

Maximum Common Subgraph

Algorithms and Algorithm Portfolios

Paulius Dilkas

Supervisors: Dr Patrick Prosser and Dr Ciaran McCreesh

School of Computing Science
University of Glasgow

21st March 2018

Outline

- 1 The Problem
- 2 Algorithms
- 3 Understanding the Data
- 4 Algorithm Selection
- 5 Observations and Insights
- 6 Switching Algorithms Mid-Execution

Outline

- 1 The Problem
- 2 Algorithms
- 3 Understanding the Data
- 4 Algorithm Selection
- 5 Observations and Insights
- 6 Switching Algorithms Mid-Execution

Maximum Common Subgraph

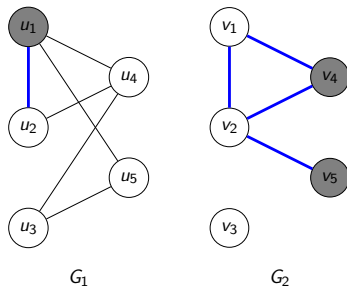
Definition

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

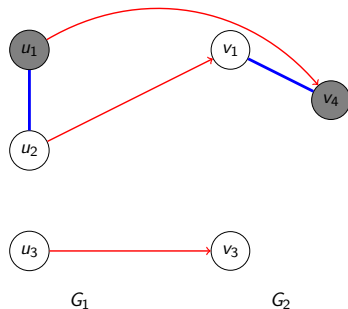
A *maximum common (induced) subgraph* between graphs G_1 and G_2 is a graph $G_3 = (V_3, E_3)$ such that G_3 is isomorphic to induced subgraphs of both G_1 and G_2 with $|V_3|$ maximised.



Maximum Common Subgraph

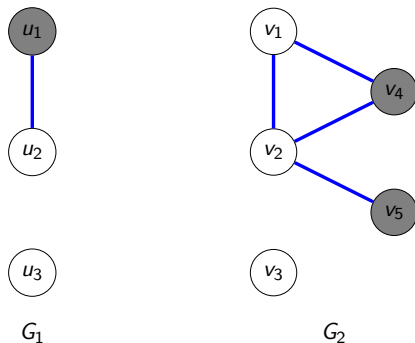
Definition

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.



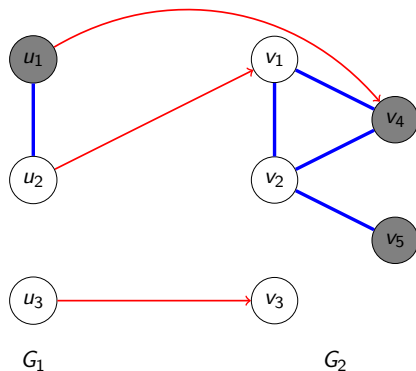
Related: Subgraph Isomorphism

- A decision problem: is G_1 isomorphic to a subgraph of G_2 ?
- G_1 is the **pattern** graph
- G_2 is the **target** graph

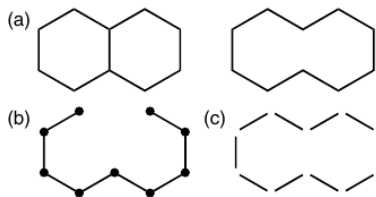


Related: Subgraph Isomorphism

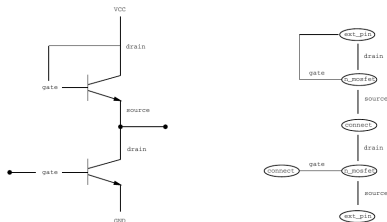
- A decision problem: is G_1 isomorphic to a subgraph of G_2 ?
- G_1 is the **pattern** graph
- G_2 is the **target** graph



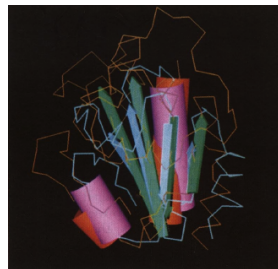
Why Is It Important?



Source: Ehrlich and Rarey 2011



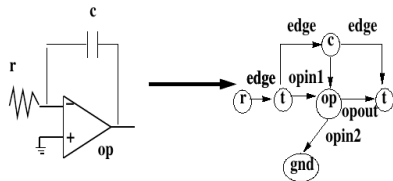
Source: Cook and Holder 1994



Source: M. Grindley et al. 1993

circuit

graph representation



Source: Djoko, Cook and Holder 1997

Outline

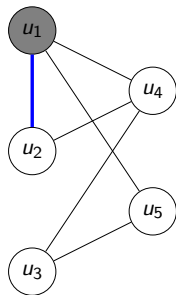
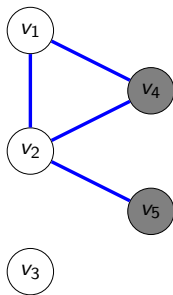
- 1 The Problem
- 2 Algorithms**
- 3 Understanding the Data
- 4 Algorithm Selection
- 5 Observations and Insights
- 6 Switching Algorithms Mid-Execution

Algorithms

- MCSPLIT, MCSPLIT↓
 - McCreesh, Prosser and Trimble 2017
- clique encoding
 - McCreesh, Ndiaye et al. 2016
- k ↓
 - Hoffmann, McCreesh and Reilly 2017

McSPIT: a Branch and Bound Algorithm

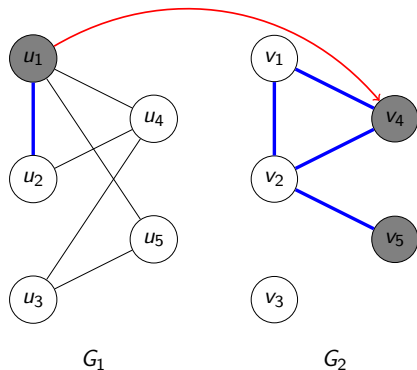
Partial solution:
Upper bound: 4

 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

McSPPLIT: a Branch and Bound Algorithm

Partial solution:
Upper bound: 4

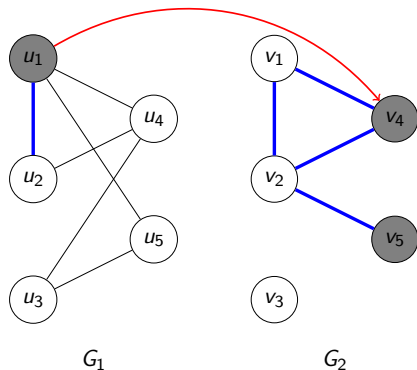


Label	G_1	G_2
0	u_2, u_3, u_4, u_5	v_1, v_2, v_3
1	u_1	v_4, v_5

Decision: $u_1 \mapsto v_4$

McSPIT: a Branch and Bound Algorithm

Partial solution:
Upper bound: 4

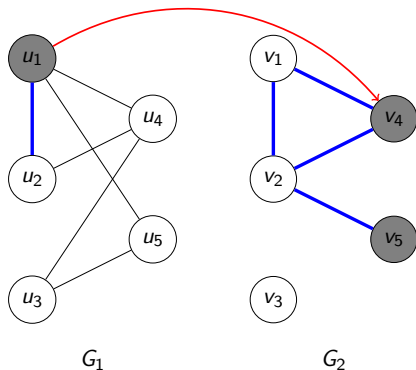


Label	G_1	G_2
00	u_3	v_3
01	u_4, u_5	\emptyset
02	u_2	v_1, v_2
10	\emptyset	v_5

McSPPLIT: a Branch and Bound Algorithm

Partial solution: $u_1 \mapsto v_4$

Upper bound: $1 + 2$

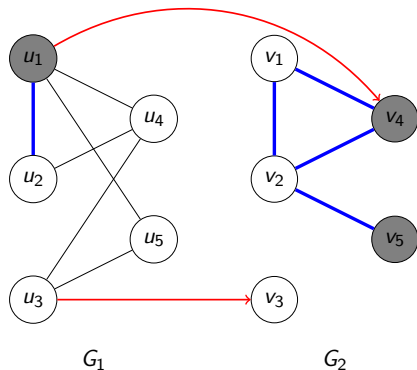


Label	G_1	G_2
00	u_3	v_3
01	u_2	v_1, v_2

MCSPPLIT: a Branch and Bound Algorithm

Partial solution: $u_1 \mapsto v_4$

Upper bound: $1 + 2$



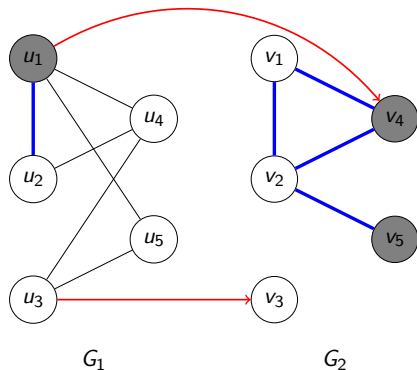
Label	G_1	G_2
00	u_3	v_3
01	u_2	v_1, v_2

Decision: $u_3 \mapsto v_3$

McSPPLIT: a Branch and Bound Algorithm

Partial solution: $u_1 \mapsto v_4$

Upper bound: $1 + 2$

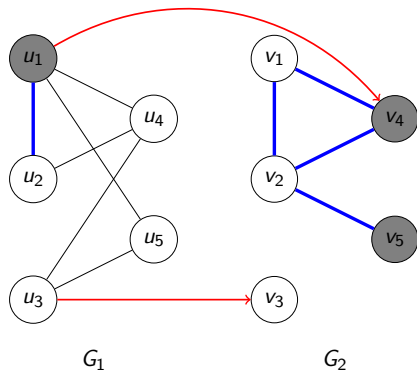


Label	G_1	G_2
010	u_2	v_1, v_2
011	u_4, u_5	\emptyset

McSPPLIT: a Branch and Bound Algorithm

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

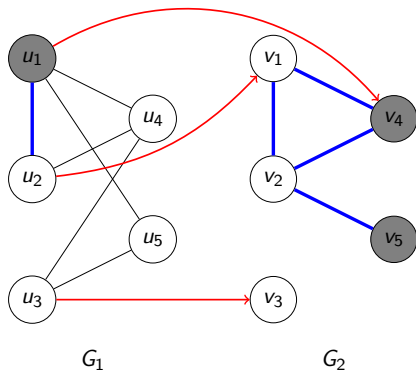
Upper bound: $2 + 1$



Label	G_1	G_2
010	u_2	v_1, v_2

McSPPLIT: 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

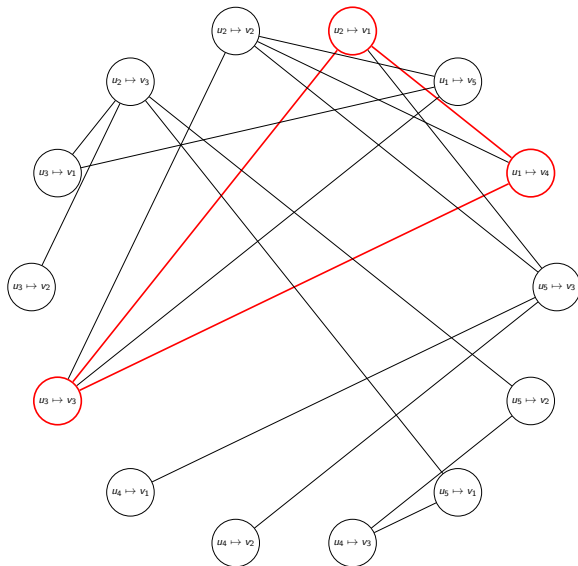
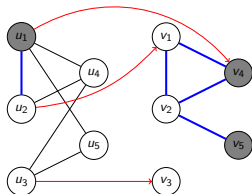
$k \downarrow$

- $k = 0$: 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

McSP_{LIT}↓

- The main idea of $k\downarrow$ applied to McSP_{LIT}
- 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

- 1 The Problem
- 2 Algorithms
- 3 Understanding the Data**
- 4 Algorithm Selection
- 5 Observations and Insights
- 6 Switching Algorithms Mid-Execution

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 $\mu: V \rightarrow \{0, \dots, N - 1\}$ 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 $\mu: V \rightarrow \{0, \dots, N - 1\}$ 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

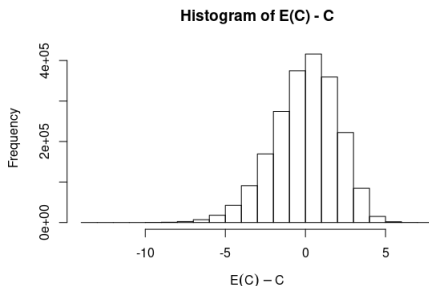
Definition

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

Outline

- 1 The Problem
- 2 Algorithms
- 3 Understanding the Data
- 4 Algorithm Selection**
- 5 Observations and Insights
- 6 Switching Algorithms Mid-Execution

(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 $m: \mathcal{I} \times \mathcal{A} \rightarrow \mathbb{R}$, the *algorithm selection problem* is to find a mapping $s: \mathcal{I} \rightarrow \mathcal{A}$ 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 $m: \mathcal{I} \times \mathcal{A} \rightarrow \mathbb{R}$, the *algorithm selection problem* is to find a mapping $s: \mathcal{I} \rightarrow \mathcal{A}$ that optimises $\mathbb{E}[m(i, s(i))]$.

(G_1, G_2)

(Per-Instance) Algorithm Selection

Definition (Bischl et al. 2016)

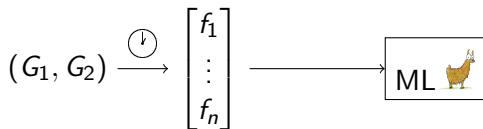
Given a set \mathcal{I} of problem instances, a space of algorithms \mathcal{A} , and a performance measure $m: \mathcal{I} \times \mathcal{A} \rightarrow \mathbb{R}$, the *algorithm selection problem* is to find a mapping $s: \mathcal{I} \rightarrow \mathcal{A}$ that optimises $\mathbb{E}[m(i, s(i))]$.

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

(Per-Instance) Algorithm Selection

Definition (Bischl et al. 2016)

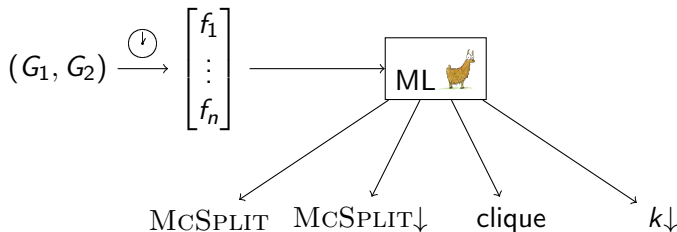
Given a set \mathcal{I} of problem instances, a space of algorithms \mathcal{A} , and a performance measure $m: \mathcal{I} \times \mathcal{A} \rightarrow \mathbb{R}$, the *algorithm selection problem* is to find a mapping $s: \mathcal{I} \rightarrow \mathcal{A}$ that optimises $\mathbb{E}[m(i, s(i))]$.



(Per-Instance) Algorithm Selection

Definition (Bischl et al. 2016)

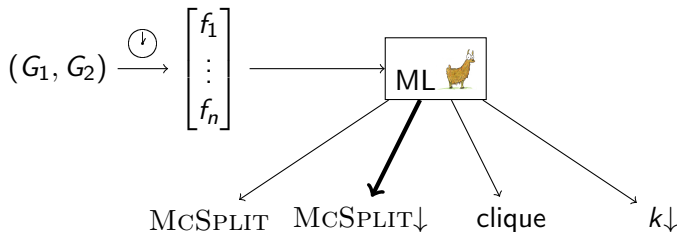
Given a set \mathcal{I} of problem instances, a space of algorithms \mathcal{A} , and a performance measure $m: \mathcal{I} \times \mathcal{A} \rightarrow \mathbb{R}$, the *algorithm selection problem* is to find a mapping $s: \mathcal{I} \rightarrow \mathcal{A}$ that optimises $\mathbb{E}[m(i, s(i))]$.



(Per-Instance) Algorithm Selection

Definition (Bischl et al. 2016)

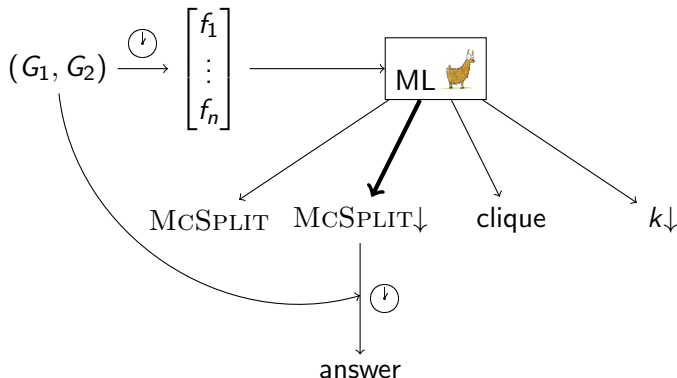
Given a set \mathcal{I} of problem instances, a space of algorithms \mathcal{A} , and a performance measure $m: \mathcal{I} \times \mathcal{A} \rightarrow \mathbb{R}$, the *algorithm selection problem* is to find a mapping $s: \mathcal{I} \rightarrow \mathcal{A}$ that optimises $\mathbb{E}[m(i, s(i))]$.



(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 $m: \mathcal{I} \times \mathcal{A} \rightarrow \mathbb{R}$, the *algorithm selection problem* is to find a mapping $s: \mathcal{I} \rightarrow \mathcal{A}$ that optimises $\mathbb{E}[m(i, s(i))]$.



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

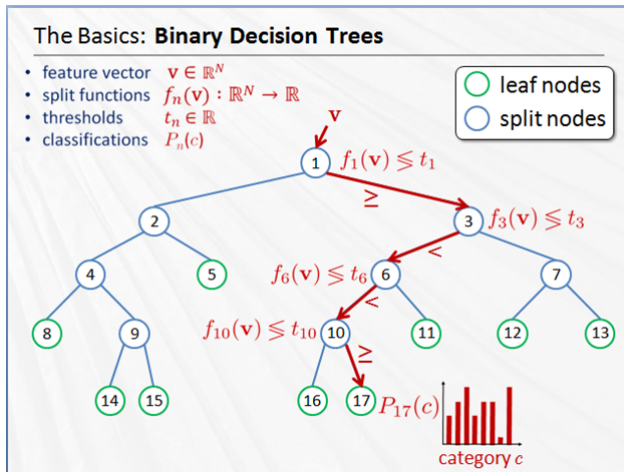
- ① number of vertices
- ② number of edges
- ③ mean/max degree
- ④ density
- ⑤ mean/max distance between pairs of vertices
- ⑥ number of loops
- ⑦ proportion of vertex pairs with distance $\geq 2, 3, 4$
- ⑧ connectedness
- ⑨ standard deviation of degrees
- ⑩ labelling percentage

Features (34 in total)

1–8 are from Kotthoff, McCreesh and Solnon 2016

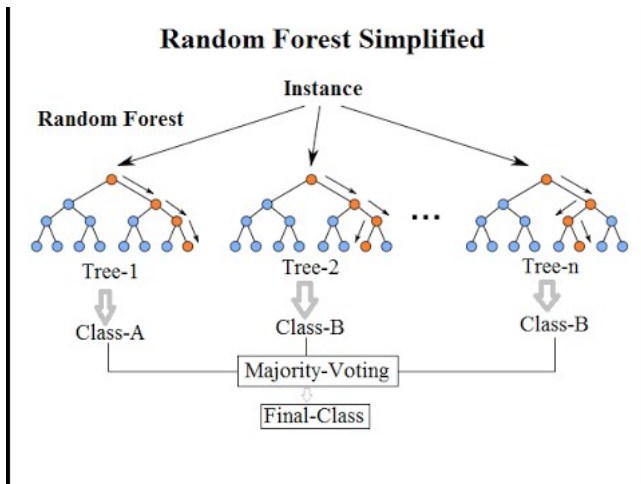
- ① number of vertices
- ② number of edges
- ③ mean/max degree
- ④ density
- ⑤ mean/max distance between pairs of vertices
- ⑥ number of loops
- ⑦ proportion of vertex pairs with distance $\geq 2, 3, 4$
- ⑧ connectedness
- ⑨ standard deviation of degrees
- ⑩ labelling percentage
- ⑪ ratios of features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

Random Forests (Breiman 2001)



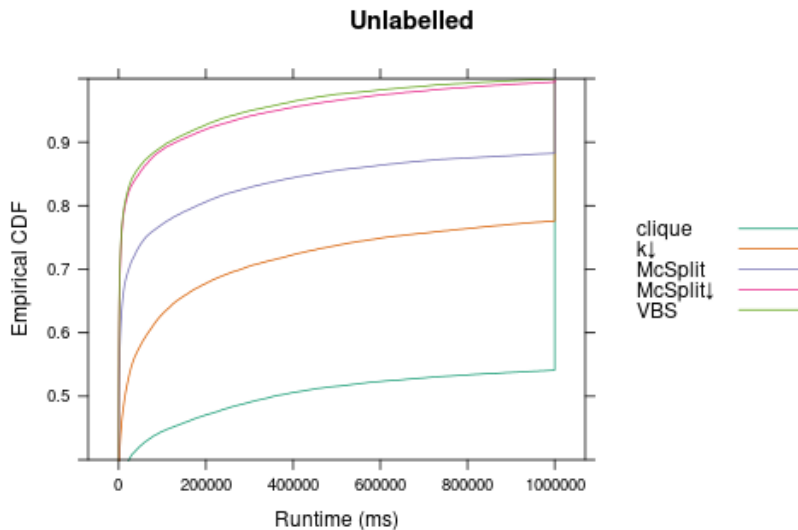
Source: Tae-Kyun Kim & Bjorn Stenger, Intelligent Systems and Networks (ISN) Research Group, Imperial College London

Random Forests (Breiman 2001)

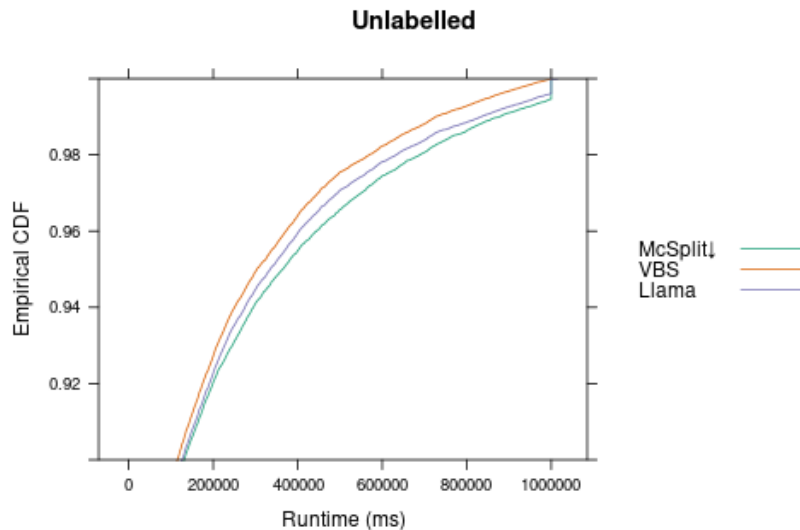


Source: Random Forests(r), Explained, Ilan Reinstein, KDnuggets

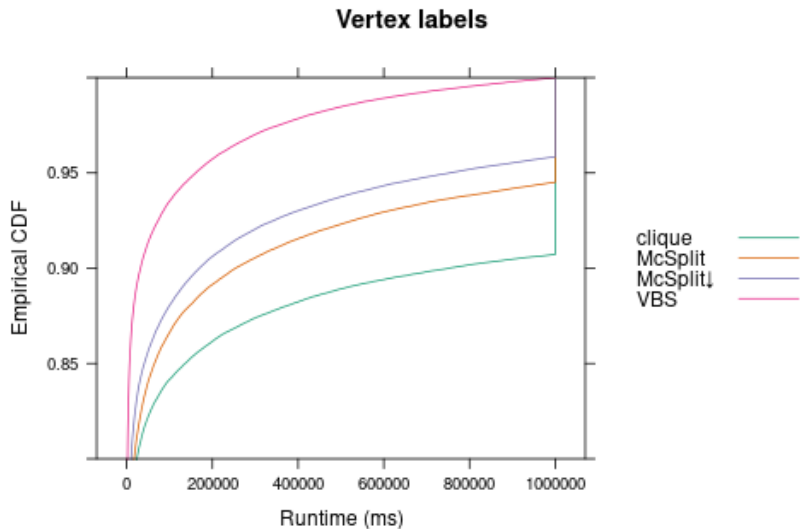
Results



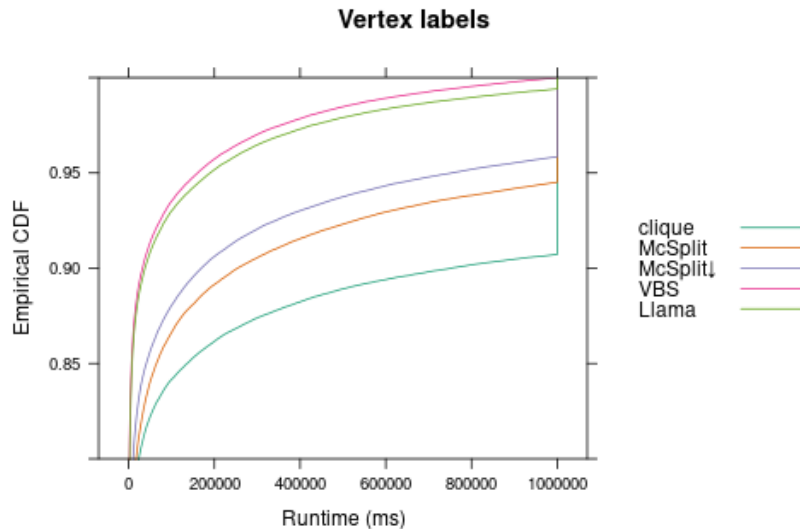
Results (27%)



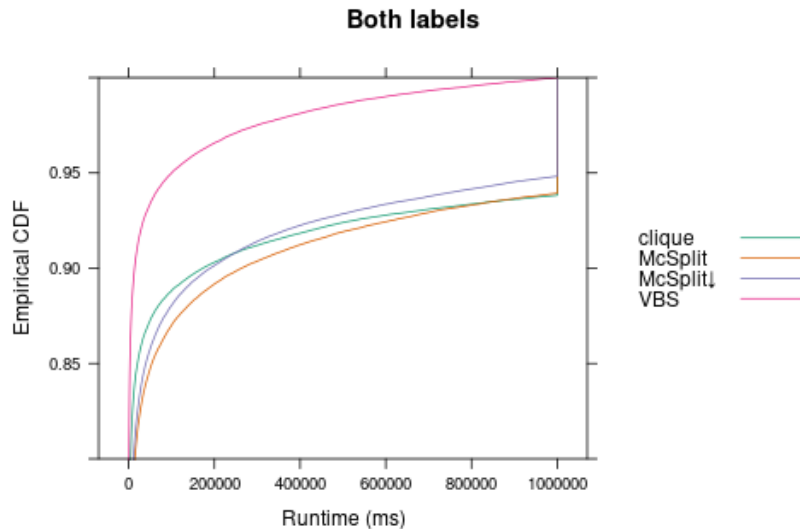
Results



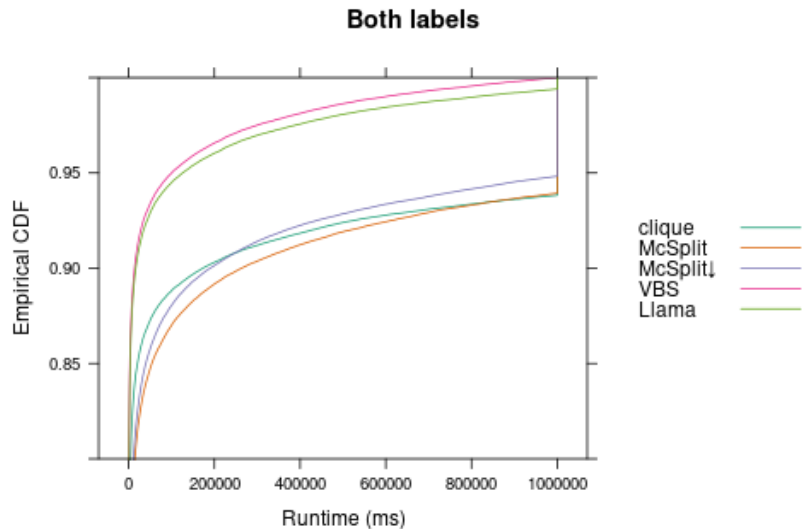
Results (86%)



Results



Results (88%)



Errors

- Out-of-bag error
- For each algorithm
 - $1 - \text{recall}$

Definition

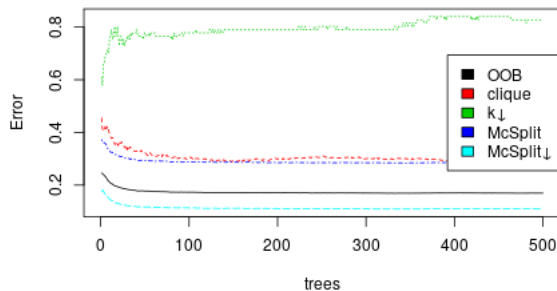
For an algorithm A , *recall* (sensitivity) is

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

Errors (%)

Error	Labelling		
	no	vertex	both
out-of-bag	17	13	14
clique	30	8	7
McSP _{LIT}	29	22	29
McSP _{LIT} ↓	11	11	11
k ↓	80		

Convergence of Errors for Unlabelled Graphs



Outline

- 1 The Problem
- 2 Algorithms
- 3 Understanding the Data
- 4 Algorithm Selection
- 5 Observations and Insights**
- 6 Switching Algorithms Mid-Execution

Overview

- Most important features:
 - labelling percentage
 - standard deviation of degrees (for both graphs)
- Looking at a single feature is not enough

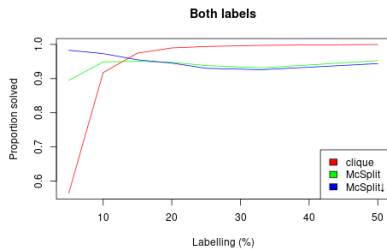
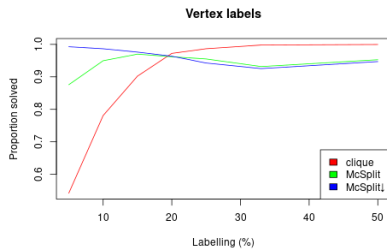
Overview

- Most important features:
 - labelling percentage
 - standard deviation of degrees (for both graphs)
- Looking at a single feature is not enough
- k_{\downarrow} is no longer competitive
- Unclear when MCSPLIT outperforms MCSPLIT $_{\downarrow}$

Overview

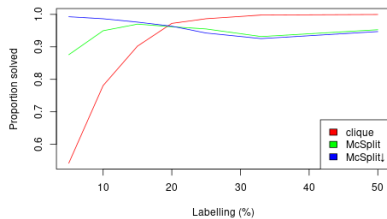
- Most important features:
 - labelling percentage
 - standard deviation of degrees (for both graphs)
- Looking at a single feature is not enough
- k_{\downarrow} is no longer competitive
- Unclear when MCSPLIT outperforms MCSPLIT $_{\downarrow}$
- Benchmarking algorithms with just a few different labels is important
- No work has been done on non-uniform distributions of labels

What Happens When Labelling Changes?

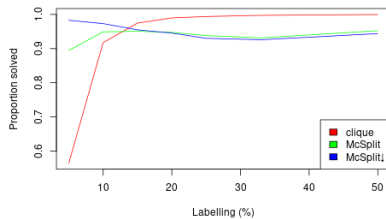


What Happens When Labelling Changes?

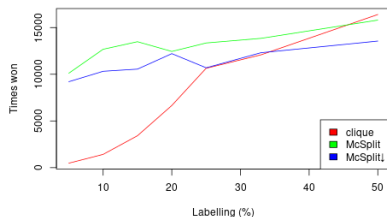
Vertex labels



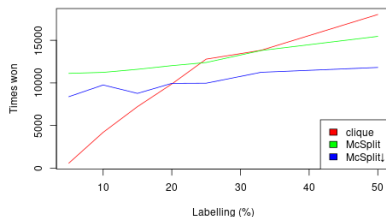
Both labels



Vertex labels



Both labels



Margins

Definition

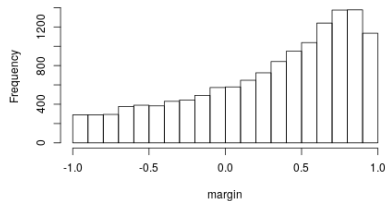
Let c_1, \dots, c_n be n **classes** and let p be a **data point** that belongs to class c_p . Let v_1, \dots, v_n denote the **number of votes** for each class when given p as input. The **margin** of p is

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

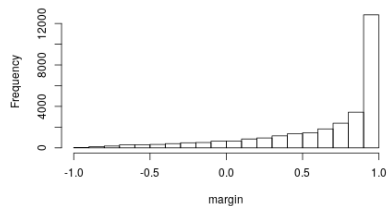
which is a number in $[-1, 1]$.

Margins: Unlabelled Graphs

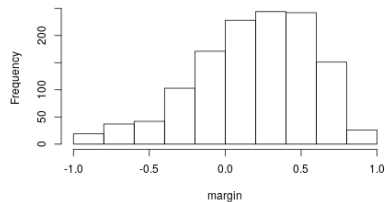
McSplit



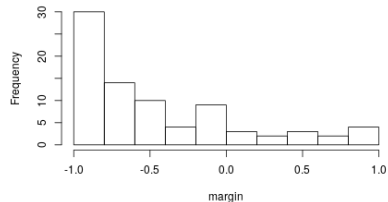
McSplit↓



clique

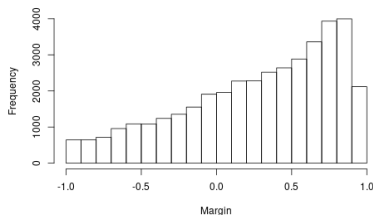


k↓

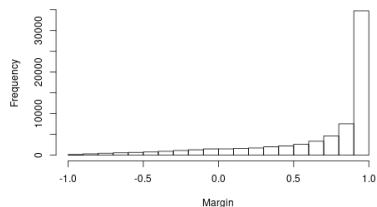


Margins: Vertex and Edge Labels

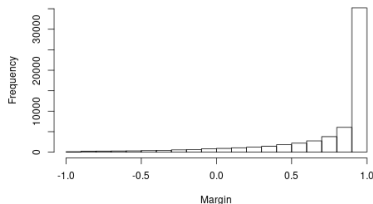
Both labels, McSplit



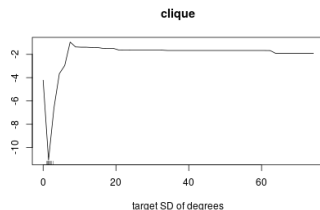
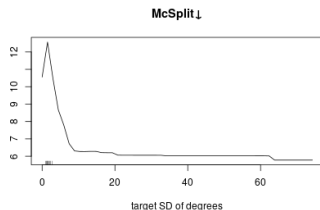
Both labels, McSplit



Both labels, clique



Partial Dependence: Unlabelled Graphs

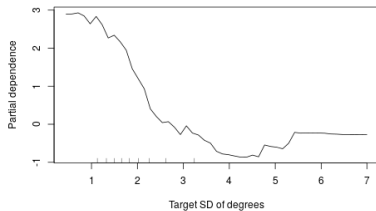


Defined as

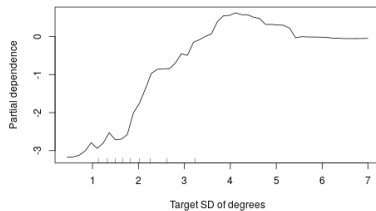
$$f(x) = \log p_k(x) - \frac{1}{K} \sum_{i=1}^K \log p_i(x)$$

- $p_i(x)$ is the proportion of votes for class i
- K is the number of different classes
- k is the class under consideration

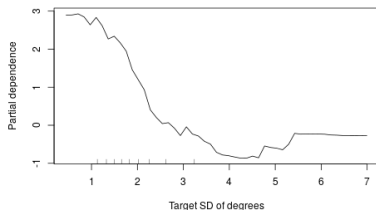
Partial Dependence: Vertex and Edge Labels

Both labels, McSplit \downarrow 

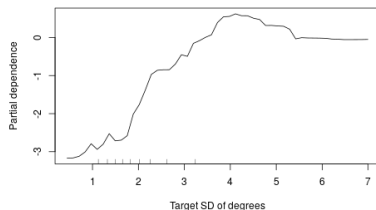
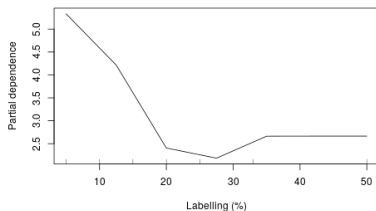
Both labels, clique



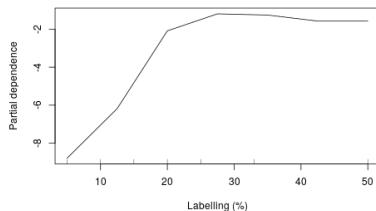
Partial Dependence: Vertex and Edge Labels

Both labels, McSplit \downarrow 

Both labels, clique

Both labels, McSplit \downarrow 

Both labels, clique

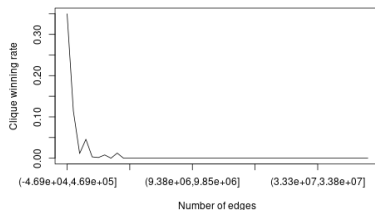
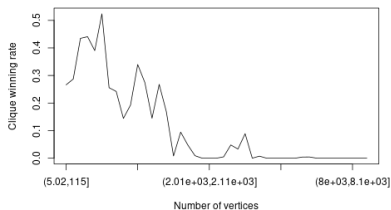
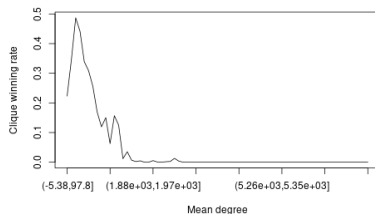
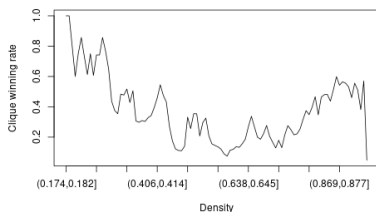


Outline

- 1 The Problem
- 2 Algorithms
- 3 Understanding the Data
- 4 Algorithm Selection
- 5 Observations and Insights
- 6 Switching Algorithms Mid-Execution**

Some Preliminary Analysis

Some Preliminary Analysis



Idea 1: Switch After a Fixed Number of Decisions

Idea 1: Switch After a Fixed Number of Decisions

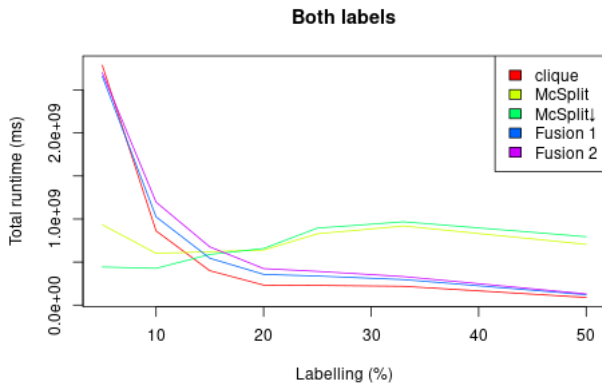
- Vertices of the association graph can be constructed from MCSPLIT label classes, edges from the original input graphs
- Only a few extra lines of code:

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

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?

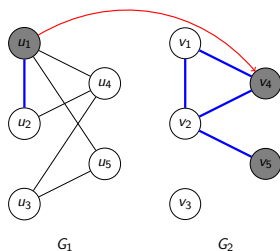
Idea 2: From Partially Solved to Unsolved Instances

- 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

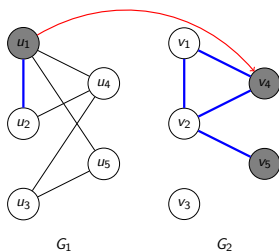


Label	G_1	G_2
00	u_3	v_3
01	u_2	v_1, v_2

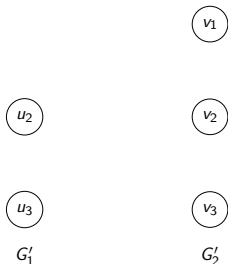
Partial solution and incumbent:

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

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

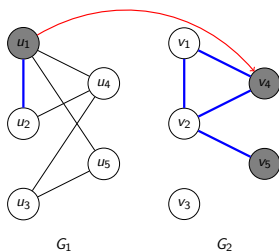


Step 1: Add vertices from label classes

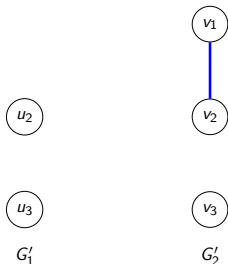
Partial solution and incumbent:

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

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

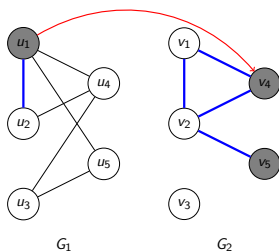


Step 2: $E' = E \cap (V'_1 \times V'_1)$
(preserving edge labels)

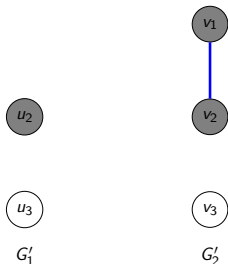
Partial solution and incumbent:

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

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

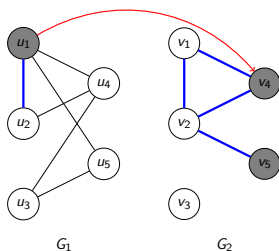


Step 3: label vertices according to vertex classes

Partial solution and incumbent:

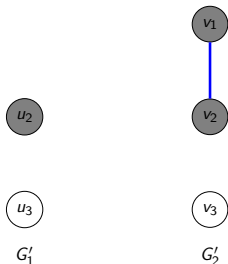
$$\text{incumbent} = M = \{u_1 \mapsto v_4\}$$

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!

Dissertation and code available at

<https://github.com/dilkas/maximum-common-subgraph>