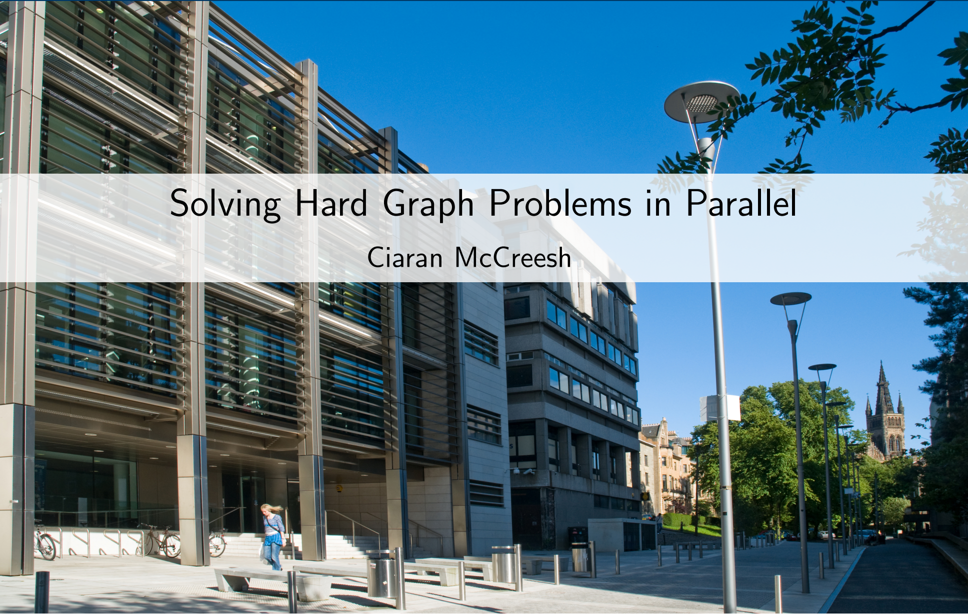
University of Glasgow

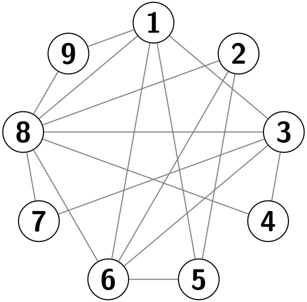
Solving Hard Graph Problems in Parallel

Ciaran McCreesh

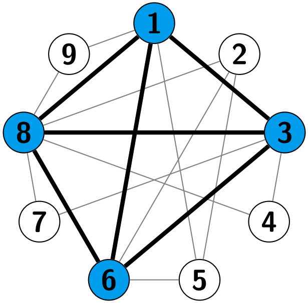


Clique Problems

The Maximum Clique Problem



The Maximum Clique Problem



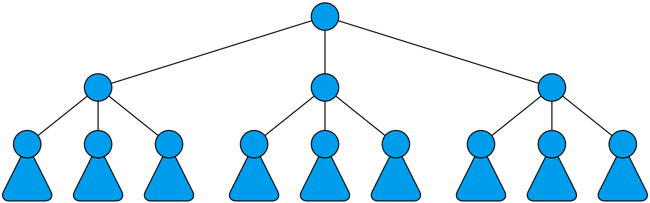
Algorithms

- 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.

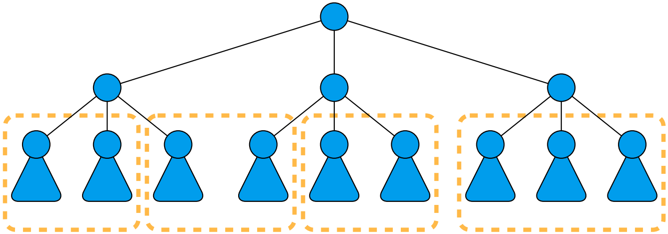
Maximum Clique Variants

- 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.

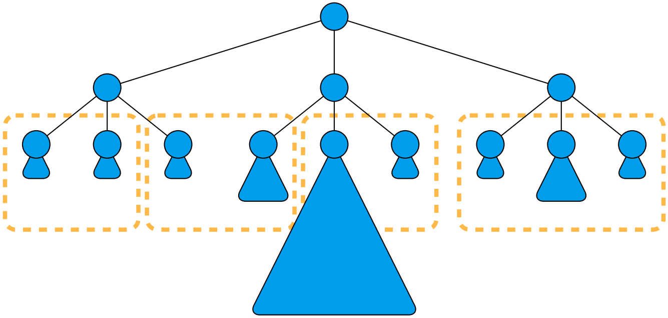
Search as a Tree



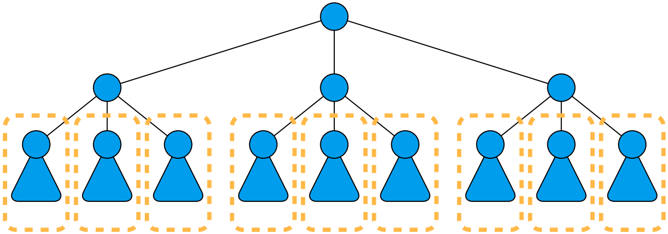
Parallel Search



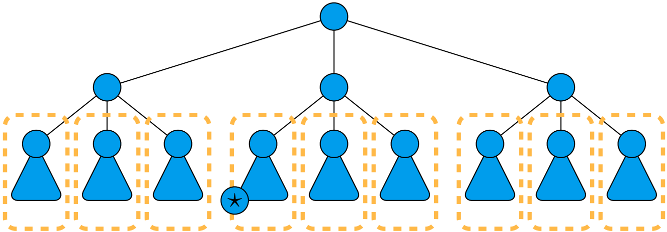
Parallel Search



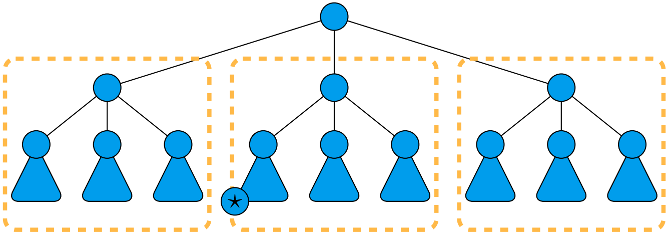
Parallel Search



Work-Stealing is Not Just About Balance



Work-Stealing is Not Just About Balance

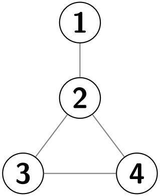


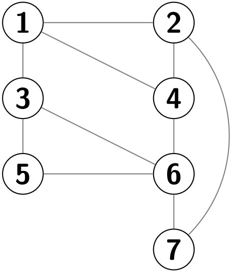
Preventing a Slowdown, Part 1

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

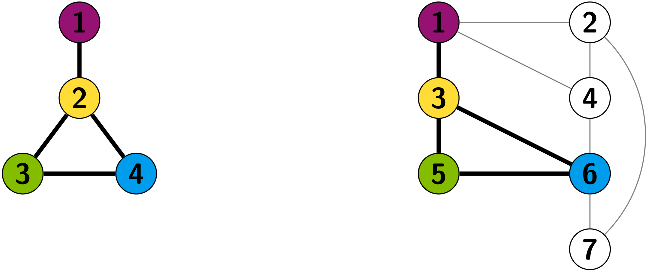
Subgraph Problems

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

```
search (Domains D) → Fail F or Success
begin
    if D = ∅ then return Success
    D_v ← a domain in D with minimum size
    F ← {v}
    foreach v′ ∈ D_v ordered by a heuristic do
        D′ ← clone(D)
        case assign(D′, v, v′) of
            Fail F′ then F ← F ∪ F′
            Success then
                case search(D′ − D_v) of
                    Fail F′ then
                        if ∄ w ∈ F′ such that D_w ≠ D′_w then return Fail F′
                        F ← F ∪ F′
                    Success then return Success
    return Fail F
```

A Backjumping Algorithm

```
search (Domains D) → Fail F or Success
begin
    if D = ∅ then return Success
    D_v ← a domain in D with minimum size
    F ← {v}
    foreach v' ∈ D_v ordered by a heuristic do
        D' ← clone(D)
        case assign(D', v, v') of
            Fail F' then F ← F ∪ F'
            Success then
                case search(D' − D_v) of
                    Fail F' then
                        if ∄ w ∈ F' such that D_w ≠ D'_w then return Fail F'
                        F ← F ∪ F'
                    Success then return Success
    return Fail F
```

A Backjumping Algorithm

```
search (Domains D) → Fail F or Success
begin
    if D = ∅ then return Success
    D_v ← a domain in D with minimum size
    F ← {v}
    foreach v′ ∈ D_v ordered by a heuristic do
        D′ ← clone(D)
        case assign(D′, v, v′) of
            Fail F′ then F ← F ∪ F′
            Success then
                case search(D′ − D_v) of
                    Fail F′ then
                        if ∄ w ∈ F′ such that D_w ≠ D′_w then return Fail F′
                        F ← F ∪ F′
                    Success then return Success
    return Fail F
```

A Backjumping Algorithm

search (Domains D) $\rightarrow$ Fail F **or** Success
begin
  if $D = \emptyset$ **then return** Success
  $D_v \leftarrow$ a domain in D with minimum size
  $F \leftarrow \{v\}$
  foreach $v' \in D_v$ ordered by a heuristic **do**
    $D' \leftarrow$ clone(D)
    case assign(D', v, v') **of**
      Fail F' **then** $F \leftarrow F \cup F'$
      Success **then**
        case search$(D' - D_v)$ **of**
          Fail F' **then**
            if $\nexists w \in F'$ **such that** $D_w \neq D'_w$ **then return** Fail F'
            $F \leftarrow F \cup F'$
          Success **then return** Success
  return Fail F

A Backjumping Algorithm

search (Domains D) $\rightarrow$ Fail F **or** Success
begin
    if $D = \emptyset$ **then return** Success
    $D_v \leftarrow$ a domain in D with minimum size
    $F \leftarrow \{v\}$
    foreach $v' \in D_v$ ordered by a heuristic **do**
        $D' \leftarrow \mathtt{clone}(D)$
        case $\mathtt{assign}(D', v, v')$ **of**
            Fail F' **then** $F \leftarrow F \cup F'$
            Success **then**
                case $\mathtt{search}(D' - D_v)$ **of**
                    Fail F' **then**
                        if $\nexists w \in F'$ **such that** $D_w \neq D'_w$ **then return** Fail F'
                        $F \leftarrow F \cup F'$
                    Success **then return** Success
    return Fail F

A Backjumping Algorithm

```
search (Domains D) → Fail F or Success
begin
    if D = ∅ then return Success
    D_v ← a domain in D with minimum size
    F ← {v}
    foreach v′ ∈ D_v ordered by a heuristic do
        D′ ← clone(D)
        case assign(D′, v, v′) of
            Fail F′ then F ← F ∪ F′
            Success then
                case search(D′ − D_v) of
                    Fail F′ then
                        if ∄ w ∈ F′ such that D_w ≠ D′_w then return Fail F′
                        F ← F ∪ F′
                    Success then return Success
    return 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:

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

- 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.

$$\_\_ \times 0 = 0$$
$$0 \times \_\_ = 0$$

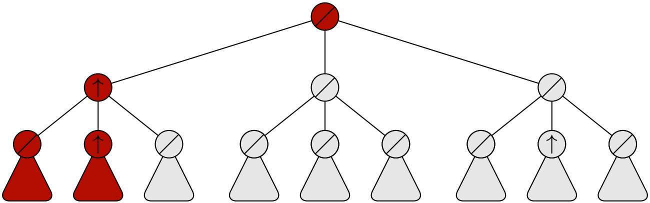
- Here, if any item is Success, the result is Success, and we do not need to evaluate the rest of the map.

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

- 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.

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

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