Maximum Common Subgraph

Algorithms and Algorithm Portfolios

Paulius Dilkas

School of Computing Science
University of Glasgow

18th March 2018

Outline

Outline

Maximum Common Subgraph

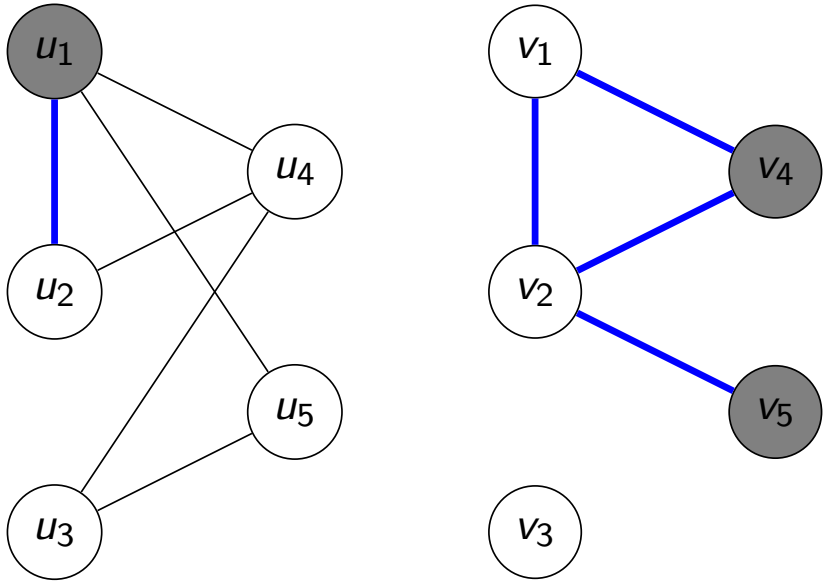
Definition

A *maximum common (induced) subgraph* between graphs G_1 and G_2 is a graph $G_3 = (V_3, E_3)$ such that G_3 is isomorphic to induced subgraphs of both G_1 and G_2 with $|V_3|$ maximised.

Maximum Common Subgraph

Definition

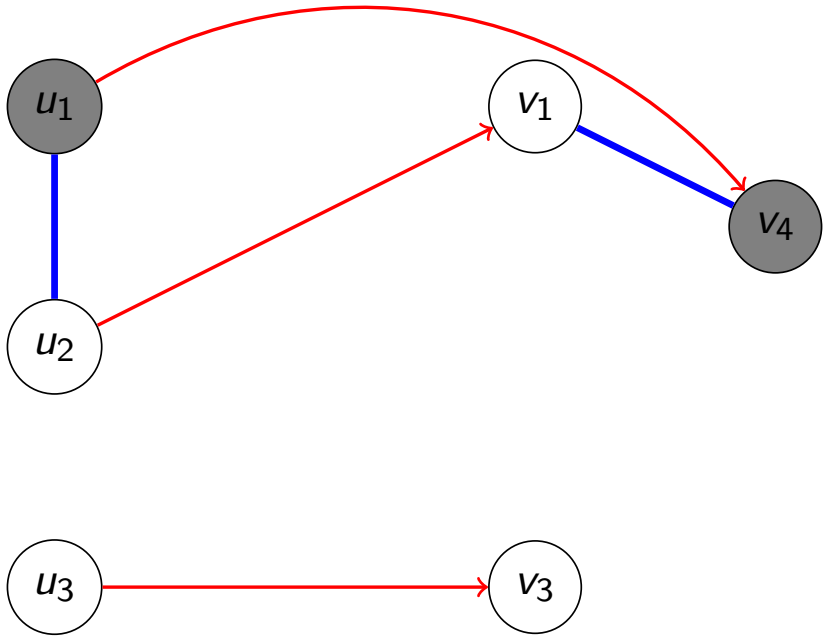
A *maximum common (induced) subgraph* between graphs G_1 and G_2 is a graph $G_3 = (V_3, E_3)$ such that G_3 is isomorphic to induced subgraphs of both G_1 and G_2 with $|V_3|$ maximised.



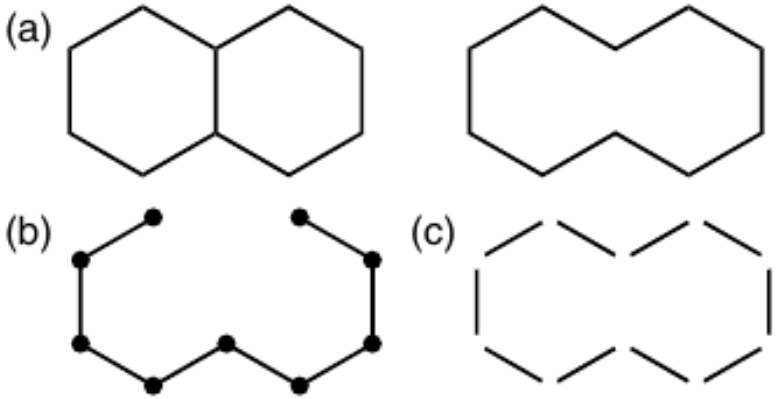
Maximum Common Subgraph

Definition

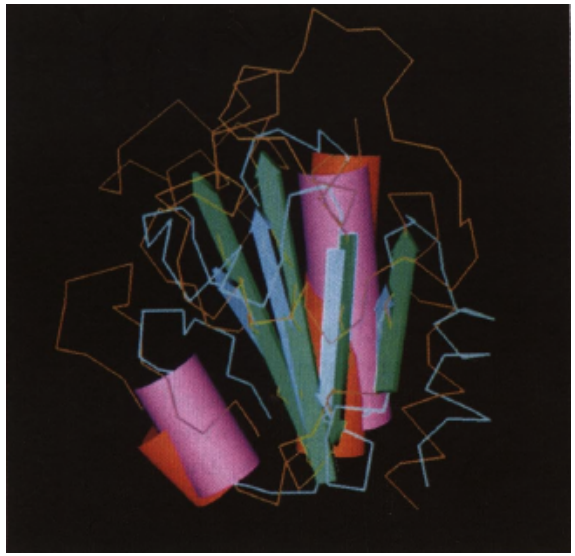
A *maximum common (induced) subgraph* between graphs G_1 and G_2 is a graph $G_3 = (V_3, E_3)$ such that G_3 is isomorphic to induced subgraphs of both G_1 and G_2 with $|V_3|$ maximised.



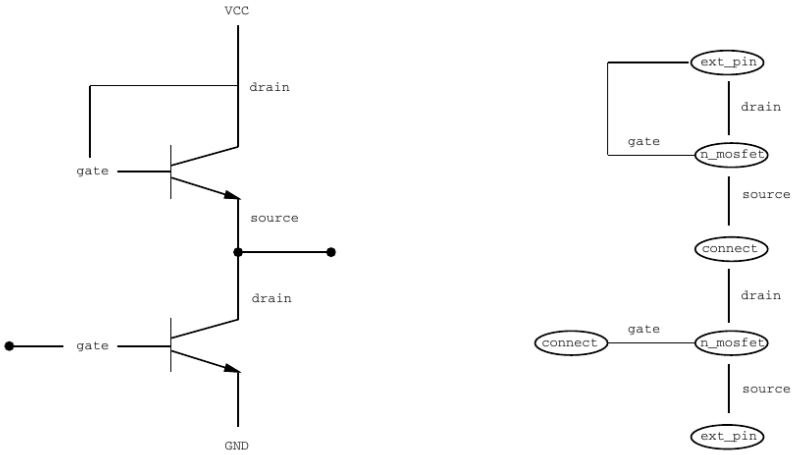
Why Is It Important?



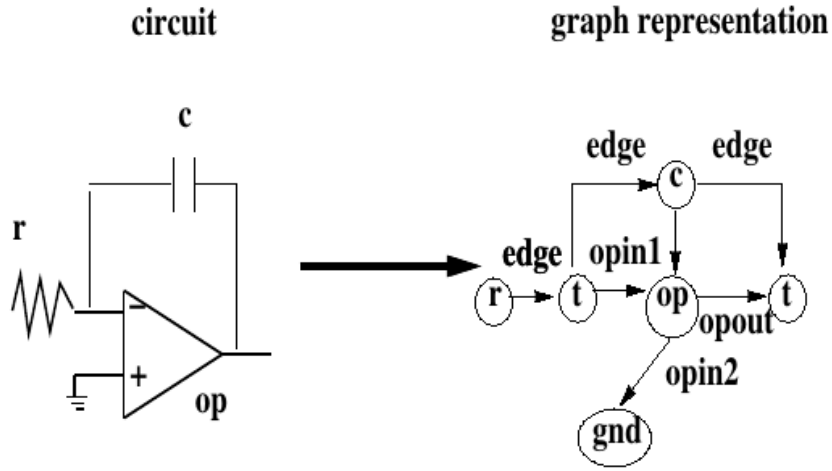
Source: Ehrlich and Rarey 2011



Source: M. Grindley et al. 1993



Source: Cook and Holder 1994



Source: Djoko, Cook and Holder 1997

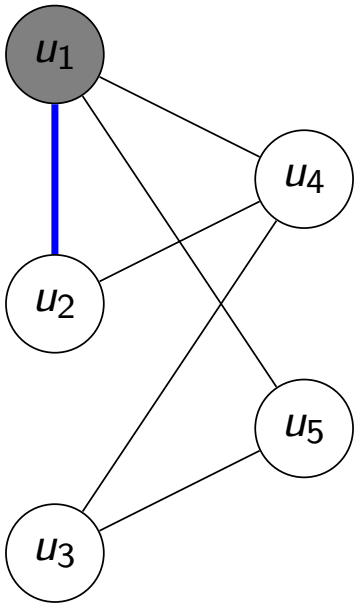
Outline

Algorithms

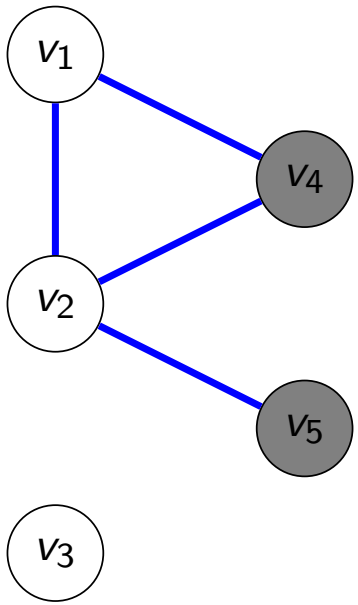
- McSplit, McSplit↓
  - McCreesh, Prosser and Trimble 2017
- clique encoding
  - McCreesh, Ndiaye et al. 2016
- $k\downarrow$
  - Hoffmann, McCreesh and Reilly 2017

McSplit: a Branch and Bound Algorithm

Partial solution:
Upper bound: 4



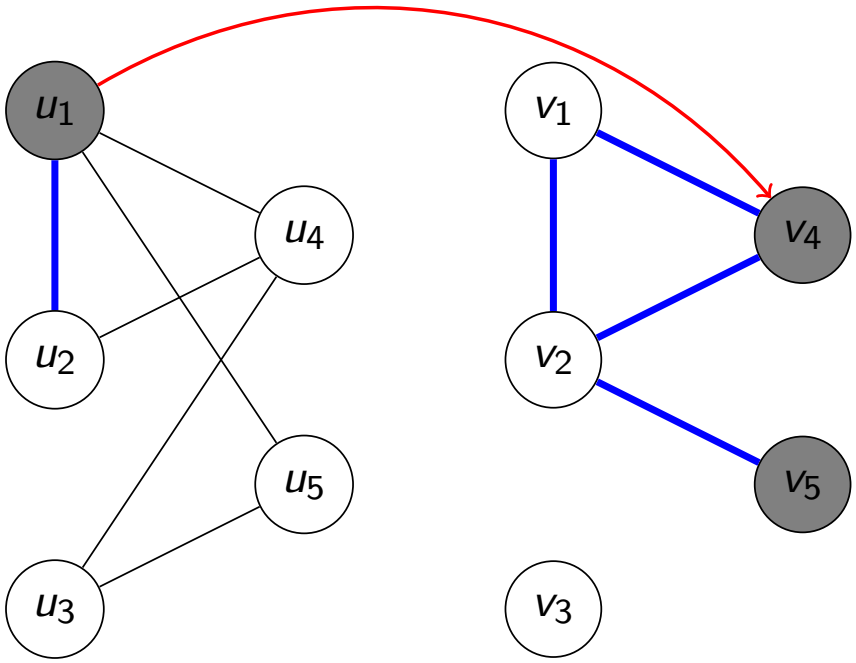
G_1



G_2

Label	G_1	G_2
0	u_2, u_3, u_4, u_5	v_1, v_2, v_3
1	u_1	v_4, v_5

McSplit: a Branch and Bound Algorithm

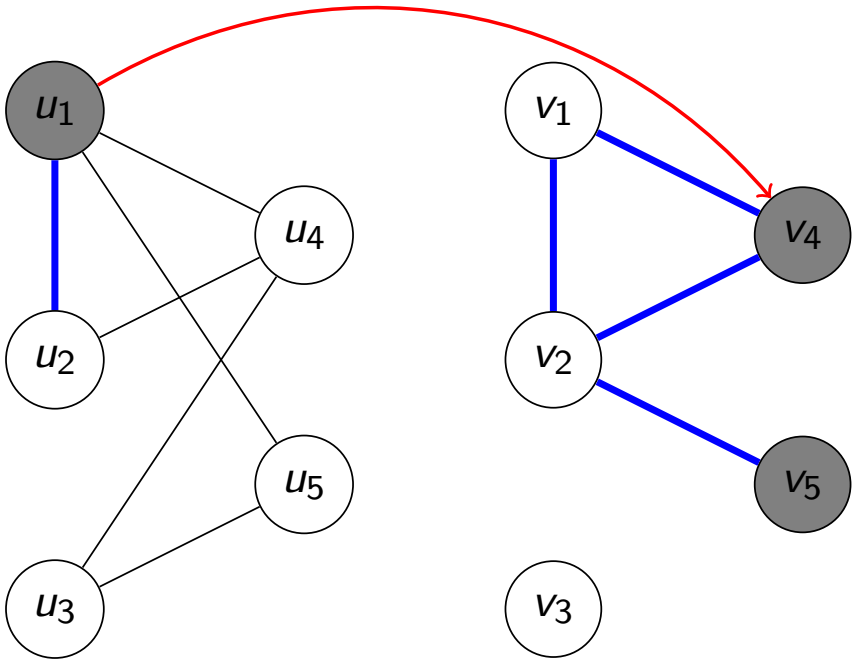


Partial solution:
Upper bound: 4

Label	G_1	G_2
0	u_2, u_3, u_4, u_5	v_1, v_2, v_3
1	u_1	v_4, v_5

Decision: $u_1 \mapsto v_4$

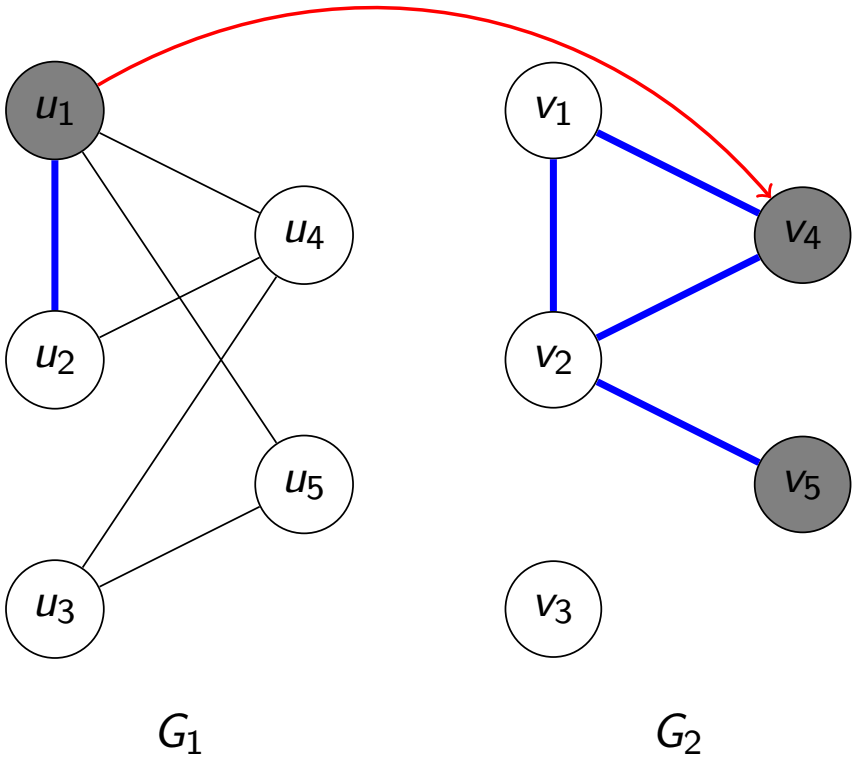
McSplit: a Branch and Bound Algorithm



Partial solution:
Upper bound: 4

Label	G_1	G_2
00	u_3	v_3
01	u_4, u_5	$\emptyset$
02	u_2	v_1, v_2
10	$\emptyset$	v_5

McSplit: a Branch and Bound Algorithm

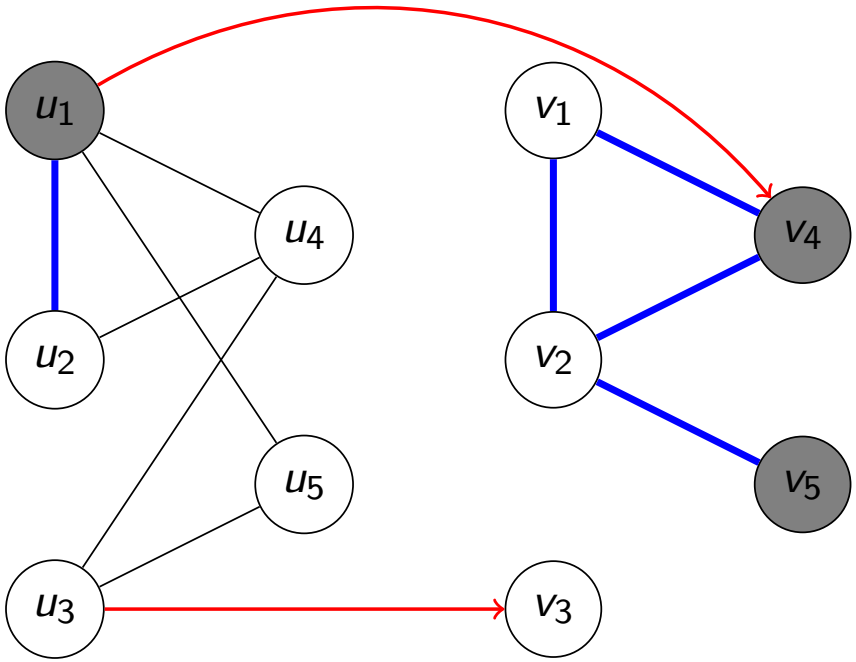


Partial solution: $u_1 \mapsto v_4$

Upper bound: $1 + 2$

Label	G_1	G_2
00	u_3	v_3
01	u_2	v_1, v_2

McSplit: a Branch and Bound Algorithm

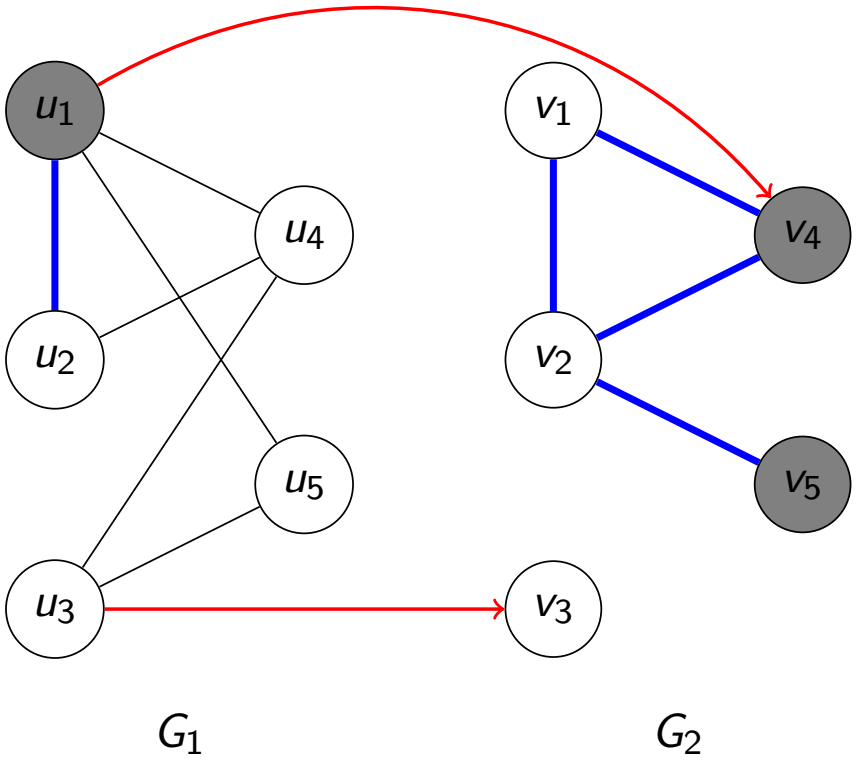


Partial solution: $u_1 \mapsto v_4$
Upper bound: $1 + 2$

Label	G_1	G_2
00	u_3	v_3
01	u_2	v_1, v_2

Decision: $u_3 \mapsto v_3$

McSplit: a Branch and Bound Algorithm

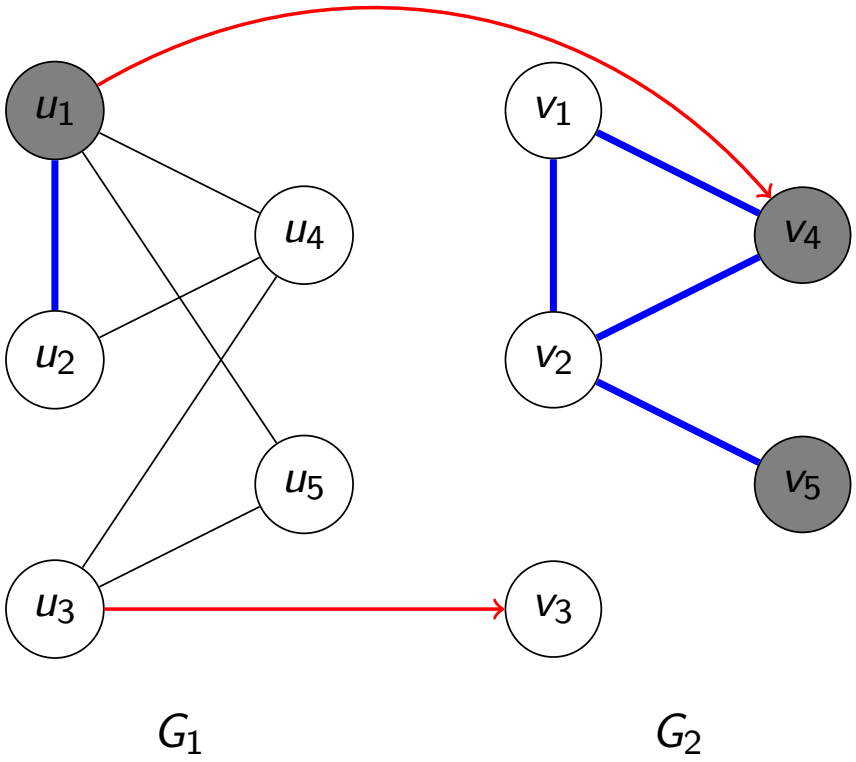


Partial solution: $u_1 \mapsto v_4$

Upper bound: $1 + 2$

Label	G_1	G_2
010	u_2	v_1, v_2
011	u_4, u_5	$\emptyset$

McSplit: a Branch and Bound Algorithm

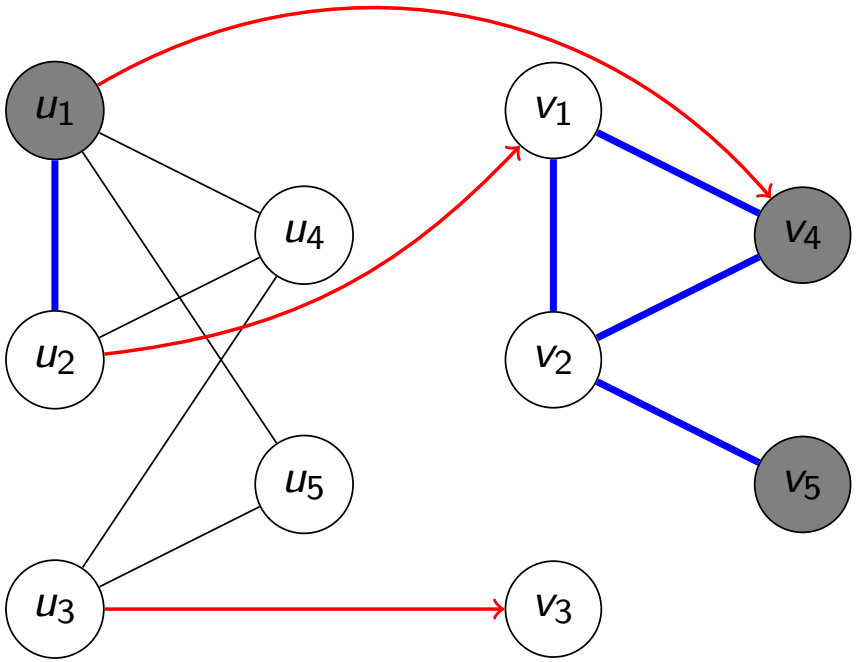


Partial solution: $u_1 \mapsto v_4$, $u_3 \mapsto v_3$

Upper bound: $2 + 1$

Label	G_1	G_2
010	u_2	v_1, v_2

McSplit: a Branch and Bound Algorithm



Partial solution: $u_1 \mapsto v_4,\ u_3 \mapsto v_3$
Upper bound: $2 + 1$

Label	G_1	G_2
010	u_2	v_1, v_2

Decision: $u_2 \mapsto v_1$
Found a solution!
Backtrack to confirm optimality

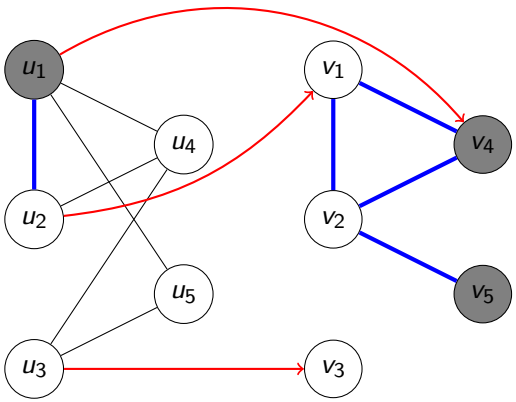
$k\downarrow$

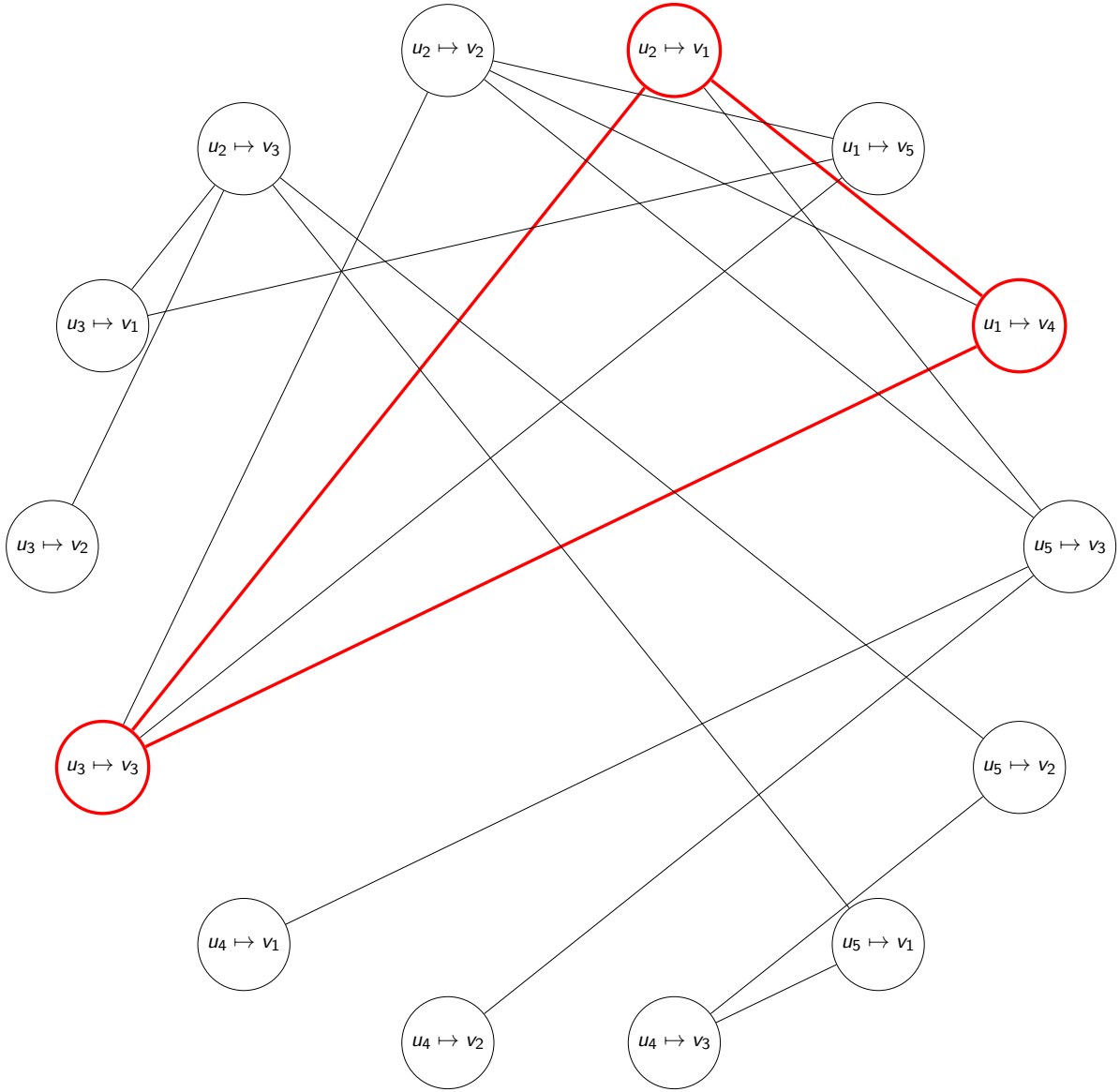
- $k = 0$: search for a complete subgraph isomorphism
- $k = 1$: allow one vertex of the smaller graph to not match anything
- ... and so on
- Developed to handle large instances
- Implements many domain filtering techniques

McSplit↓

- The main idea of $k\downarrow$ applied to McSplit
- Looks for a common subgraph of a set size
  - (decreasing with every iteration)
- This allows us to prune more search tree branches

Clique Encoding





Outline

(Per-Instance) Algorithm Selection

Definition (Bischl et al. 2016)

Given a set $\mathcal{I}$ of problem instances, a space of algorithms $\mathcal{A}$, and a performance measure $m\colon \mathcal{I} \times \mathcal{A} \to \mathbb{R}$, the *algorithm selection problem* is to find a mapping $s\colon \mathcal{I} \to \mathcal{A}$ that optimises $\mathbb{E}[m(i, s(i))]$.

(Per-Instance) Algorithm Selection

Definition (Bischl et al. 2016)

Given a set $\mathcal{I}$ of problem instances, a space of algorithms $\mathcal{A}$, and a performance measure $m\colon \mathcal{I} \times \mathcal{A} \to \mathbb{R}$, the *algorithm selection problem* is to find a mapping $s\colon \mathcal{I} \to \mathcal{A}$ that optimises $\mathbb{E}[m(i, s(i))]$.

(G_1, G_2)

(Per-Instance) Algorithm Selection

Definition (Bischl et al. 2016)

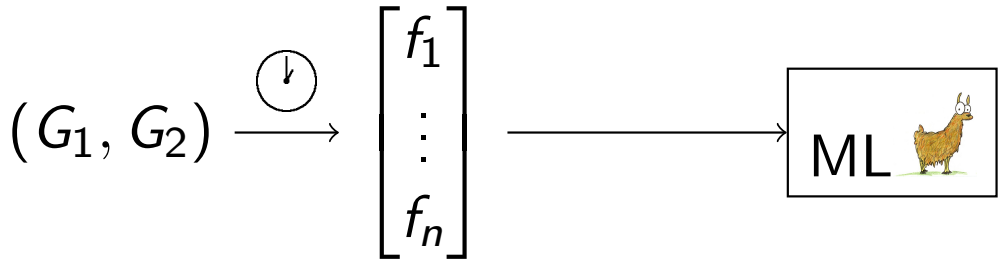
Given a set $\mathcal{I}$ of problem instances, a space of algorithms $\mathcal{A}$, and a performance measure $m\colon \mathcal{I} \times \mathcal{A} \to \mathbb{R}$, the *algorithm selection problem* is to find a mapping $s\colon \mathcal{I} \to \mathcal{A}$ that optimises $\mathbb{E}[m(i, s(i))]$.

$$(G_1, G_2) \xrightarrow{\text{clock}} \begin{bmatrix} f_1 \\ \vdots \\ f_n \end{bmatrix}$$

(Per-Instance) Algorithm Selection

Definition (Bischl et al. 2016)

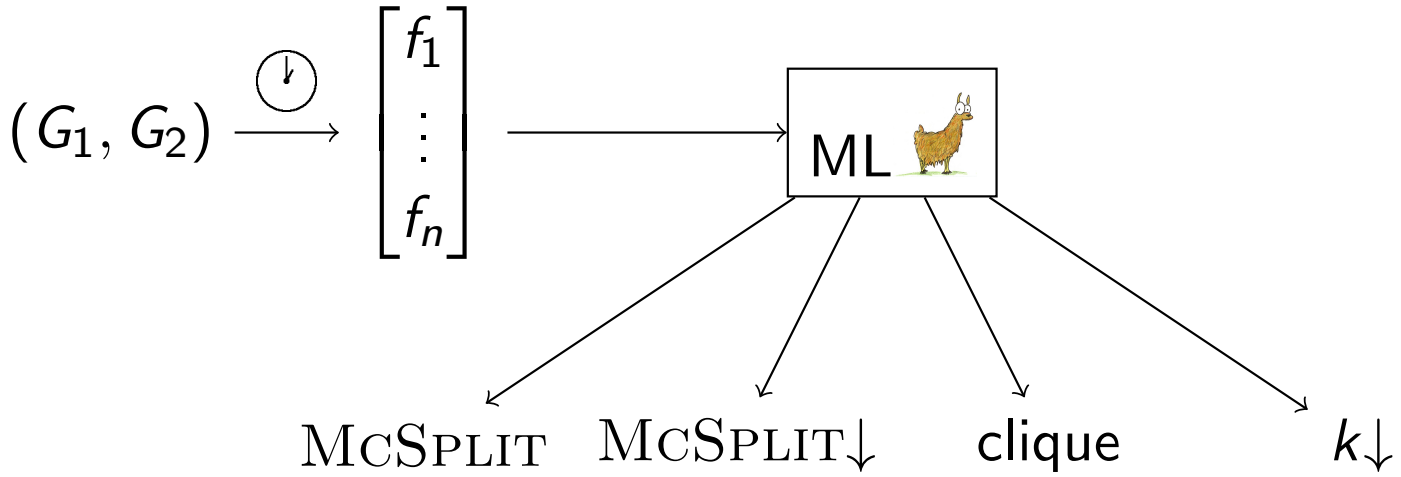
Given a set $\mathcal{I}$ of problem instances, a space of algorithms $\mathcal{A}$, and a performance measure $m\colon \mathcal{I} \times \mathcal{A} \to \mathbb{R}$, the *algorithm selection problem* is to find a mapping $s\colon \mathcal{I} \to \mathcal{A}$ that optimises $\mathbb{E}[m(i, s(i))]$.



(Per-Instance) Algorithm Selection

Definition (Bischl et al. 2016)

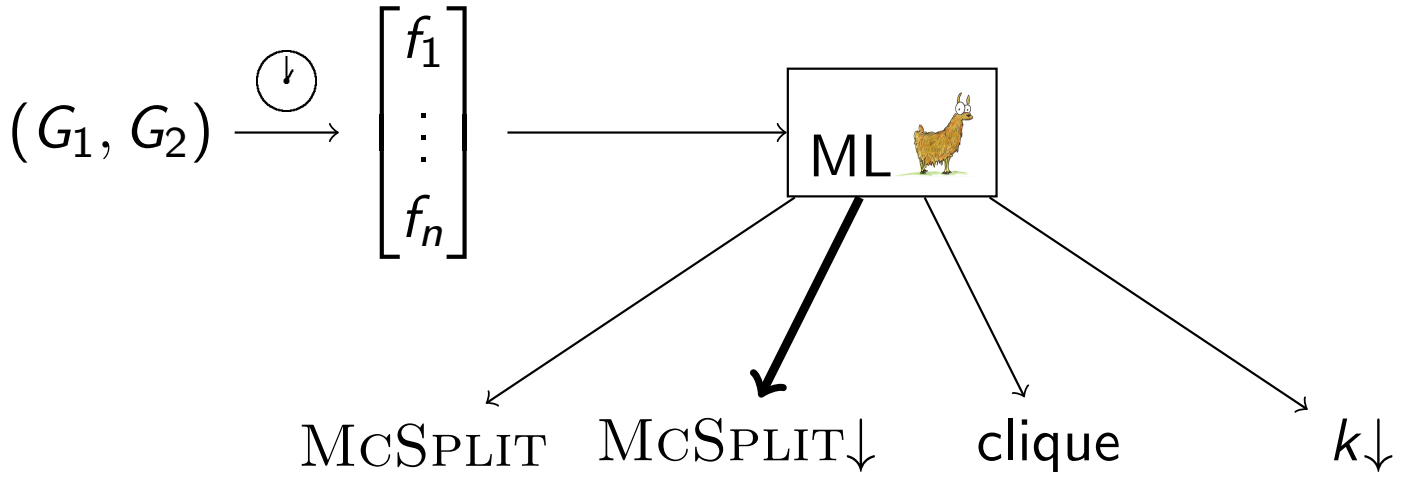
Given a set $\mathcal{I}$ of problem instances, a space of algorithms $\mathcal{A}$, and a performance measure $m\colon \mathcal{I} \times \mathcal{A} \to \mathbb{R}$, the *algorithm selection problem* is to find a mapping $s\colon \mathcal{I} \to \mathcal{A}$ that optimises $\mathbb{E}[m(i, s(i))]$.



(Per-Instance) Algorithm Selection

Definition (Bischl et al. 2016)

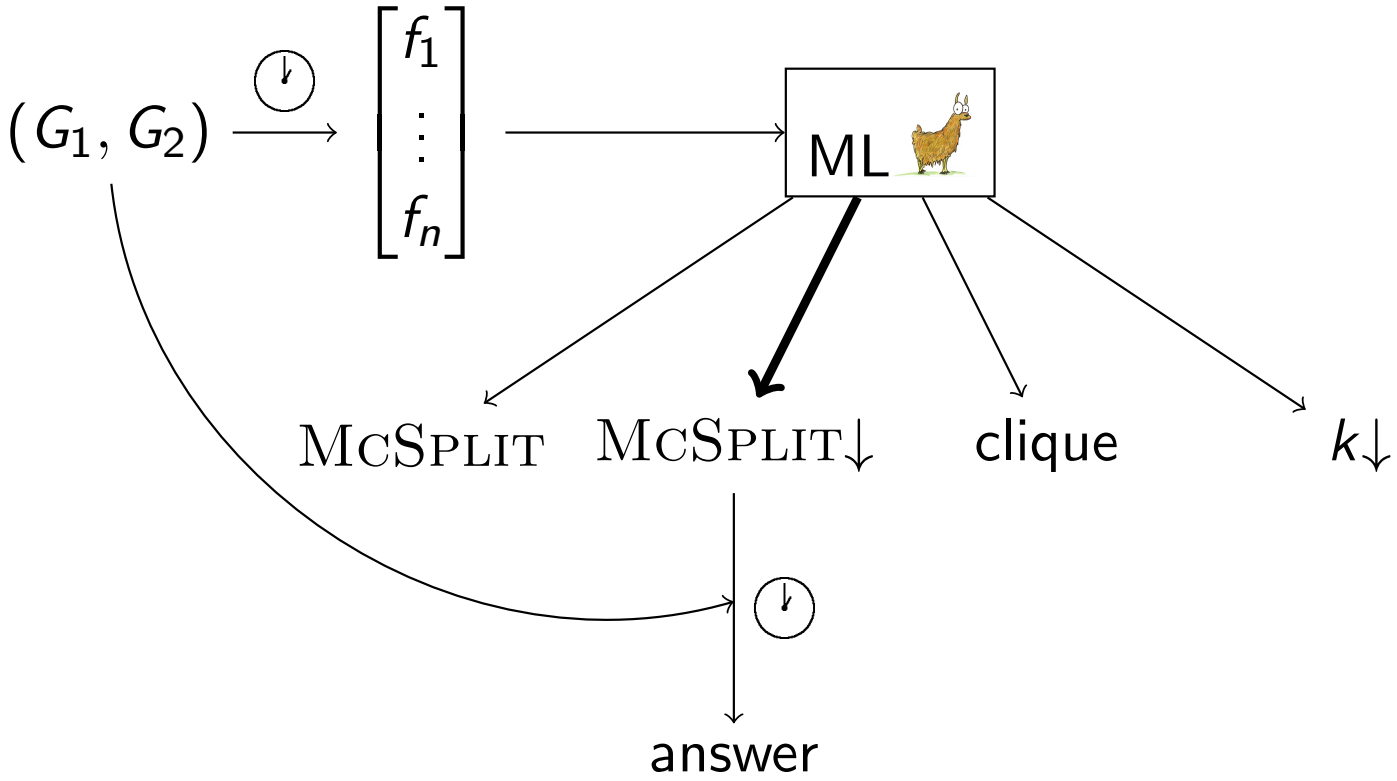
Given a set $\mathcal{I}$ of problem instances, a space of algorithms $\mathcal{A}$, and a performance measure $m\colon \mathcal{I} \times \mathcal{A} \to \mathbb{R}$, the *algorithm selection problem* is to find a mapping $s\colon \mathcal{I} \to \mathcal{A}$ that optimises $\mathbb{E}[m(i, s(i))]$.



(Per-Instance) Algorithm Selection

Definition (Bischl et al. 2016)

Given a set $\mathcal{I}$ of problem instances, a space of algorithms $\mathcal{A}$, and a performance measure $m\colon \mathcal{I} \times \mathcal{A} \to \mathbb{R}$, the *algorithm selection problem* is to find a mapping $s\colon \mathcal{I} \to \mathcal{A}$ that optimises $\mathbb{E}[m(i, s(i))]$.



Labelling

Data from Foggia, Sansone and Vento 2001; Santo et al. 2003 (81,400 pairs of graphs)

Labelling

Data from Foggia, Sansone and Vento 2001; Santo et al. 2003 (81,400 pairs of graphs)

Definition

A *vertex-labelled graph* is a 3-tuple $G = (V, E, \mu)$, where
$\mu\colon V \to \{0, \ldots, N-1\}$ is a vertex labelling function, for some $N \in \mathbb{N}$.

Labelling

Data from Foggia, Sansone and Vento 2001; Santo et al. 2003 (81,400 pairs of graphs)

Definition

A *vertex-labelled graph* is a 3-tuple $G = (V, E, \mu)$, where
$\mu\colon V \to \{0, \ldots, N-1\}$ is a vertex labelling function, for some $N \in \mathbb{N}$.

Definition

A graph $G = (V, E, \mu)$ is said to have a $p\%$ *(vertex) labelling* if

$$N = \max\left\{2^n : n \in \mathbb{N},\, 2^n < \left\lfloor \frac{p}{100\%} \times |V| \right\rfloor \right\}.$$

Labelling

Definition

A graph $G = (V, E, \mu)$ is said to have a $p\%$ *(vertex) labelling* if

$$N = \max\left\{2^n : n \in \mathbb{N},\, 2^n < \left\lfloor \frac{p}{100\%} \times |V| \right\rfloor \right\}.$$

- 5% labelling - 20 vertices per label on average
- 50% labelling - 2 vertices per label on average

Labelling

Definition

A graph $G = (V, E, \mu)$ is said to have a $p\%$ *(vertex) labelling* if

$$N = \max\left\{2^n : n \in \mathbb{N},\, 2^n < \left\lfloor \frac{p}{100\%} \times |V| \right\rfloor \right\}.$$

- 5% labelling - 20 vertices per label on average
- 50% labelling - 2 vertices per label on average
- Typical values explored: 33%, 50%, 75%

Labelling

Definition

A graph $G = (V, E, \mu)$ is said to have a $p\%$ *(vertex) labelling* if

$$N = \max\left\{2^n : n \in \mathbb{N},\, 2^n < \left\lfloor \frac{p}{100\%} \times |V| \right\rfloor \right\}.$$

- 5% labelling - 20 vertices per label on average
- 50% labelling - 2 vertices per label on average
- Typical values explored: 33%, 50%, 75%
- In my data: 5%, 10%, 15%, 20%, 25%, 33%, 50%

Labelling

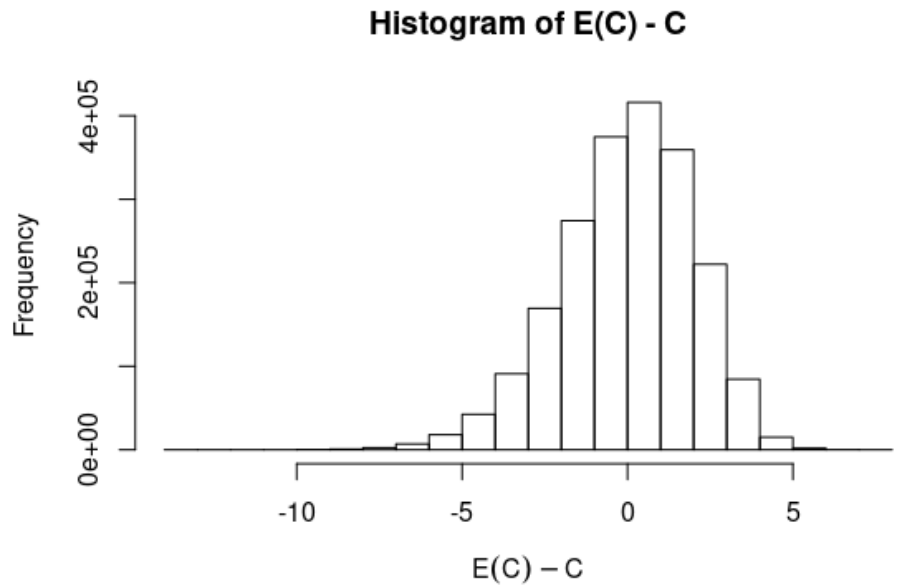
Definition

A graph $G = (V, E, \mu)$ is said to have a $p\%$ *(vertex) labelling* if

$$N = \max\left\{2^n : n \in \mathbb{N},\, 2^n < \left\lfloor \frac{p}{100\%} \times |V| \right\rfloor \right\}.$$

- 5% labelling - 20 vertices per label on average
- 50% labelling - 2 vertices per label on average
- Typical values explored: 33%, 50%, 75%
- In my data: 5%, 10%, 15%, 20%, 25%, 33%, 50%
- 3 different cases:
  - no labels
  - vertex labels
  - vertex and edge labels

Number of Vertices Per Label



For each graph and label

- C is the number of vertices with that label
- E(C) is the number we would expect from a (discrete) uniform distribution

Features (34 in total)

1–8 are from Kotthoff, McCreesh and Solnon 2016

1. number of vertices
2. number of edges
3. mean/max degree
4. density
5. mean/max distance between pairs of vertices
6. number of loops
7. proportion of vertex pairs with distance $\geq$ 2, 3, 4
8. connectedness

Features (34 in total)

1–8 are from Kotthoff, McCreesh and Solnon 2016

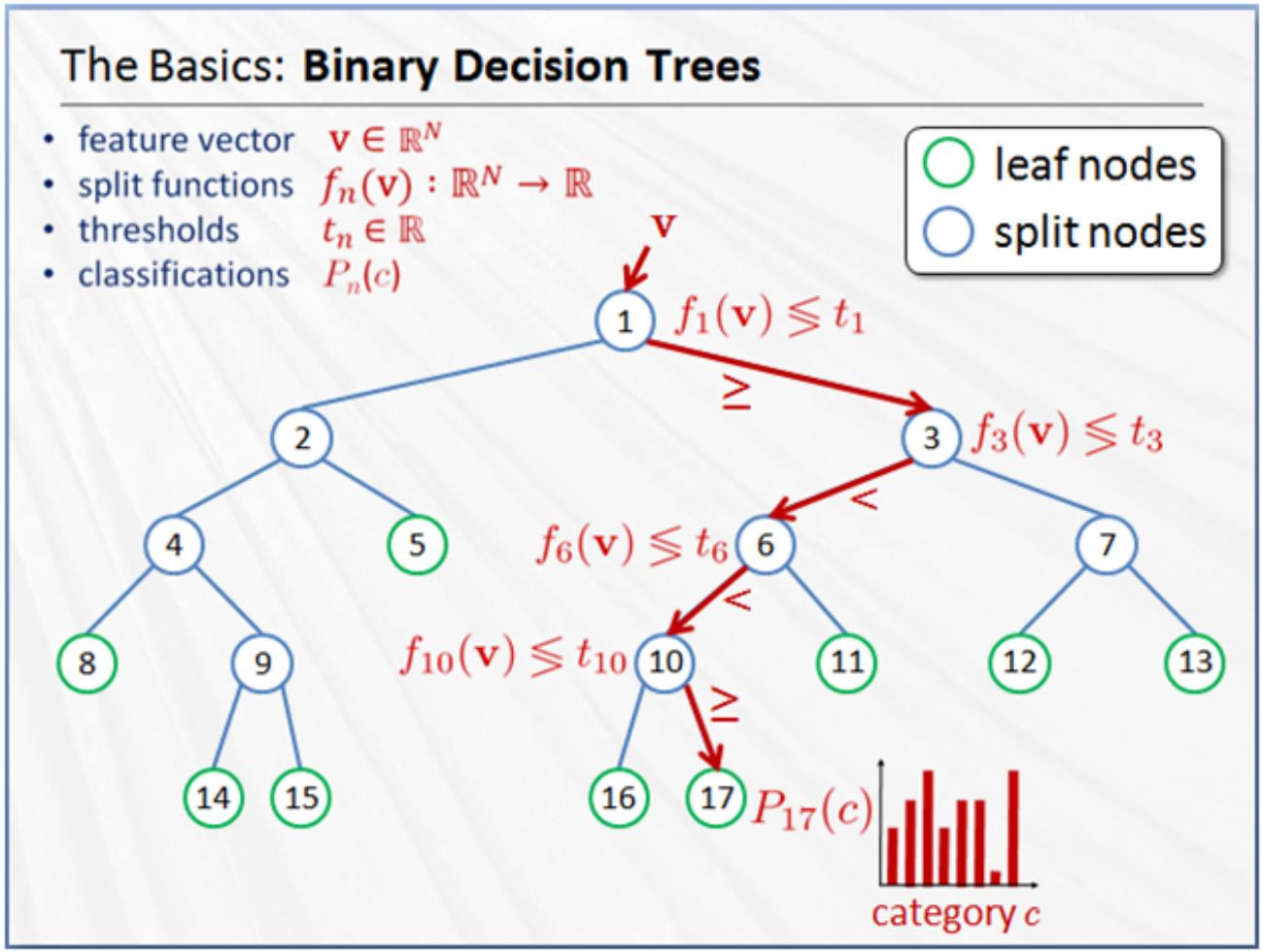
1. number of vertices
2. number of edges
3. mean/max degree
4. density
5. mean/max distance between pairs of vertices
6. number of loops
7. proportion of vertex pairs with distance $\geq$ 2, 3, 4
8. connectedness
9. standard deviation of degrees
10. labelling percentage

Features (34 in total)

1–8 are from Kotthoff, McCreesh and Solnon 2016

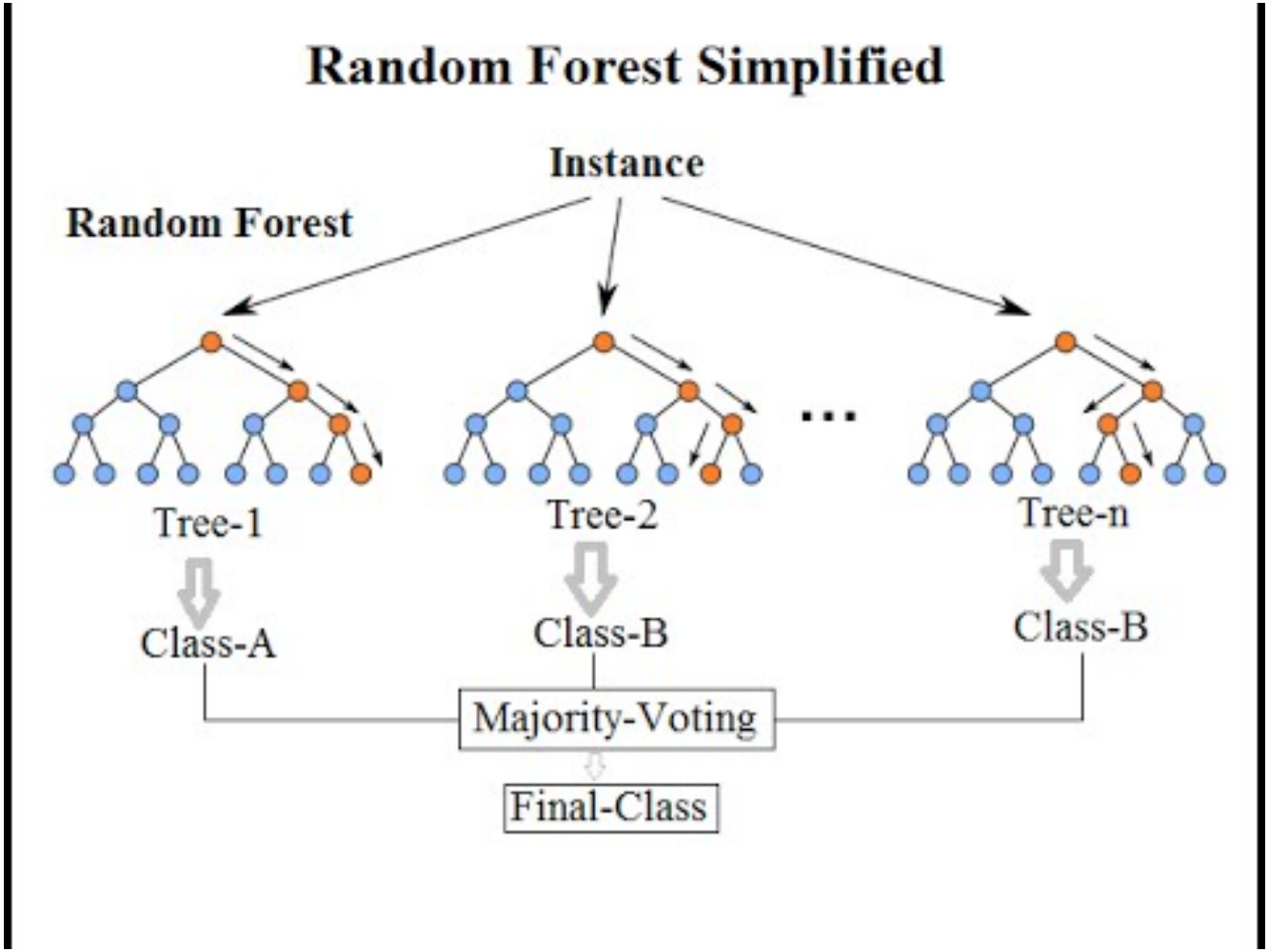
1. number of vertices
2. number of edges
3. mean/max degree
4. density
5. mean/max distance between pairs of vertices
6. number of loops
7. proportion of vertex pairs with distance $\geq$ 2, 3, 4
8. connectedness
9. standard deviation of degrees
10. labelling percentage
11. ratios of features 1–5

Random Forests (Breiman 2001)



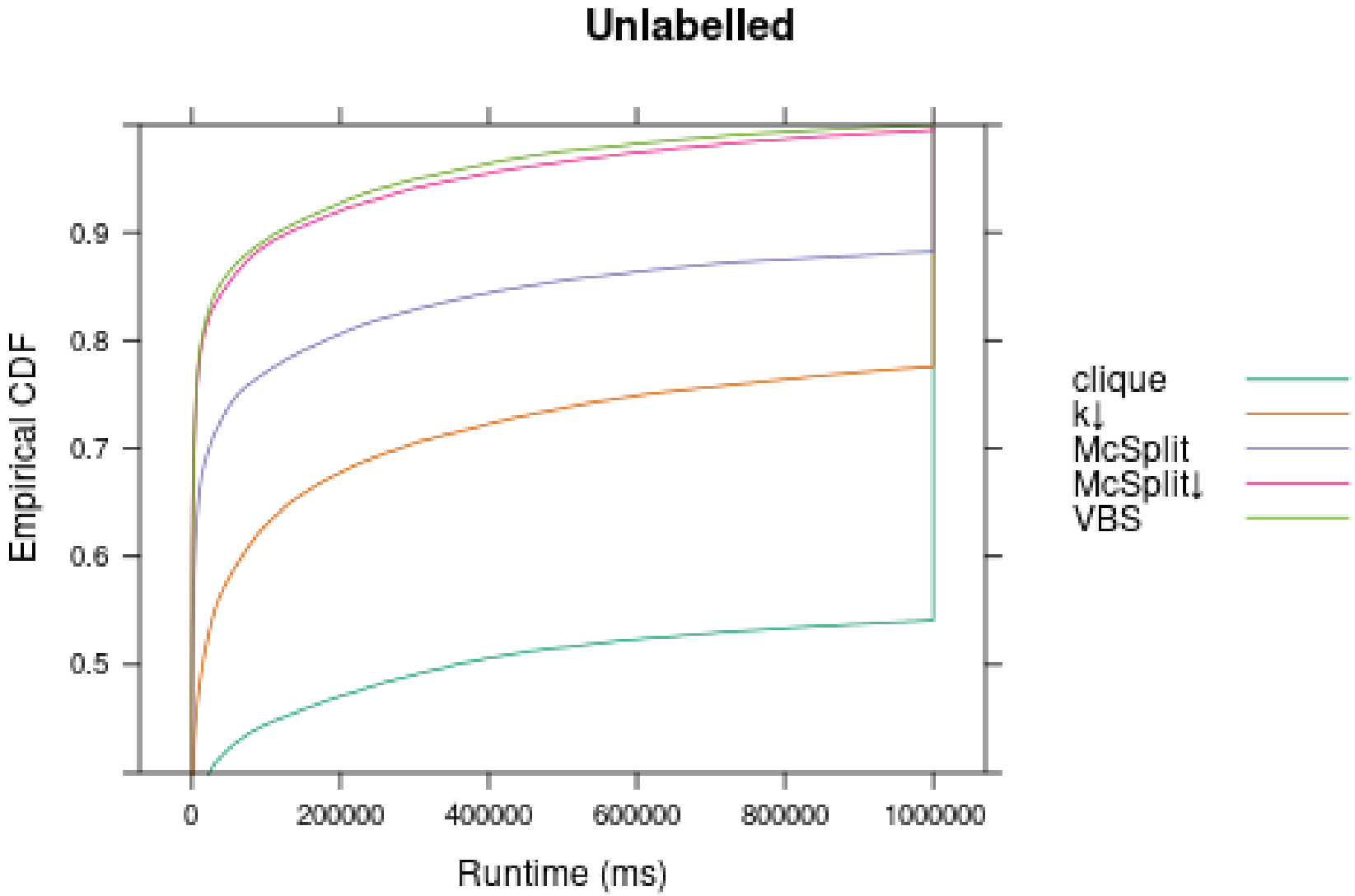
Source: Tae-Kyun Kim & Bjorn Stenger, Intelligent Systems and Networks (ISN) Research Group, Imperial College London

Random Forests (Breiman 2001)

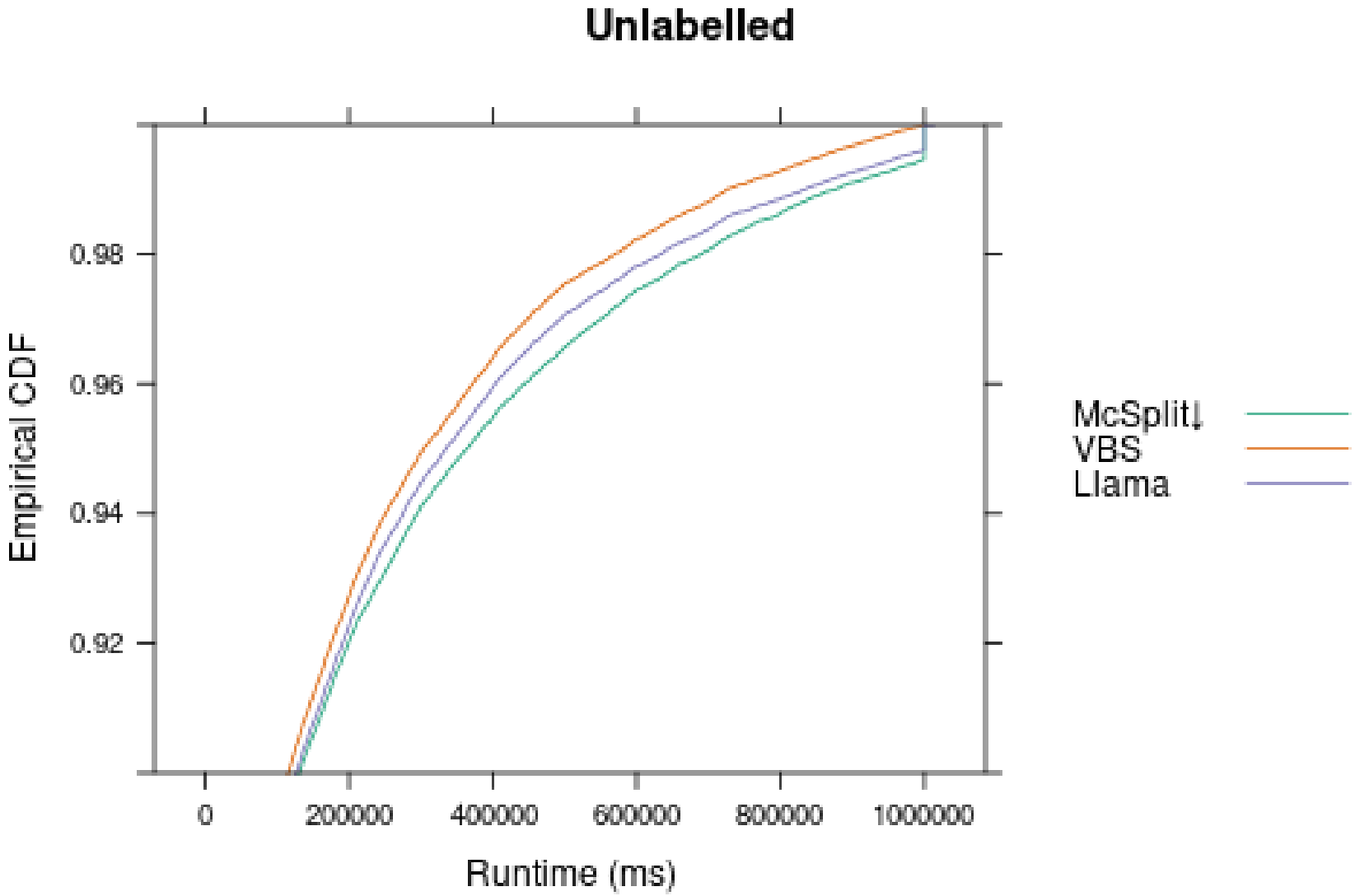


Source: Random Forests(r), Explained, Ilan Reinstein, KDnuggets

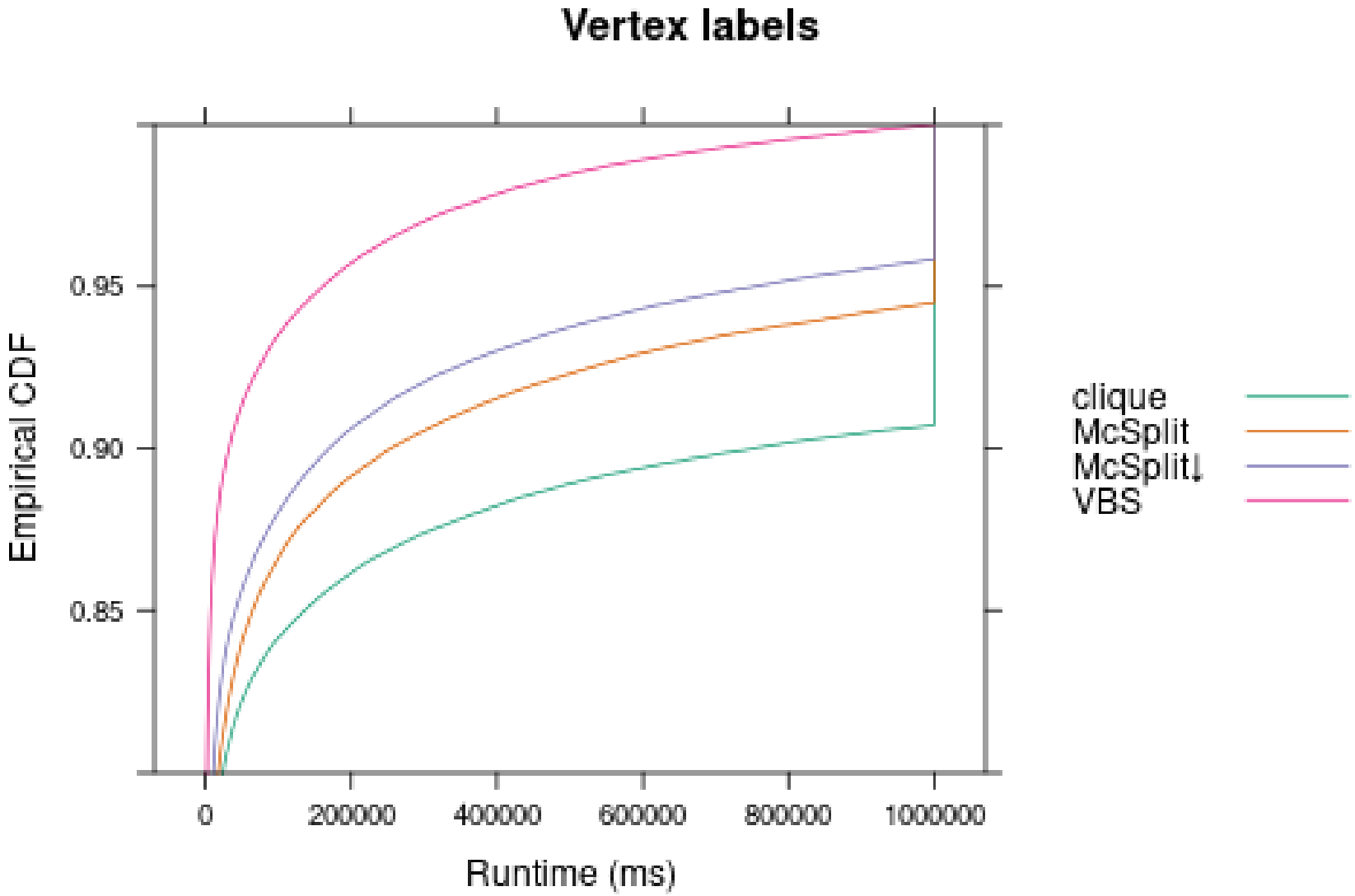
Results


Unlabelled
Empirical CDF
0.9
0.8
0.7
0.6
0.5
0
200000
400000
600000
800000
1000000
Runtime (ms)
clique
k↓
McSplit
McSplit↓
VBS

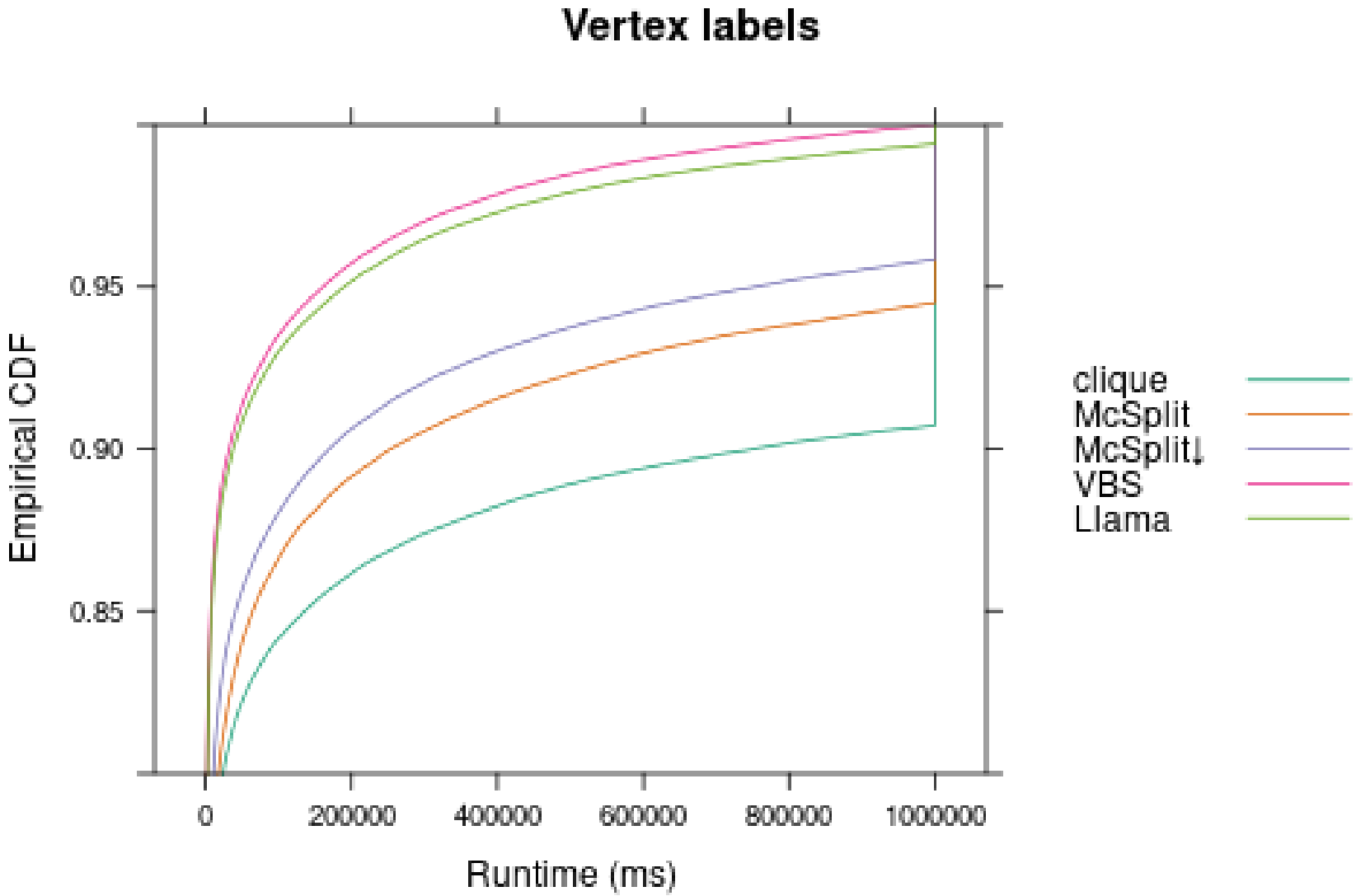
Results (27%)


Unlabelled
Empirical CDF
0.98
0.96
0.94
0.92
0
200000
400000
600000
800000
1000000
Runtime (ms)
McSplit↓
VBS
Llama

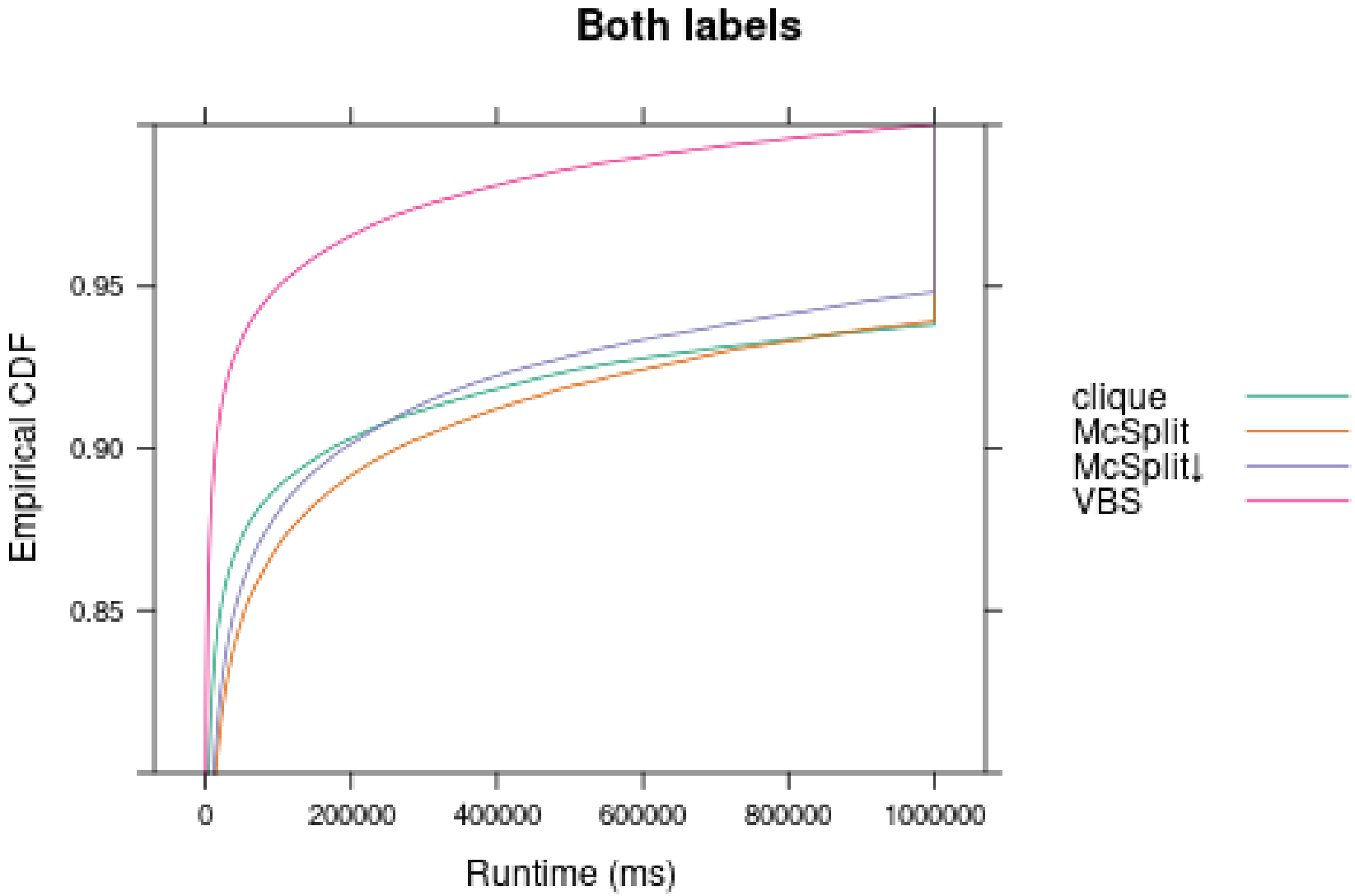
Results


Vertex labels
Empirical CDF
0.95
0.90
0.85
0
200000
400000
600000
800000
1000000
Runtime (ms)
clique
McSplit
McSplit↓
VBS

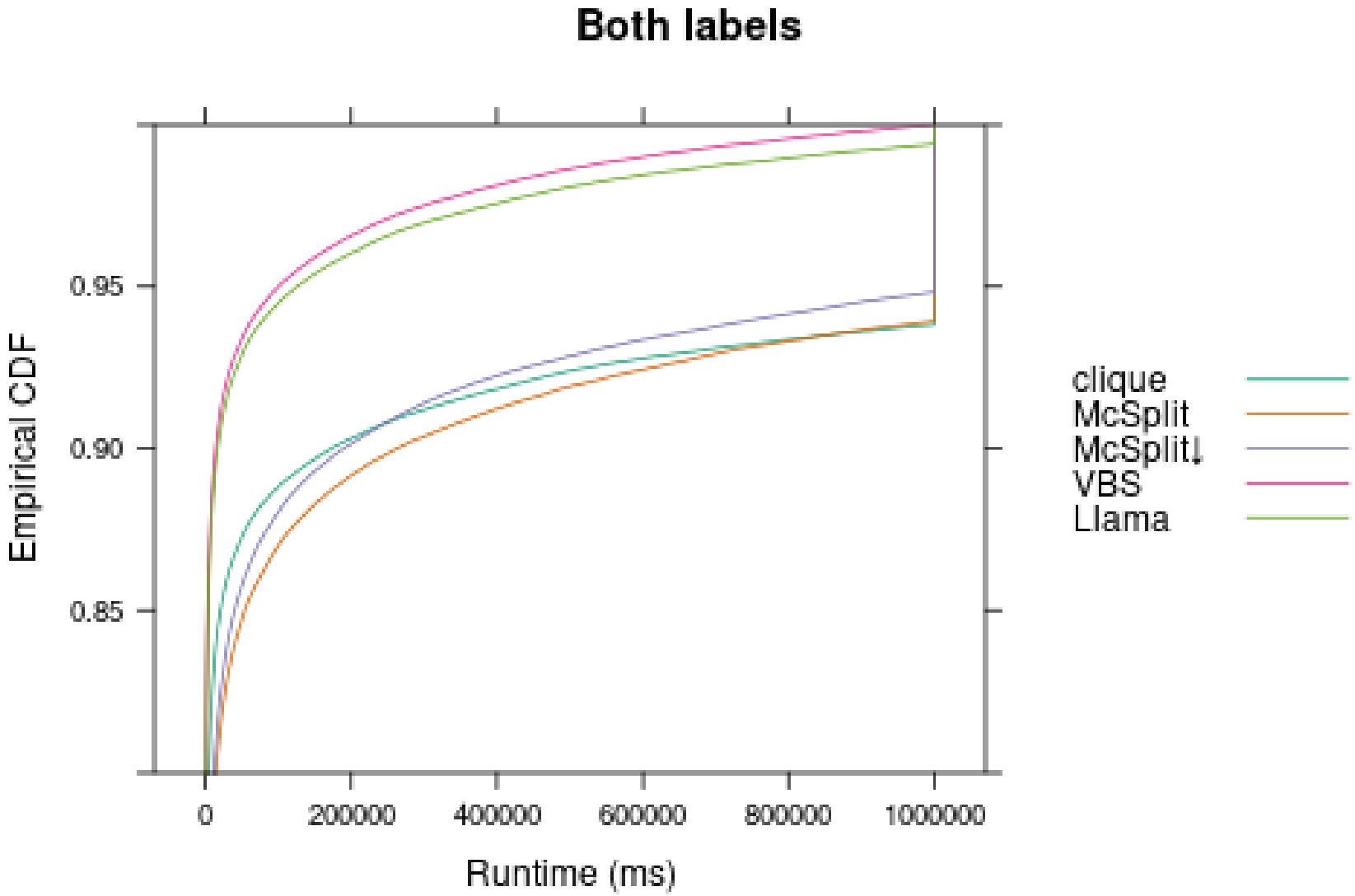
Results (86%)


Vertex labels
Empirical CDF
0.95
0.90
0.85
0
200000
400000
600000
800000
1000000
Runtime (ms)
clique
McSplit
McSplit↓
VBS
Llama

Results


Both labels
Empirical CDF
0.95
0.90
0.85
0
200000
400000
600000
800000
1000000
Runtime (ms)
clique
McSplit
McSplit↓
VBS

Results (88%)


Both labels
Empirical CDF
0.95
0.90
0.85
0
200000
400000
600000
800000
1000000
Runtime (ms)
clique
McSplit
McSplit↓
VBS
Llama

Errors

- Out-of-bag error
- For each algorithm
  - $1 -$ recall

Definition

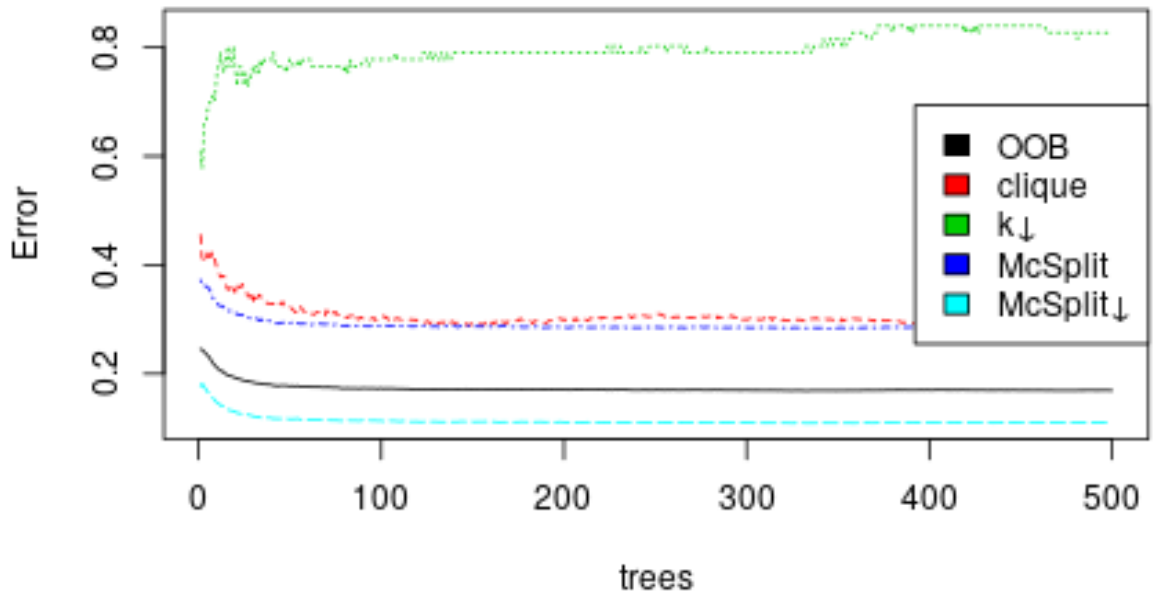
For an algorithm A, *recall* (sensitivity) is

$$\frac{\text{the number of instances that were correctly predicted as } A}{\text{the number of instances where } A \text{ is the correct prediction}}.$$

Errors (%)

Error	Labelling		
	no	vertex	both
out-of-bag	17	13	14
clique	30	8	7
McSplit	29	22	29
McSplit ↓	11	11	11
$k \downarrow$	80		

Convergence of Errors for Unlabelled Graphs



Outline

Margins

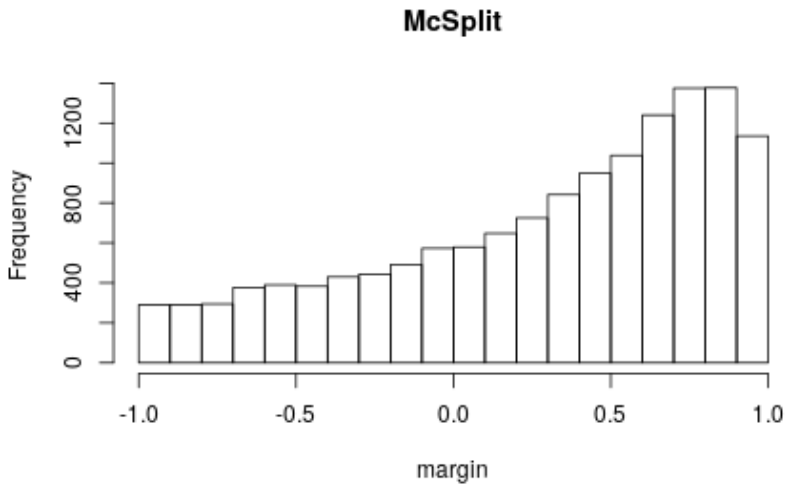
Definition

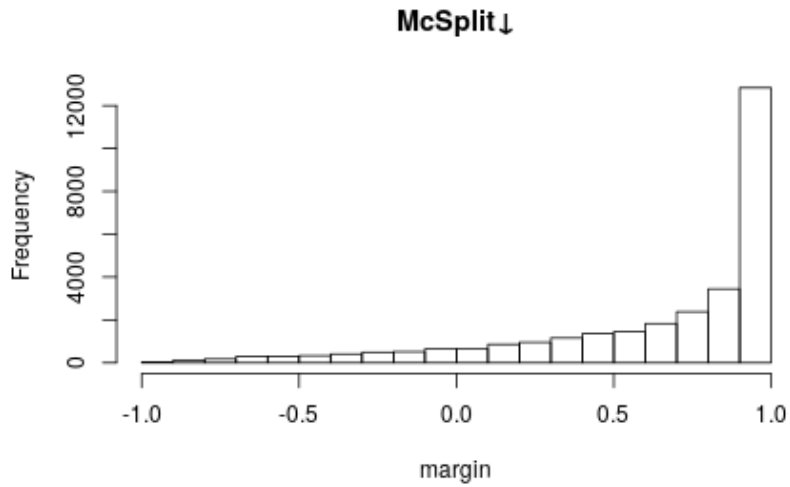
Let $c_1, \ldots, c_n$ be n classes and let p be a data point that belongs to class c_p. Let $v_1, \ldots, v_n$ denote the number of votes for each class when given p as input. The *margin* of p is

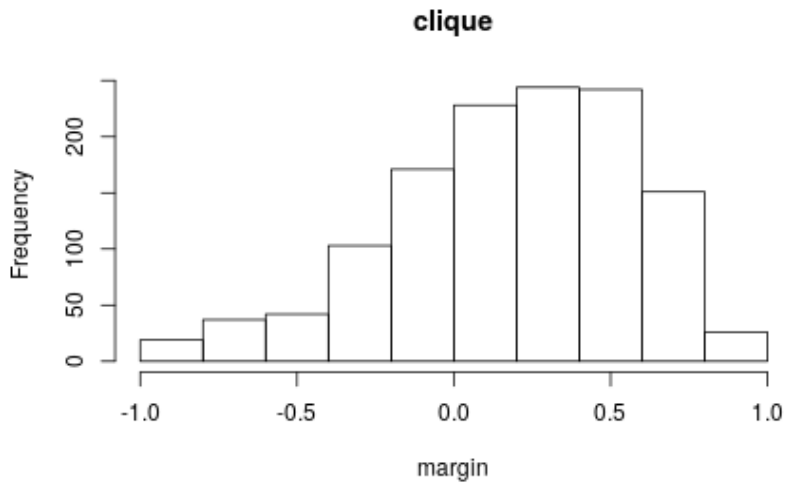
$$\frac{v_p}{\sum_{i=1}^{n} v_i} - \max_{i \neq p} \frac{v_i}{\sum_{j=1}^{n} v_j},$$

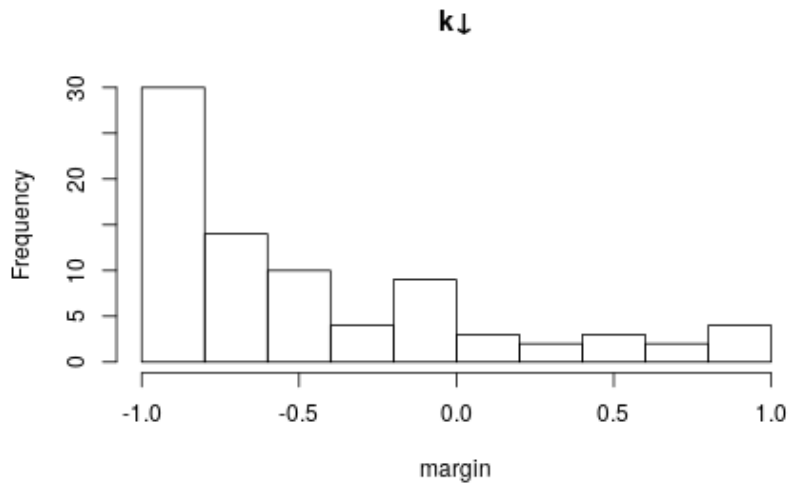
which is a number in $[-1, 1]$.

Margins: Unlabelled Graphs

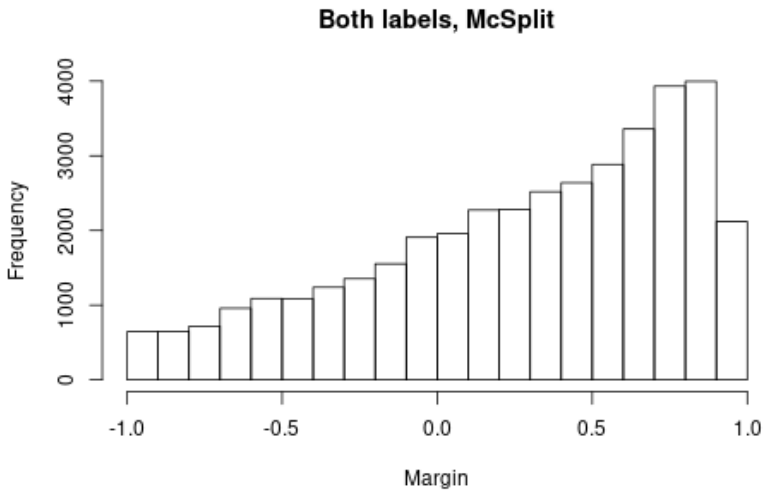


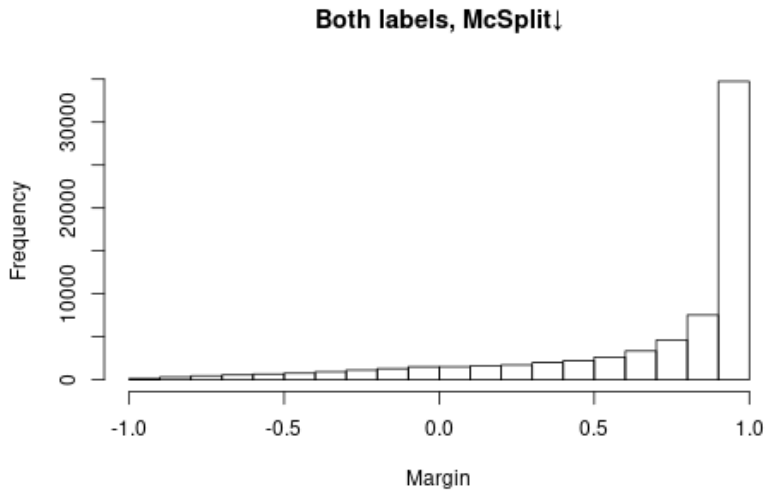


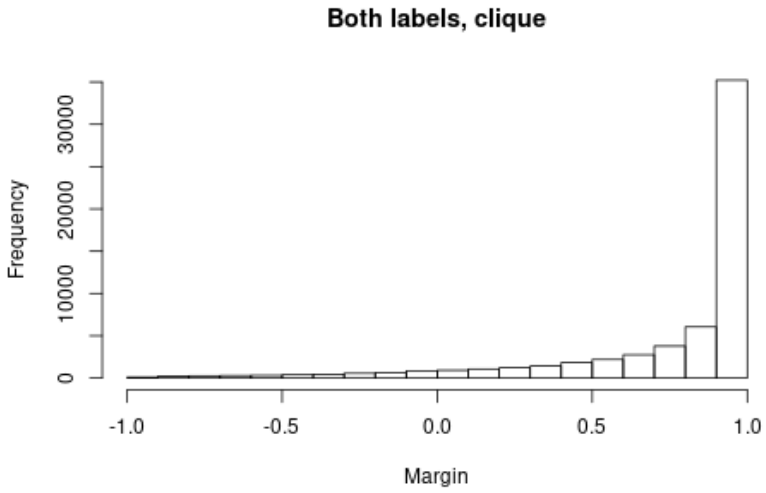




Margins: Vertex and Edge Labels







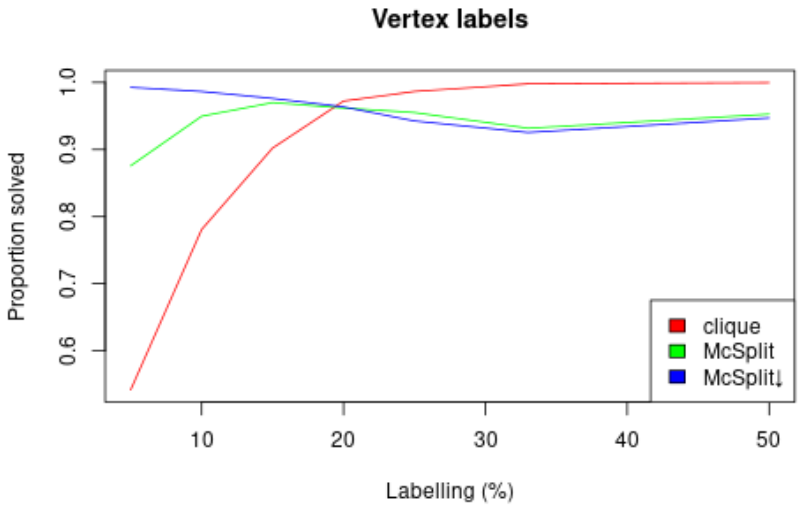
Partial Dependence

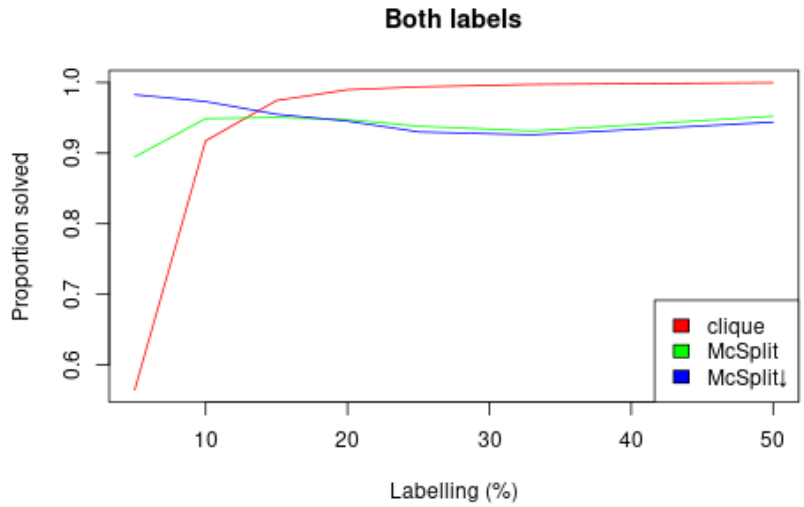
Defined as

$$f(x) = \log p_k(x) - \frac{1}{K} \sum_{i=1}^{K} \log p_i(x)$$

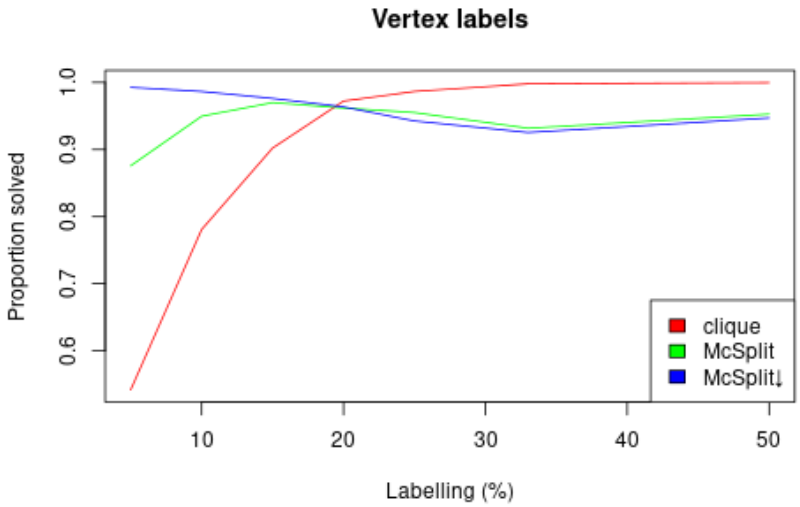
- x iterates over all values of a feature
- $p_i(x)$ is the proportion of votes for algorithm i, for a given value of x
- K is the number of different algorithms
- k is the algorithm under consideration

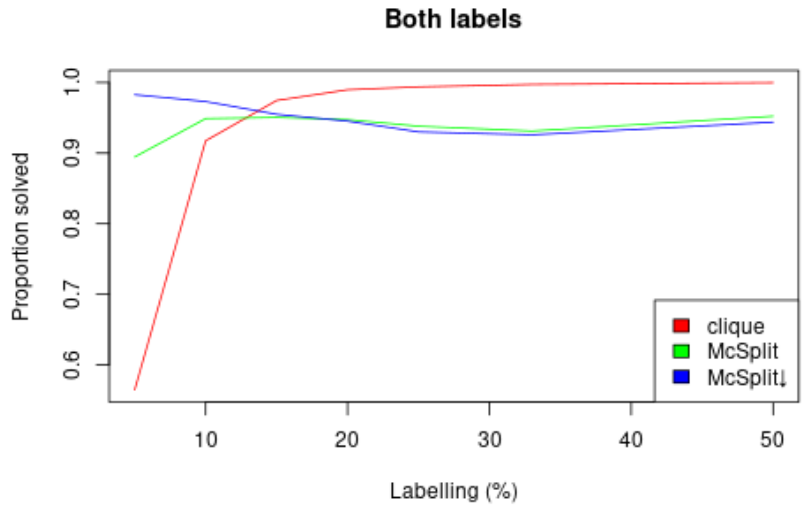
What Happens When Labelling Changes?

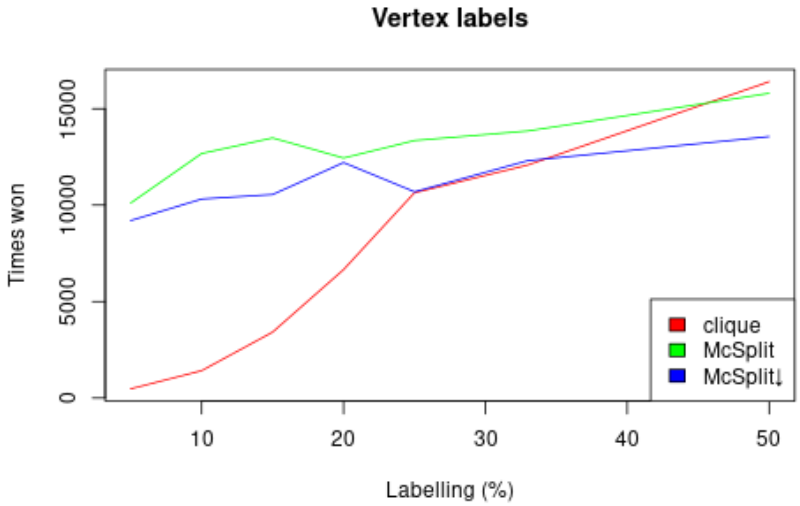


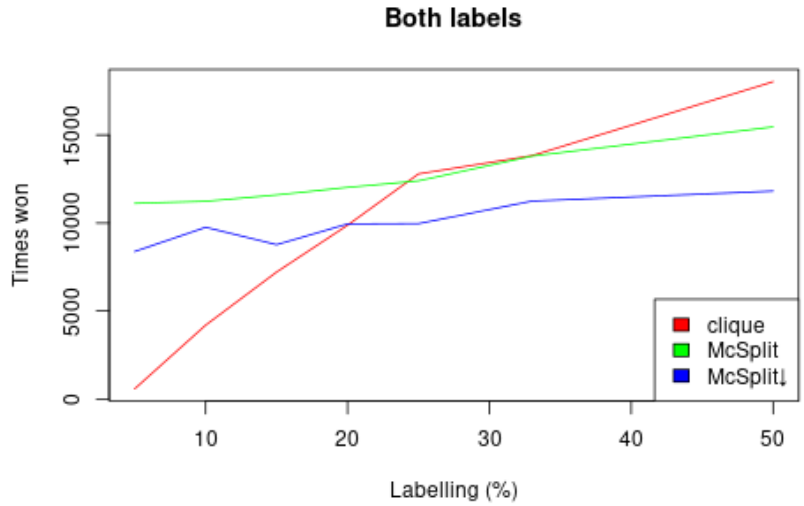


What Happens When Labelling Changes?









Outline

Idea 1: Switch After a Fixed Number of Decisions

Idea 1: Switch After a Fixed Number of Decisions

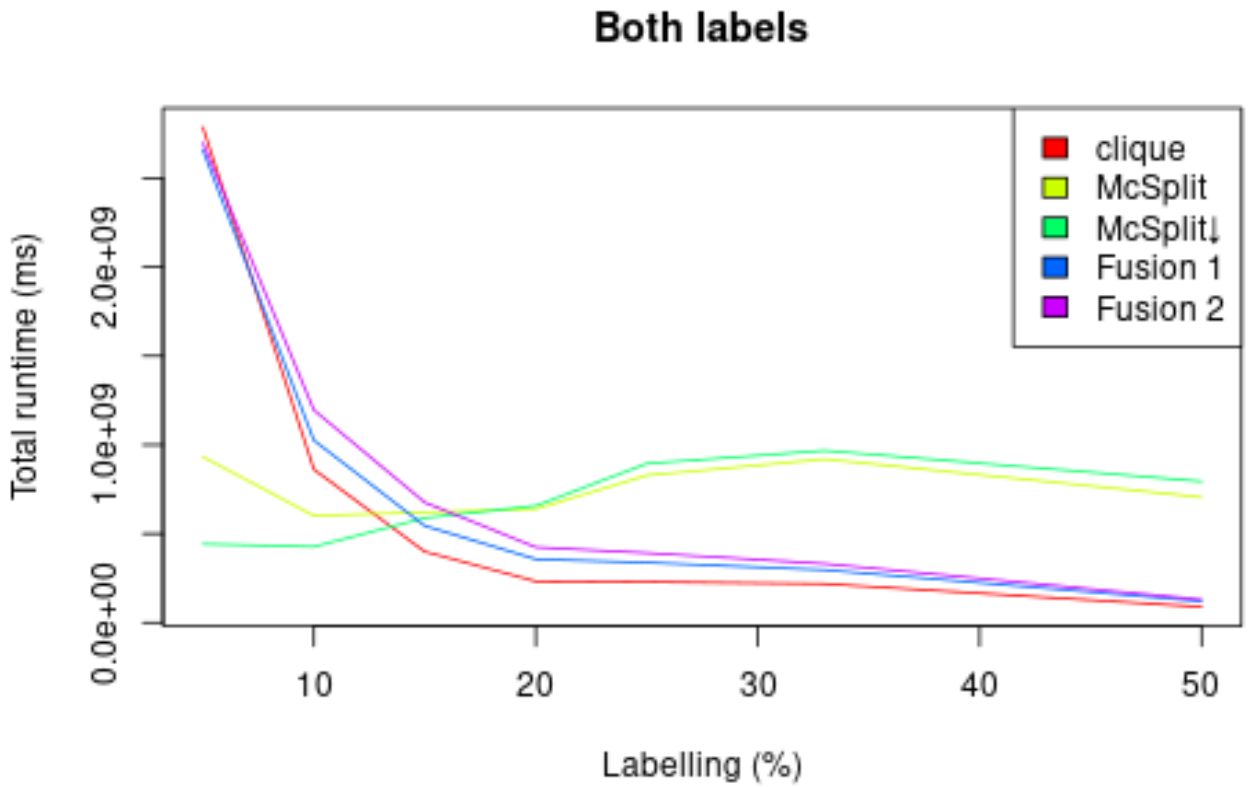
- Vertices of the association graph can be constructed from McSplit label classes, edges from the original input graphs
- Only a few extra lines of code:

$$|incumbent_{\text{clique}}| \leftarrow |incumbent_{\text{McSplit}}| - |M|$$

and then

$$incumbent_{\text{McSplit}} \leftarrow incumbent_{\text{McSplit}} \cup incumbent_{\text{clique}}$$

Not That Good...



Idea 2: From Partially Solved to Unsolved Instances

- Goal: switch algorithms in a more intelligent way
- Plan: use runtime data on unsolved instances
  - using either machine learning or simple guidelines

Idea 2: From Partially Solved to Unsolved Instances

- Goal: switch algorithms in a more intelligent way
- Plan: use runtime data on unsolved instances
  - using either machine learning or simple guidelines
- Implementation can be optimised to:
  - only track important information
  - reuse information between iterations

Idea 2: From Partially Solved to Unsolved Instances

- Goal: switch algorithms in a more intelligent way
- Plan: use runtime data on unsolved instances
  - using either machine learning or simple guidelines
- Implementation can be optimised to:
  - only track important information
  - reuse information between iterations
- Is it any good?

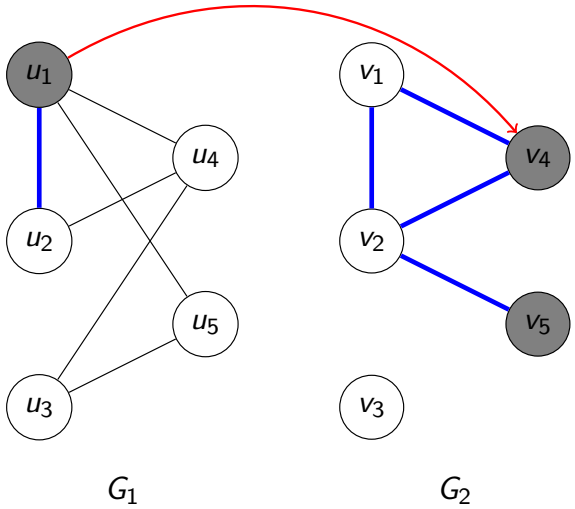
Idea 2: From Partially Solved to Unsolved Instances

- Goal: switch algorithms in a more intelligent way
- Plan: use runtime data on unsolved instances
  - using either machine learning or simple guidelines
- Implementation can be optimised to:
  - only track important information
  - reuse information between iterations
- Is it any good?
  - Remains to be seen
  - Could be that...

Idea 2: From Partially Solved to Unsolved Instances

- Goal: switch algorithms in a more intelligent way
- Plan: use runtime data on unsolved instances
  - using either machine learning or simple guidelines
- Implementation can be optimised to:
  - only track important information
  - reuse information between iterations
- Is it any good?
  - Remains to be seen
  - Could be that...
    - the answer is "never switch"
    - or the performance gains are minimal

Idea 2: From Partially Solved to Unsolved Instances

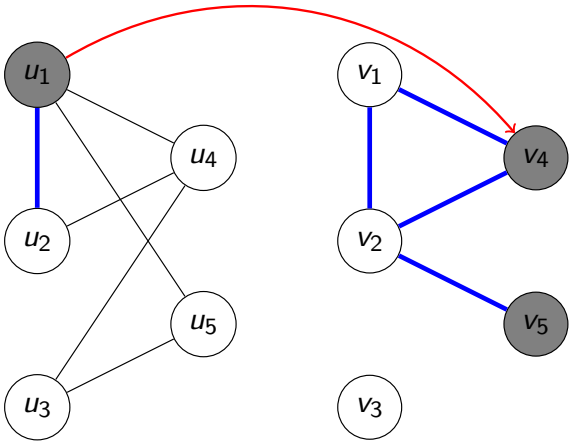


Label	G_1	G_2
00	u_3	v_3
01	u_2	v_1, v_2

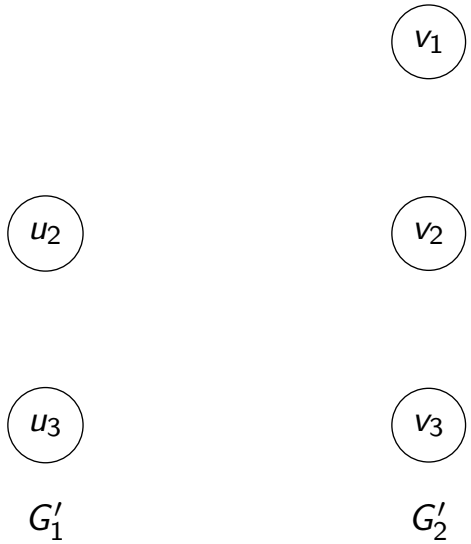
Partial solution and incumbent:

$incumbent = M = \{u_1 \mapsto v_4\}$

Idea 2: From Partially Solved to Unsolved Instances



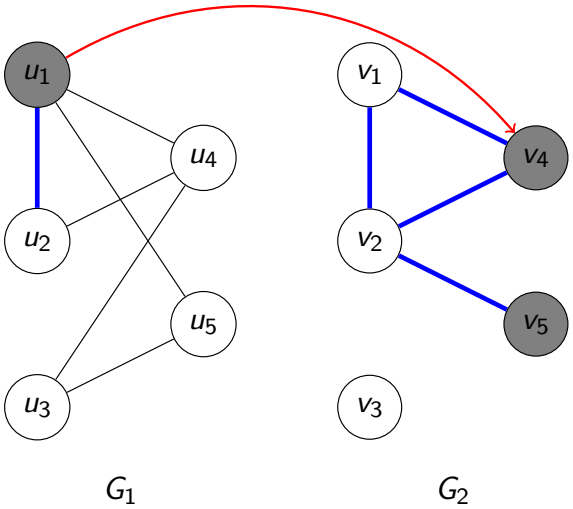
Label	G_1	G_2
00	u_3	v_3
01	u_2	v_1, v_2



Step 1: Add vertices from label classes

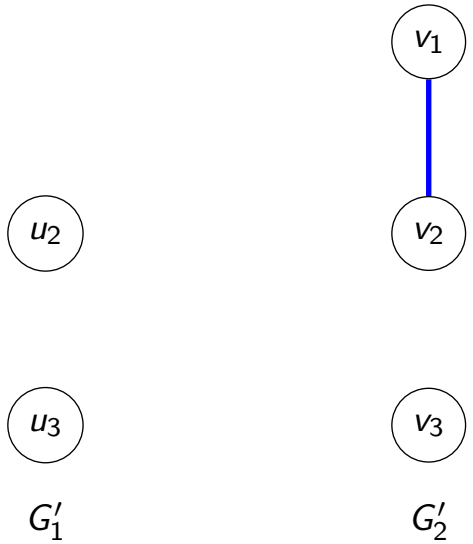
Partial solution and incumbent:
$incumbent = M = \{u_1 \mapsto v_4\}$

Idea 2: From Partially Solved to Unsolved Instances



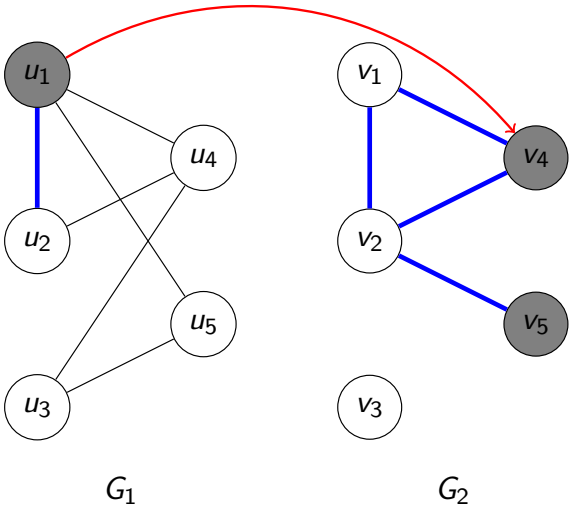
Label	G_1	G_2
00	u_3	v_3
01	u_2	v_1, v_2

Partial solution and incumbent:
$incumbent = M = \{u_1 \mapsto v_4\}$



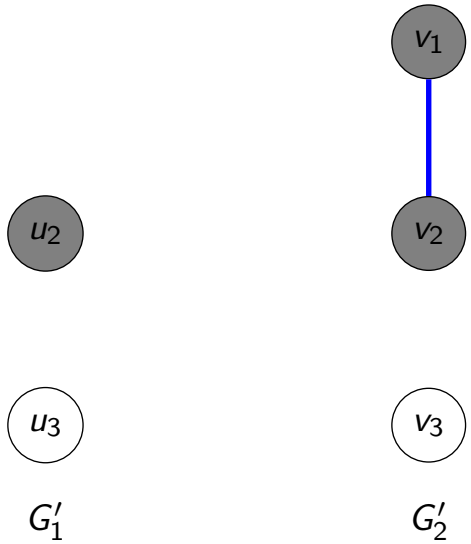
Step 2: $E' = E \cap (V_1' \times V_1')$
(preserving edge labels)

Idea 2: From Partially Solved to Unsolved Instances



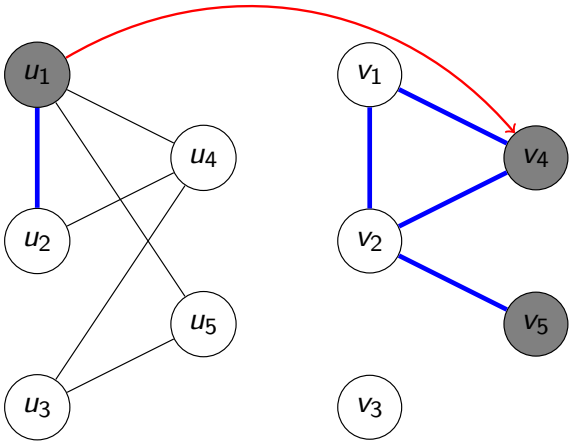
Label	G_1	G_2	New label
00	u_3	v_3	0
01	u_2	v_1, v_2	1

Partial solution and incumbent:
$incumbent = M = \{u_1 \mapsto v_4\}$



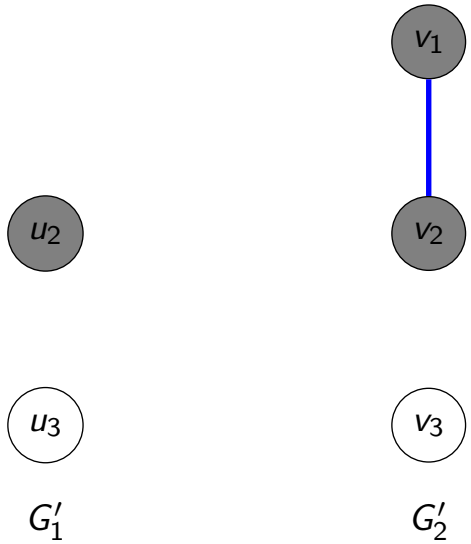
Step 3: label vertices according to vertex classes

Idea 2: From Partially Solved to Unsolved Instances



Label	G_1	G_2	New label
00	u_3	v_3	0
01	u_2	v_1, v_2	1

Partial solution and incumbent:
$incumbent = M = \{u_1 \mapsto v_4\}$



Step 4: Set

$$|incumbent'| = |incumbent| - |M|$$

Thank You!

The project was supervised by

Dr Patrick Prosser and Dr Ciaran McCreesh

Dissertation and code available at

`https://github.com/dilkas/maximum-common-subgraph`