ time]
- yields optimal solution [$O(n^2 + n \cdot 2^{2n})$ time]
- greedy heuristic: take first feasible solution [$O(n^2)$ time]

Experiments

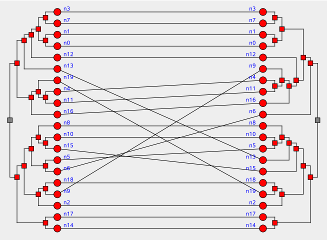
Experiments

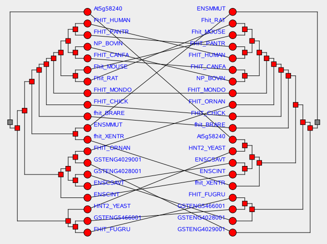
- implemented in Java:
  - rec-split++
  - iterated 1STL
  - hier-sort(++)
  - greedy
  - branch-and-bound
- and, using CPLEX:
  - simple ILP formulation
- goals of our study
  - evaluation of crossing reduction performance
    using ratio $(\mathrm{cr}_{\mathcal{A}} + 1)/(\mathrm{cr}_{\mathrm{opt}} + 1)$ for algorithm $\mathcal{A}$
  - running time analysis for real-world input sizes

Tanglegram Data

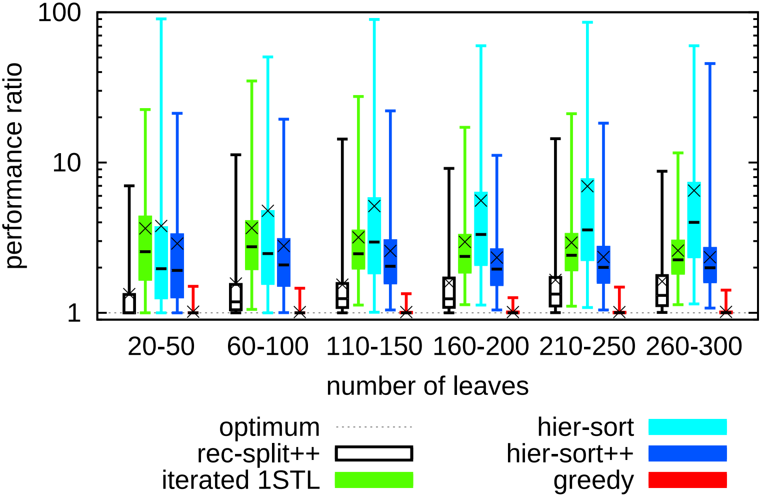
Among others...

- 100 mutated pairs of arbitrary binary trees for $n = 20, 30, \ldots, 300$
- 1303 real-world phylogenetic tree pairs from the TreeFam database
  - $n \in [15, 600]$ leaves
  - 75% have $n \leq 50$ leaves
  - 95% have $n \leq 100$ leaves
  - rather low crossing numbers

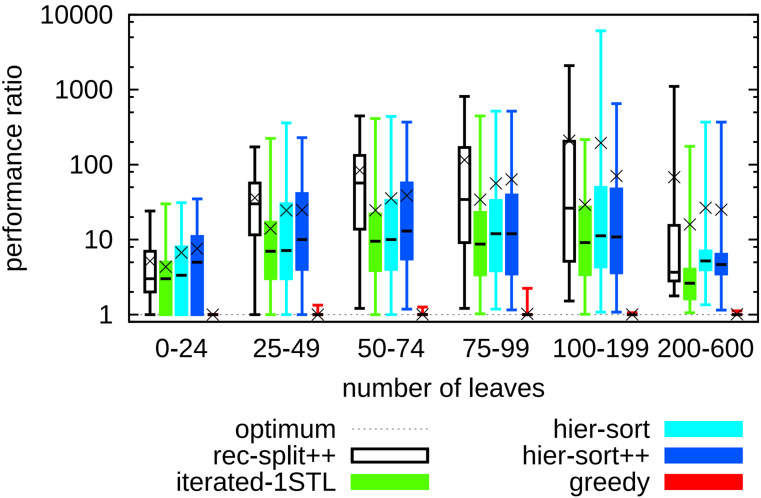




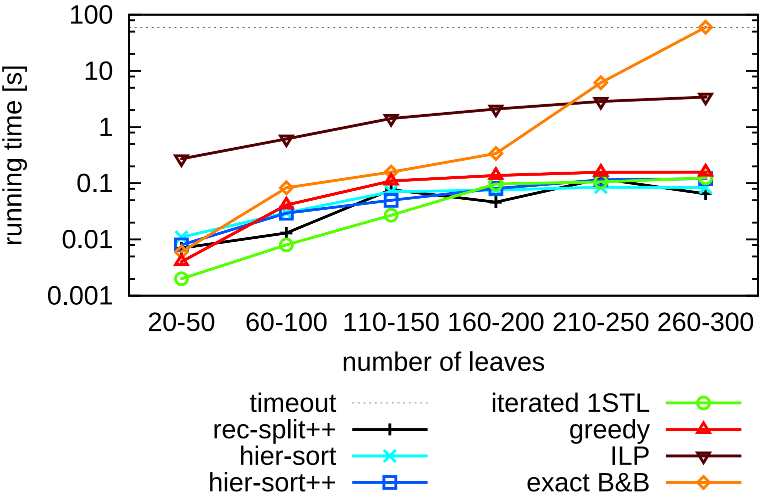
Performance Mutated Trees



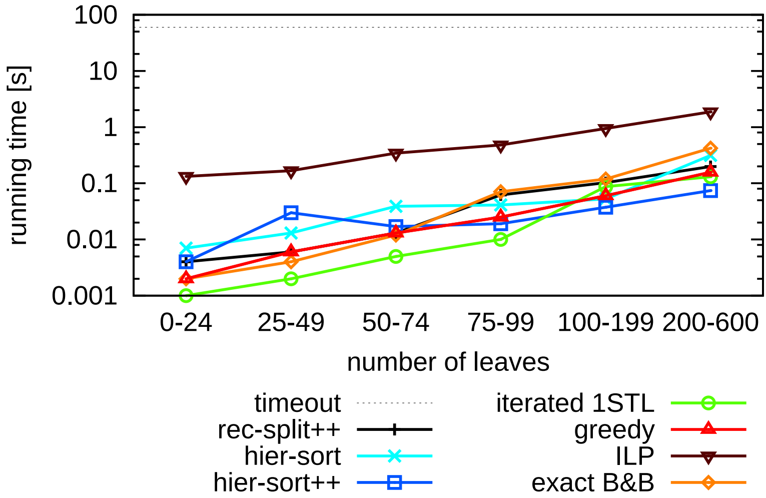
Performance Real-World Phylogenetic Trees



Running Time Mutated Trees



Running Time Real-World Phylogenetic Trees



Conclusions

- compared 3 existing algorithms and new greedy heuristic
- greedy heuristic: *the* method of choice for binary trees
  - found optimal solutions in 82% of our instances
  - performance never > 2.24
  - solved even the largest instances ($n \approx 600$) in ≤ 0.5 sec
- two new exact methods: branch-and-bound and ILP: often fast enough in practice

Open Problems

- where *is* greedy bad?
- non-binary trees
- inter-tree edges forming arbitrary bipartite graphs