Scaling Parallel Combinatorial Search

Patrick Maier (Sheffield Hallam)
P.Maier@shu.ac.uk

joint work with **Blair Archibald**, Ciaran McCreesh, and Phil Trinder (Glasgow)

CoDiMa Workshop
Manchester 24 May 2019

Why Exact Combinatorial Search?

- Classical NP-hard problems
  - Travelling salesman
  - Graph colouring
  - Boolean satisfiability
  - ...
- Many applications
  - Puzzles
  - Computational algebra
  - Scheduling
  - Vehicle routing
  - Biochemistry
  - ...
- Active area of research in algorithms
  - **BUT: Hard to parallelise**

			1			7		2
	3		9	5				
		1			2			3
5	9					3		1
	2						7	
7		3					9	8
8			2			1		
				8	5		6	
6		5			9			

Search Types

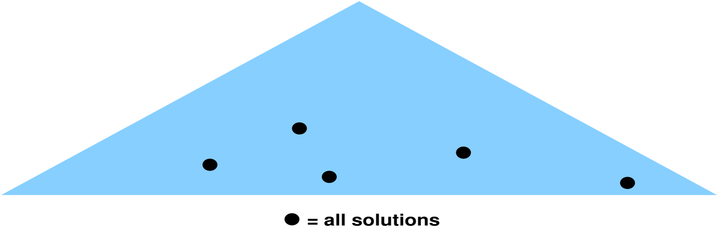
Combinatorial search systematically traverses a search tree by backtracking.

Three types of combinatorial searches:

Search Types

Combinatorial search systematically traverses a search tree by backtracking.

Three types of combinatorial searches: **Enumeration**



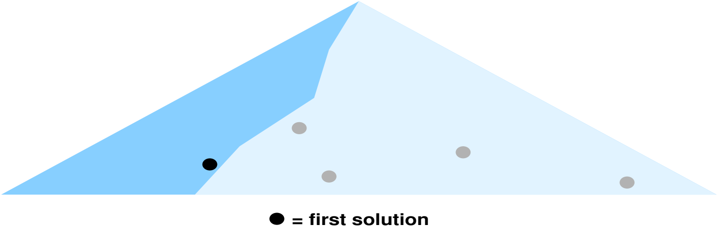
Examples:

Enumeration   Find all k-cliques of the graph

Search Types

Combinatorial search systematically traverses a search tree by backtracking.

Three types of combinatorial searches: **Decision**



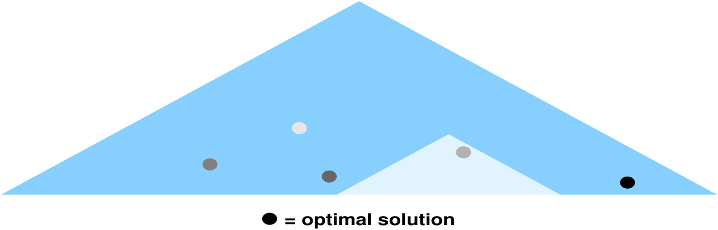
Examples:

Enumeration	Find all k-cliques of the graph
Decision	Does the graph have a k-clique?

Search Types

Combinatorial search systematically traverses a search tree by backtracking.

Three types of combinatorial searches: **Optimisation**



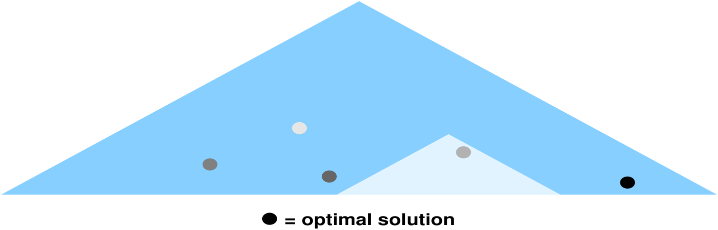
Examples:

Enumeration	Find all k-cliques of the graph
Decision	Does the graph have a k-clique?
Optimisation	Find a maximum clique of the graph

Search Types

Combinatorial search systematically traverses a search tree by backtracking.

Three types of combinatorial searches: **Optimisation**



Examples:

Enumeration	Find all k-cliques of the graph
Decision	Does the graph have a k-clique?
Optimisation	Find a maximum clique of the graph

Completeness property: Provable optimality/infeasibility of solutions

Generic Backtracking Search

Every backtracking search can be expressed by suitably defining

- a search tree (type node),
- a set of facts (type facts) to be gathered during search, and
- three functions generate, learn and prune.

Generic API for combinatorial search (Haskell)

```
class BacktrackingSearch node facts where
  generate :: node -> [node]
  -- construct search tree on demand by expanding current node
  learn :: facts -> node -> facts
  -- add solutions (and non-solutions) to current facts
  prune :: facts -> node -> Bool
  -- skip subtrees that cannot contain solutions

-- generic backtracking search (left-to-right DFS)
search :: (BacktrackingSearch node facts) => facts -> node -> facts
search facts node = if prune facts node
                    then facts
                    else foldl search (learn facts node) (generate node)
```

Outline

1. Generic Combinatorial Search
2. Why Parallel Combinatorial Search Is Challenging
3. A Generic Framework for Parallel Combinatorial Search

Why Parallel Combinatorial Search?

Universal goal: Solve bigger instances!

Better algorithms help.

- But algorithms' progress is unsteady and unpredictable.

Better hardware used to help.

- Up to 2005, CPU speed increased exponentially.
- Since 2005, CPU speed has (almost) stalled.

Parallel HW grows exponentially.

- Moore's law remains intact.

Why Parallel Combinatorial Search?

Universal goal: Solve bigger instances!

Better algorithms help.

- But algorithms' progress is unsteady and unpredictable.

Better hardware used to help.

- Up to 2005, CPU speed increased exponentially.
- Since 2005, CPU speed has (almost) stalled.

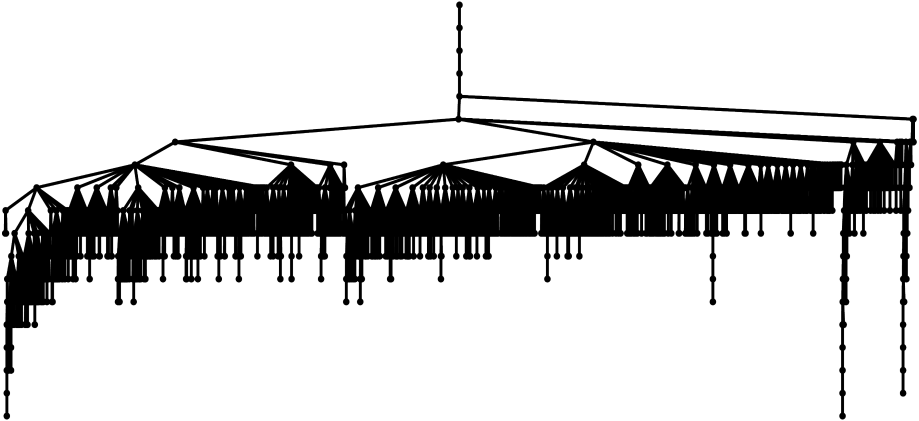
Parallel HW grows exponentially.

- Moore's law remains intact.

Conclusion: Need scalable parallel combinatorial search!

Why is Exact Parallel Combinatorial Search Difficult?

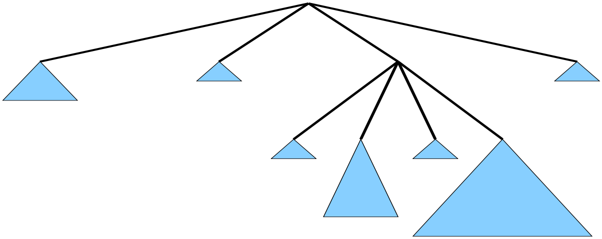
Good News: There is lots of parallelism!



Example: Search tree for a problem in finite geometry.

Why is Exact Parallel Combinatorial Search Difficult?

Problem 1: Search trees are very irregular!

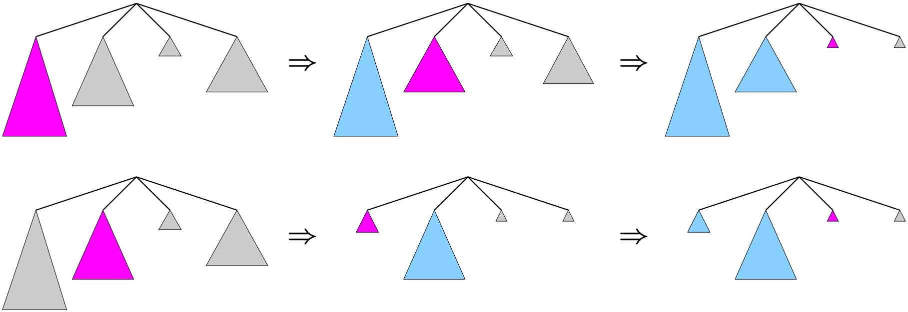


Observation 1: Dynamic scheduling is necessary.

Why is Exact Parallel Combinatorial Search Difficult?

Problem 2: Subtrees are not independent!

- Knowledge sharing and pruning affect the shape of the search tree.



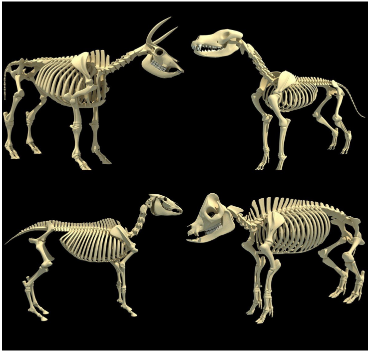
Observation 2: Search order matters.

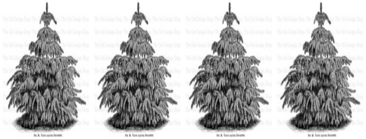
- Random scheduling can lead to very unpredictable performance!

Outline

1. Generic Combinatorial Search
2. Why Parallel Combinatorial Search Is Challenging
3. A Generic Framework for Parallel Combinatorial Search

The YewPar Framework for Parallel Combinatorial Search





YewPar search skeleton library

- Search tree generation and pruning
- Knowledge propagation
- Termination

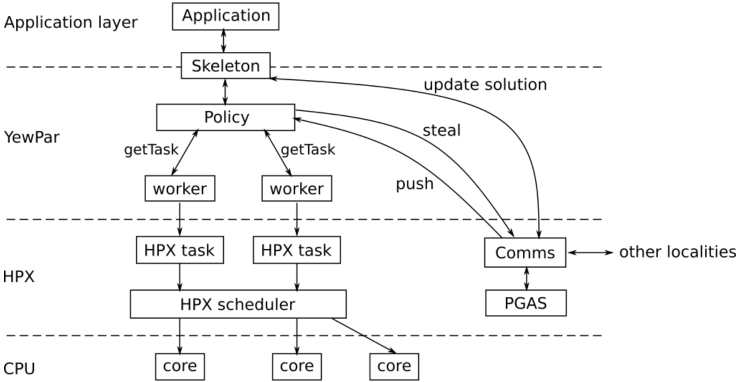
Aside: Algorithmic skeletons are parametric/templated patterns abstracting parallel coordination.

YewPar distributed task-parallel scheduler

- Scheduling policies: ordered, unordered, ...

HPX C++ library (STE||AR Group, LSU)

YewPar Architecture



Key design principles

- Asynchronous distributed work stealing
  - with configurable policies (including policies preserving heuristics search order)
- Asynchronous distributed knowledge propagation

YewPar Skeleton API

Cliques
Numerical Semigroups
...

Search Application = Search Skeleton + Node Generator

Depth-Bounded
Stack-Stealing
Budget
...

Enumeration
Decision
Optimisation

Search Skeleton = Parallel Coordination + Search Type

User builds search application by

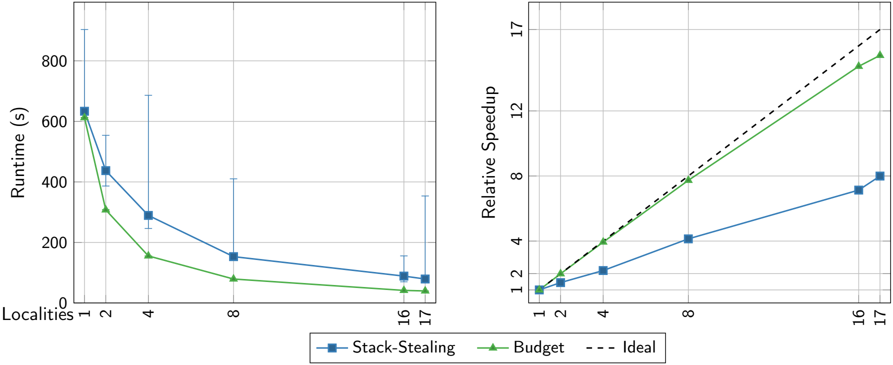
- providing a node generator and
- picking a skeleton from the library (or extending the library).
  - Skeleton determines search type and parallel coordination (work generation and task scheduling policies)

YewPar Experimental Evaluation

Benchmark applications:

Enumeration	Unbalanced Tree Search
	Enumeration of Numerical Semigroups
Decision	Existence of k-cliques
	Subgraph isomorphism
	Existence of spreads in certain finite geometries
Optimisation	Maximum clique
	Travelling salesperson
	0/1 knapsack

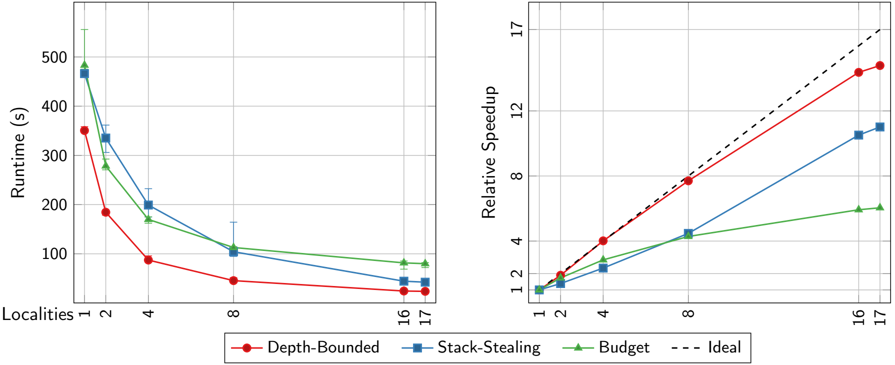
YewPar Scaling: Numerical Semigroups



Counting Numerical Semigroups (of genus 50)

- Scaling from 1 cluster node (15 workers) to 17 nodes (255 workers)
- Budget skeleton scales almost linearly, with little runtime variance
- Stack-Stealing skeleton scales, but with huge runtime variance
- Depth-Bounded skeleton times out

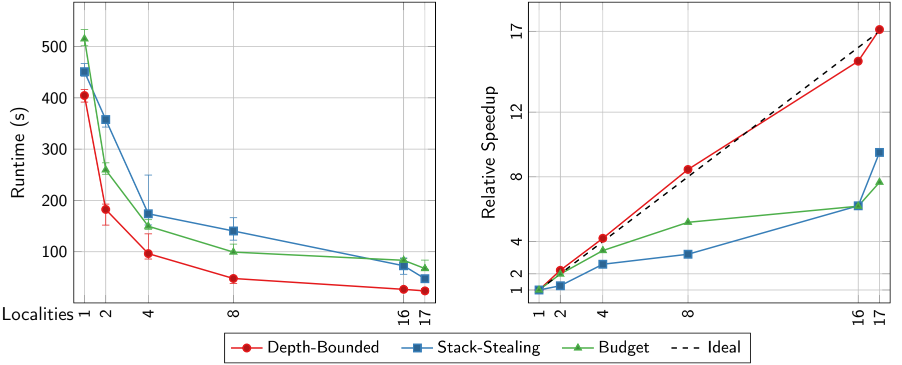
YewPar Scaling: Maximum Clique



Finding a Maximum Clique (DIMACS instance brock800_1)

- Scaling from 1 cluster node (15 workers) to 17 nodes (255 workers)
- Depth-Bounded scales almost linearly
- Stack-Stealing scales
- Budget scales worst

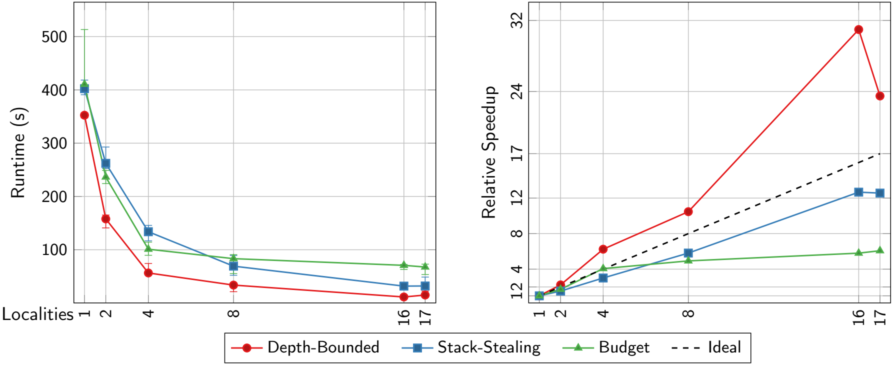
YewPar Scaling: Maximum Clique



Finding a Maximum Clique (DIMACS instance brock800_2)

- Scaling from 1 cluster node (15 workers) to 17 nodes (255 workers)
- Depth-Bounded scales almost linearly (occasionally super-linearly)
- Stack-Stealing and Budget do not scale well

YewPar Scaling: Maximum Clique



Finding a Maximum Clique (DIMACS instance brock800_3)

- Scaling from 1 cluster node (15 workers) to 17 nodes (255 workers)
- Depth-Bounded scales super-linearly
- Stack-Stealing scales
- Budget does not scale well

YewPar Summary

- YewPar is a general purpose framework for exact combinatorial search.
  - High-level parallelism abstractions (skeletons) supporting multiple search types
  - Suitable for parallelising state-of-the-art sequential search algorithms
- YewPar scales.
  - Good scaling on compute clusters (tested up to 17 nodes, 255 cores)
- Scaling depends on the choice of skeleton.
  - No single best skeleton

References:

- B. Archibald (2018). *YewPar*. `https://github.com/BlairArchibald/YewPar`
- B. Archibald, P. Maier, R. Stewart, P. Trinder (2019). *Implementing YewPar: a Framework for Parallel Tree Search*. In proceedings of EuroPar 2019 (to appear)
- B. Archibald (2018). *Algorithmic Skeletons For Exact Combinatorial Search At Scale*. PhD thesis, University of Glasgow
- B. Archibald, P. Maier, C. McCreesh, R. Stewart, P. Trinder (2018). *Replicable Parallel Branch and Bound Search*. J. Parallel Distrib. Comput. 113
- B. Archibald, P. Maier, R. Stewart, P. Trinder, J. De Beule (2017). *Towards Generic Scalable Parallel Combinatorial Search*. In proceedings of PASCO 2017

Future Work

- More applications
- Scaling up to HPC
- Symmetry
  - How to prune branches that are symmetric to already explored branches?
- Integration with GAP?
  - Maybe, if GAP can be linked as a well-behaved library