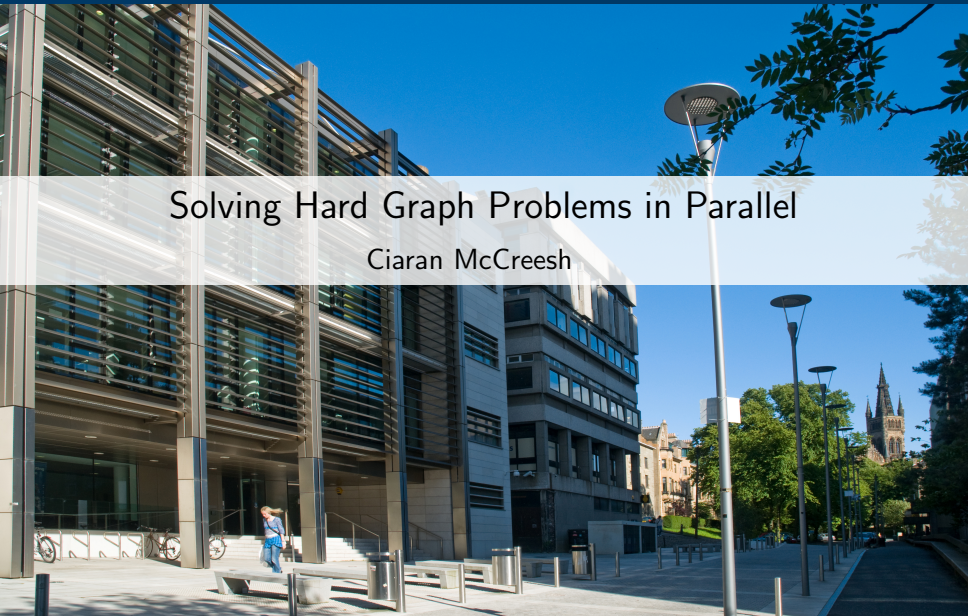


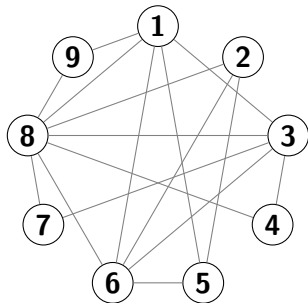
Solving Hard Graph Problems in Parallel

Ciaran McCreesh

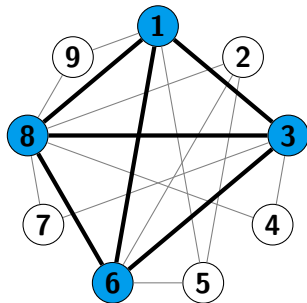


Clique Problems

The Maximum Clique Problem



The Maximum Clique Problem



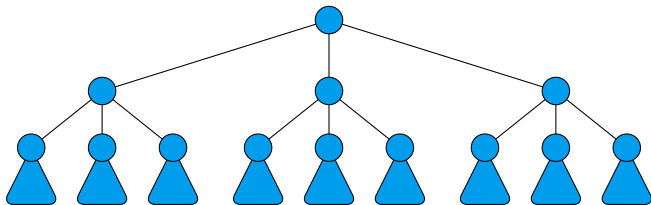
Algorithms

- Branch and bound algorithms, using greedy graph colouring as a bound and ordering heuristic.
- A parallel branch and bound algorithm, with an explanation of why it works.
- An explanation of why the colour ordering process works.

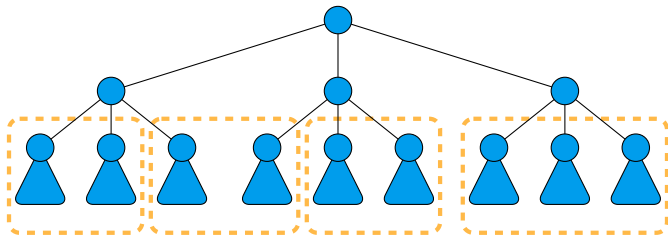
Maximum Clique Variants

- Balanced induced biclique. A branch and bound algorithm (can be parallelised), and the first study.
- Maximum labelled clique. Two-pass parallel branch and bound algorithm, which closed all open problem instances from the literature.

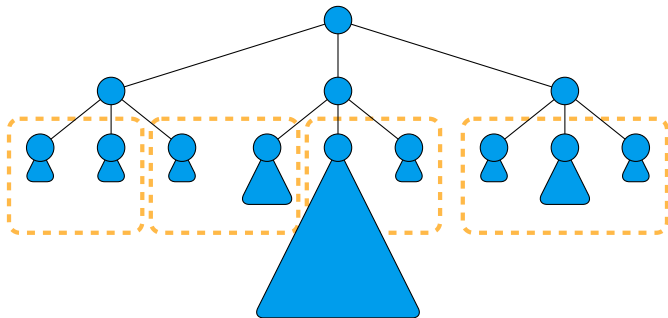
Search as a Tree



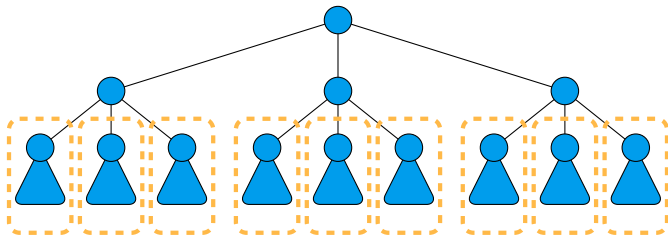
Parallel Search



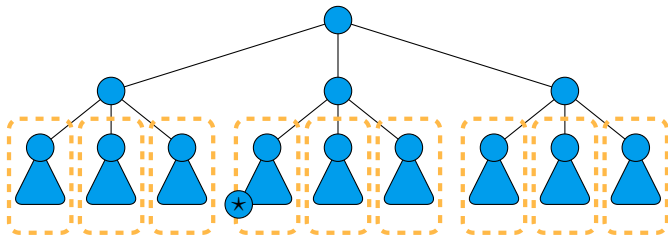
Parallel Search



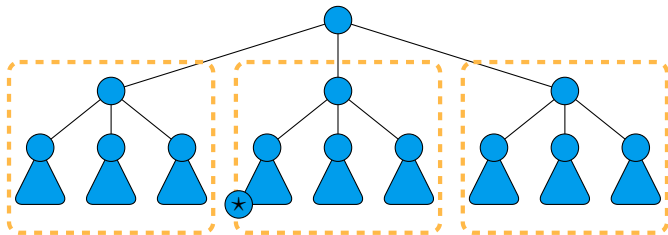
Parallel Search



Work-Stealing is Not Just About Balance



Work-Stealing is Not Just About Balance

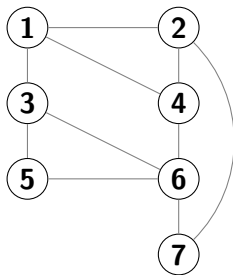
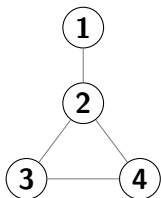


Preventing a Slowdown, Part 1

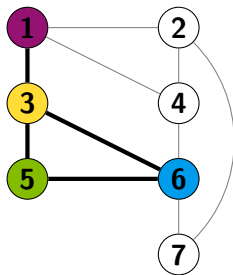
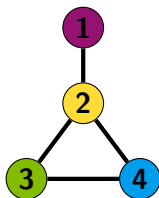
- A subset of the sequential order must be preserved.
- Bounds must be communicated.

Subgraph Problems

The Subgraph Isomorphism Problem



The Subgraph Isomorphism Problem



Algorithms

- A new backjumping search algorithm which substantially outperforms the competition.
- New, cheaper filtering for all-different constraints.
- Supplemental graphs, to generate implied constraints.
- A different way of doing backjumping, not using conflict sets.
- Improved conflict analysis for failed all-different constraints.
- Parallel backjumping as a lazy fold with right zero elements.

A Backjumping Algorithm

```

1 search (Domains  $D$ )  $\rightarrow$  Fail  $F$  or Success
2 begin
3   if  $D = \emptyset$  then return Success
4    $D_v \leftarrow$  a domain in  $D$  with minimum size
5    $F \leftarrow \{v\}$ 
6   foreach  $v' \in D_v$  ordered by a heuristic do
7      $D' \leftarrow \text{clone}(D)$ 
8     case assign( $D', v, v'$ ) of
9       Fail  $F'$  then  $F \leftarrow F \cup F'$ 
10      Success then
11        case search( $D' - D_v$ ) of
12          Fail  $F'$  then
13            if  $\nexists w \in F'$  such that  $D_w \neq D'_w$  then return Fail  $F'$ 
14             $F \leftarrow F \cup F'$ 
15          Success then return Success
16   return Fail  $F$ 

```

A Backjumping Algorithm

```

1 search (Domains  $D$ )  $\rightarrow$  Fail  $F$  or Success
2 begin
3   if  $D = \emptyset$  then return Success
4    $D_v \leftarrow$  a domain in  $D$  with minimum size
5    $F \leftarrow \{v\}$ 
6   foreach  $v' \in D_v$  ordered by a heuristic do
7      $D' \leftarrow \text{clone}(D)$ 
8     case assign( $D', v, v'$ ) of
9       Fail  $F'$  then  $F \leftarrow F \cup F'$ 
10      Success then
11        case search( $D' - D_v$ ) of
12          Fail  $F'$  then
13            if  $\nexists w \in F'$  such that  $D_w \neq D'_w$  then return Fail  $F'$ 
14             $F \leftarrow F \cup F'$ 
15          Success then return Success
16   return Fail  $F$ 

```

A Backjumping Algorithm

```

1 search (Domains  $D$ ) → Fail  $F$  or Success
2 begin
3   if  $D = \emptyset$  then return Success
4    $D_v \leftarrow$  a domain in  $D$  with minimum size
5    $F \leftarrow \{v\}$ 
6   foreach  $v' \in D_v$  ordered by a heuristic do
7      $D' \leftarrow \text{clone}(D)$ 
8     case assign( $D', v, v'$ ) of
9       Fail  $F'$  then  $F \leftarrow F \cup F'$ 
10      Success then
11        case search( $D' - D_v$ ) of
12          Fail  $F'$  then
13            if  $\nexists w \in F'$  such that  $D_w \neq D'_w$  then return Fail  $F'$ 
14             $F \leftarrow F \cup F'$ 
15          Success then return Success
16   return Fail  $F$ 

```

A Backjumping Algorithm

```

1 search (Domains  $D$ ) → Fail  $F$  or Success
2 begin
3   if  $D = \emptyset$  then return Success
4    $D_v \leftarrow$  a domain in  $D$  with minimum size
5    $F \leftarrow \{v\}$ 
6   foreach  $v' \in D_v$  ordered by a heuristic do
7      $D' \leftarrow \text{clone}(D)$ 
8     case assign( $D', v, v'$ ) of
9       Fail  $F'$  then  $F \leftarrow F \cup F'$ 
10      Success then
11        case search( $D' - D_v$ ) of
12          Fail  $F'$  then
13            if  $\nexists w \in F'$  such that  $D_w \neq D'_w$  then return Fail  $F'$ 
14             $F \leftarrow F \cup F'$ 
15          Success then return Success
16   return Fail  $F$ 

```

A Backjumping Algorithm

```

1 search (Domains  $D$ ) → Fail  $F$  or Success
2 begin
3   if  $D = \emptyset$  then return Success
4    $D_v \leftarrow$  a domain in  $D$  with minimum size
5    $F \leftarrow \{v\}$ 
6   foreach  $v' \in D_v$  ordered by a heuristic do
7      $D' \leftarrow \text{clone}(D)$ 
8     case assign( $D', v, v'$ ) of
9       Fail  $F'$  then  $F \leftarrow F \cup F'$ 
10      Success then
11        case search( $D' - D_v$ ) of
12          Fail  $F'$  then
13            if  $\nexists w \in F'$  such that  $D_w \neq D'_w$  then return Fail  $F'$ 
14             $F \leftarrow F \cup F'$ 
15          Success then return Success
16   return Fail  $F$ 

```

A Backjumping Algorithm

```

1 search (Domains  $D$ ) → Fail  $F$  or Success
2 begin
3   if  $D = \emptyset$  then return Success
4    $D_v \leftarrow$  a domain in  $D$  with minimum size
5    $F \leftarrow \{v\}$ 
6   foreach  $v' \in D_v$  ordered by a heuristic do
7      $D' \leftarrow \text{clone}(D)$ 
8     case assign( $D', v, v'$ ) of
9       Fail  $F'$  then  $F \leftarrow F \cup F'$ 
10      Success then
11        case search( $D' - D_v$ ) of
12          Fail  $F'$  then
13            if  $\exists w \in F'$  such that  $D_w \neq D'_w$  then return Fail  $F'$ 
14             $F \leftarrow F \cup F'$ 
15          Success then return Success
16   return Fail  $F$ 

```

The diagram illustrates a backjumping algorithm. It starts with a search function that returns either a failure state F or success. The algorithm begins by checking if the domain D is empty. If not, it selects a domain D_v with the minimum size and initializes a failure set F with the variable v . It then iterates over variables v' in D_v ordered by a heuristic. For each v' , it clones the domain D to D' and attempts to assign v' to D' . If this assignment fails (Fail F'), the failure set F is updated to include F' . If the assignment is successful, it recursively searches the remaining domain $D' - D_v$. If this recursive search fails (Fail F'), it checks if there exists a variable w in F' such that $D_w \neq D'_w$. If such a w exists, it returns Fail F' , which is highlighted in blue. A blue arrow points from this return statement back to the failure state F in the outer loop, indicating a backjump. If the recursive search is successful, it returns Success. Finally, if all variables in D_v are processed, it returns Fail F .

Backjumping as a Lazy Fold

- Lazily map each subproblem to Jump F **or** Fail F **or** Success.
- Lazily fold, starting with Fail $\{v\}$, as follows:

$$\begin{aligned}
 _ \otimes \text{Success} &= \text{Success} \\
 _ \otimes \text{Jump } F &= \text{Jump } F \\
 \text{Fail } F \otimes \text{Fail } G &= \text{Fail } (F \cup G)
 \end{aligned}$$

- Convert Jump F back to Fail F for the result.
- If a Jump F occurs to the left of a Success, we have a bug.

Folding Zero

- When multiplying, if any item is 0, the result is 0.

$$\begin{aligned} _ \times 0 &= 0 \\ 0 \times _ &= 0 \end{aligned}$$

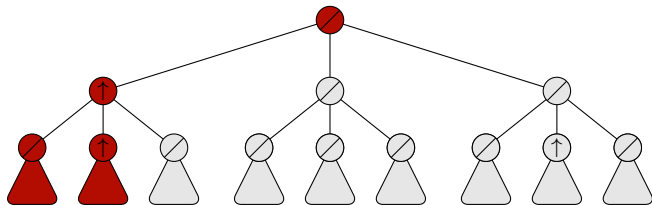
- Here, if any item is Success, the result is Success, and we do not need to evaluate the rest of the map.

$$_ \otimes \text{Success} = \text{Success}$$

- If any item is Jump F , the result is either Jump F , or some Jump G or Success that is further to the left. We do not need to evaluate any item to the right.

$$_ \otimes \text{Jump } F = \text{Jump } F$$

Lazy Fold Backjumping as a Tree



Preventing a Slowdown, Part 2

- Any subproblem which we have shown will not be used, must be cancelled (recursively) immediately.
- When the result of a fold is known, the continuation must be executed immediately.

Subgraph Isomorphism Variants

- Induced and non-induced.
- Directed edges.
- Labelled vertices and edges.

The Maximum Common Subgraph Problem

- Can be solved via the maximum clique problem.
- The greedy colouring technique sometimes misbehaves badly, though...

Colouring Problems

The Graph Colouring Problem

- There are branch-and-bound algorithms for graph colouring, which should be easy to parallelise.
- Clause-learning works very well for some problem instances. Can backjumping be an alternative?
 - Backjumping plus optimisation is a bit hairy. . .

Graph Colouring Variants

- Lots of theoretical results, but less in the way of real-world problem instances.

`http://dcs.gla.ac.uk/~ciaran
c.mccreesh.1@research.gla.ac.uk`