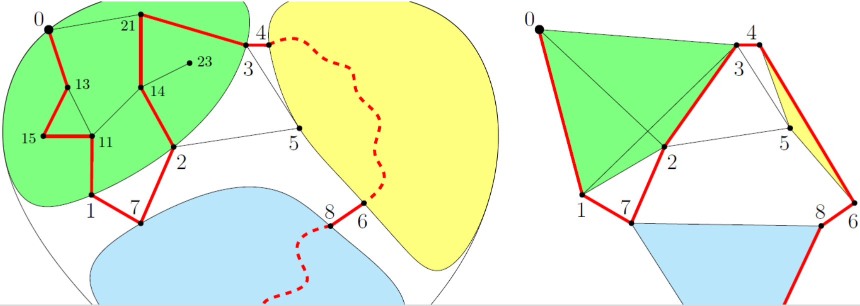
Finding Optimal Longest Paths by Dynamic Programming in Parallel

12th Annual Symposium on Combinatorial Search

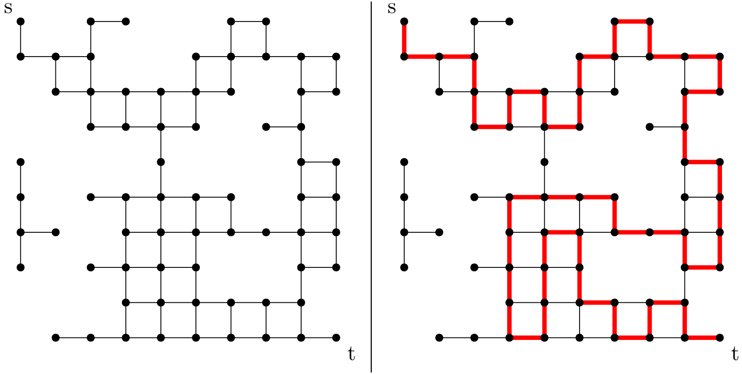
Kai Fieger, Tomáš Balyo, Christian Schulz, Dominik Schreiber | July 17, 2019

INSTITUTE OF THEORETICAL INFORMATICS, KARLSRUHE INSTITUTE OF TECHNOLOGY



Longest Path Problem (LP)

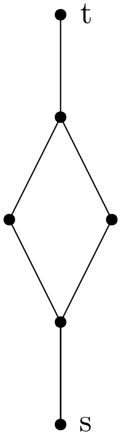
Given: Undirected graph $G = (V, E)$, start and target vertex $s, t \in V$
Problem: Find a simple path of maximum length from s to t.



NP-complete!

Exhaustive Depth First Search

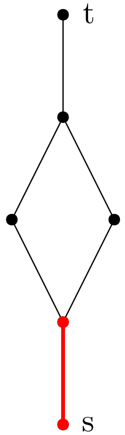
Search *exhaustiveDFS(v)*
    if *v is unmarked* **then**
        mark v;
        foreach $\{v, w\} \in E$ **do**
            exhaustiveDFS(w);
        end
        unmark v;
    end



Exhaustive Depth First Search

Search *exhaustiveDFS(v)*

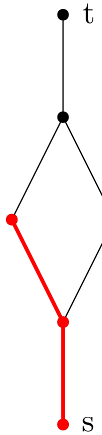
```
if v is unmarked then
    mark v;
    foreach {v, w} ∈ E do
        exhaustiveDFS(w);
    end
    unmark v;
end
```



Exhaustive Depth First Search

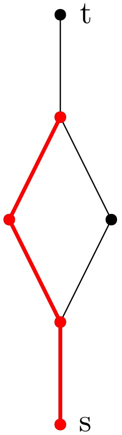
Search *exhaustiveDFS(v)*

if *v is unmarked* **then**
  mark *v*;
  foreach $\{v, w\} \in E$ **do**
    exhaustiveDFS(*w*);
  end
  unmark *v*;
end



Exhaustive Depth First Search

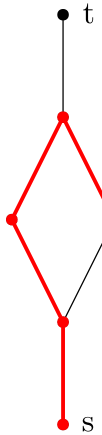
Search *exhaustiveDFS(v)*
- **if** *v is unmarked* **then**
  - mark *v*;
  - **foreach** $\{v, w\} \in E$ **do**
    - exhaustiveDFS(*w*);
  - **end**
  - unmark *v*;
- **end**



Exhaustive Depth First Search

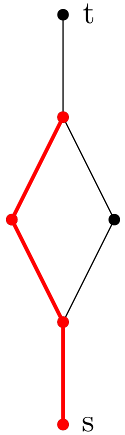
Search *exhaustiveDFS(v)*

if *v is unmarked* **then**
  mark v;
  foreach $\{v, w\} \in E$ **do**
    exhaustiveDFS(w);
  end
  unmark v;
end



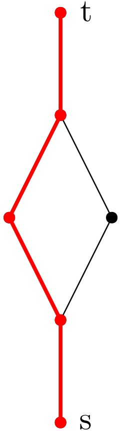
Exhaustive Depth First Search

Search *exhaustiveDFS(v)*
- **if** *v is unmarked* **then**
  - mark v;
  - **foreach** $\{v, w\} \in E$ **do**
    - exhaustiveDFS(w);
  - **end**
  - unmark v;
- **end**



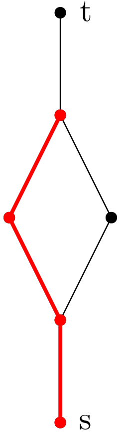
Exhaustive Depth First Search

Search *exhaustiveDFS(v)*
- **if** *v is unmarked* **then**
  - mark *v*;
  - **foreach** $\{v, w\} \in E$ **do**
    - exhaustiveDFS(*w*);
  - **end**
  - unmark *v*;
- **end**



Exhaustive Depth First Search

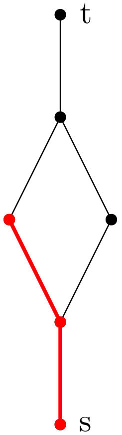
Search *exhaustiveDFS(v)*
  if *v is unmarked* **then**
    mark *v*;
    foreach $\{v, w\} \in E$ **do**
      exhaustiveDFS(*w*);
    end
    unmark *v*;
  end



Exhaustive Depth First Search

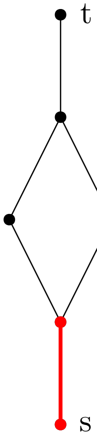
Search *exhaustiveDFS(v)*

> **if** *v is unmarked* **then**
> > mark v;
> > **foreach** $\{v, w\} \in E$ **do**
> > > exhaustiveDFS(w);
> >
> > **end**
> > unmark v;
>
> **end**



Exhaustive Depth First Search

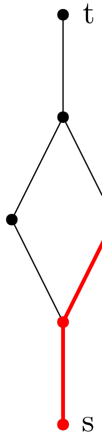
```
Search exhaustiveDFS(v)
    if v is unmarked then
        mark v;
        foreach {v, w} ∈ E do
            exhaustiveDFS(w);
        end
        unmark v;
    end
```



Exhaustive Depth First Search

Search *exhaustiveDFS(v)*

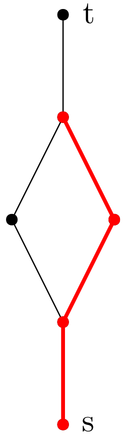
```
if v is unmarked then
    mark v;
    foreach {v, w} ∈ E do
        exhaustiveDFS(w);
    end
    unmark v;
end
```



Exhaustive Depth First Search

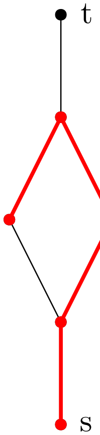
Search *exhaustiveDFS(v)*

```
if v is unmarked then
    mark v;
    foreach {v, w} ∈ E do
        exhaustiveDFS(w);
    end
    unmark v;
end
```



Exhaustive Depth First Search

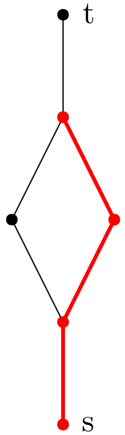
Search *exhaustiveDFS(v)*
  if *v is unmarked* **then**
    mark *v*;
    foreach $\{v, w\} \in E$ **do**
      exhaustiveDFS(*w*);
    end
    unmark *v*;
  end



Exhaustive Depth First Search

Search *exhaustiveDFS(v)*

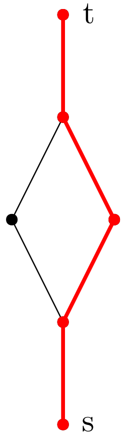
```
if v is unmarked then
    mark v;
    foreach {v, w} ∈ E do
        exhaustiveDFS(w);
    end
    unmark v;
end
```



Exhaustive Depth First Search

Search *exhaustiveDFS(v)*

```
if v is unmarked then
    mark v;
    foreach {v, w} ∈ E do
        exhaustiveDFS(w);
    end
    unmark v;
end
```



Generalized Longest Path Problem

Given:

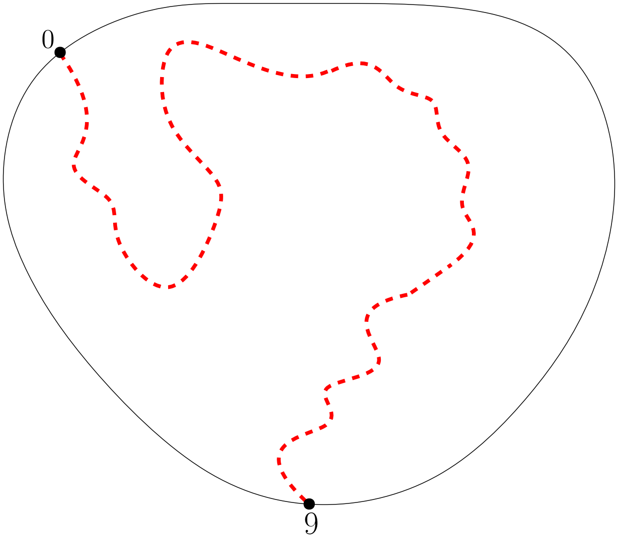
- $G = (V, E)$ and $s, t \in V$
- a block $B \subseteq V$
- the boundary of B:
  $b(B) := \{v \in B \mid v = s \vee v = t \vee \exists\{v, w\} \in E : w \notin B\}$
- a set of boundary vertex pairs $P \subseteq \{\{a, b\} \mid a, b \in b(B)\}$

Problem:

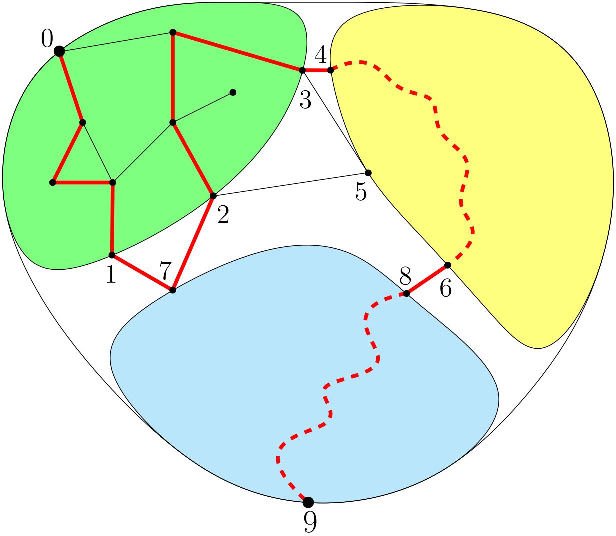
- Restrict G to the vertices of B
- Find a simple path from a to b for each $\{a, b\} \in P$
- Find these paths in such a way that they do not intersect and have the maximum possible cumulative weight.

Special case: $B = V$ and $P = \{\{s, t\}\}$ results in LP problem for G

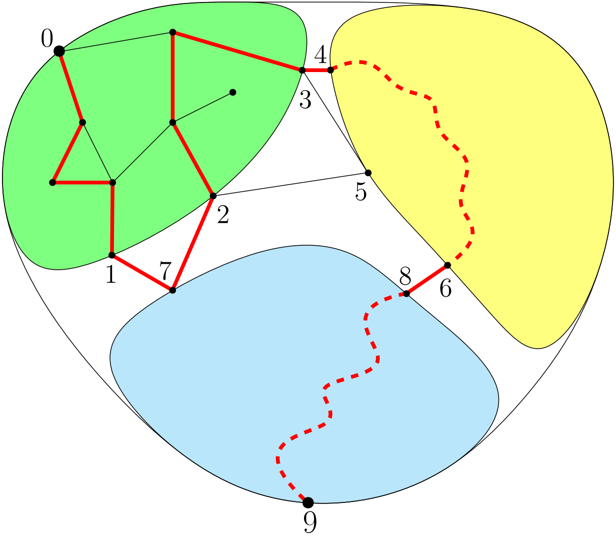
$B = V$ **and** $P = \{\{0, 9\}\}$



Partition of G



Subproblems



$P = \{\{0, 9\}\}$

$P_{green} = \{\{0, 1\}, \{2, 3\}\}$
$P_{yellow} = \{\{4, 6\}\}$
$P_{blue} = \{\{7, 7\}, \{8, 9\}\}$

Longest Path by Dynamic Programming

Multiple levels of partitioning $(0, 1, \ldots, L)$

- Level 0: Finest level of partitioning, many small blocks
- Level $i \rightsquigarrow i+1$: Multiple subblocks yield a bigger block $B = B_1 \cup B_2 \cup \ldots \cup B_n$
- Level L: one single block $B = V$

Longest Path by Dynamic Programming (LPDP)

- Solve generalized LP for every level 0 block and every possible P
- Level $i = 1, \ldots, L$:
  Combine solutions for subblocks $B_1, B_2, \ldots, B_n$ to solve block B

Longest Path by Dynamic Programming

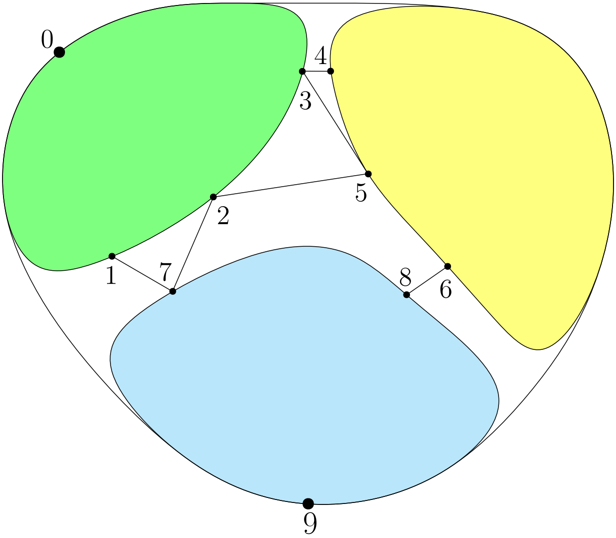
Multiple levels of partitioning $(0, 1, \ldots, L)$

- Level 0: Finest level of partitioning, many small blocks
- Level $i \rightsquigarrow i + 1$: Multiple subblocks yield a bigger block $B = B_1 \cup B_2 \cup \ldots \cup B_n$
- Level L: one single block $B = V$

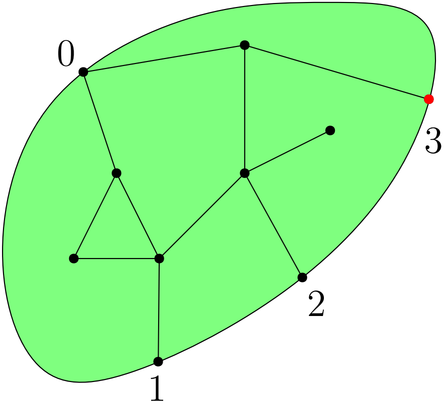
Longest Path by Dynamic Programming (LPDP)

- Solve generalized LP for every level 0 block and every possible P
- Level $i = 1, \ldots, L$:
  Combine solutions for subblocks $B_1, B_2, \ldots, B_n$ to solve block B

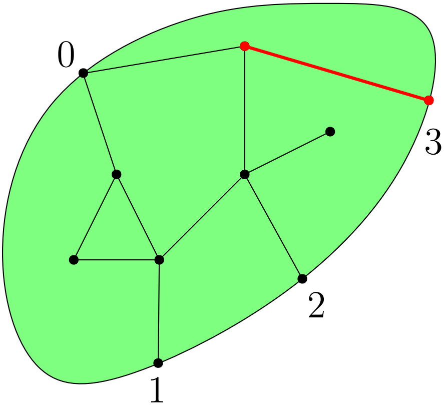
Solving a Level 0 Block



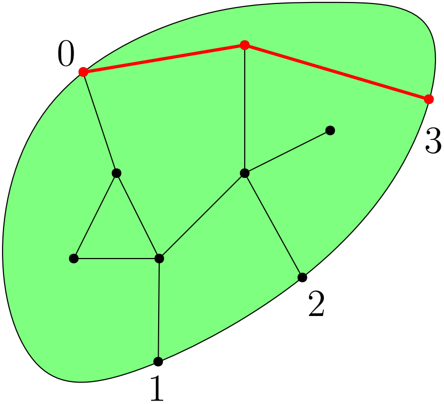
Solving a Level 0 Block



Solving a Level 0 Block

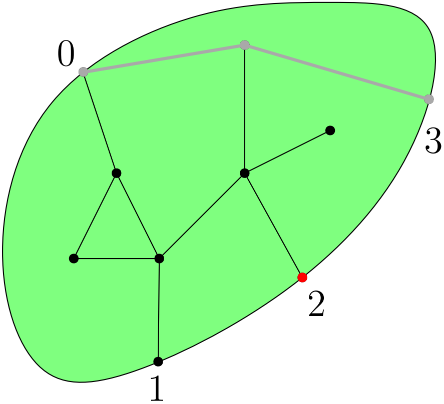


Solving a Level 0 Block

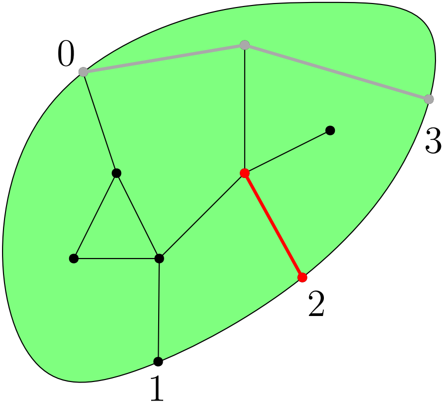


$P = \{\{0, 3\}\}$

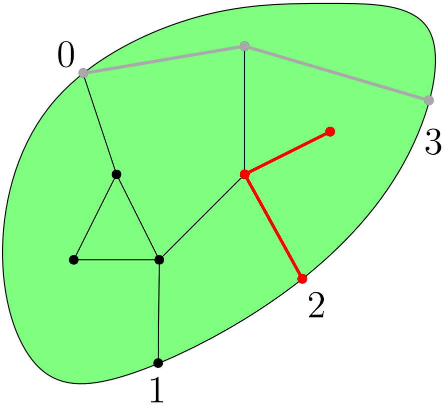
Solving a Level 0 Block



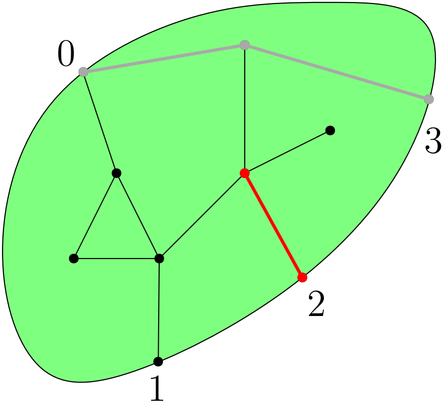
Solving a Level 0 Block



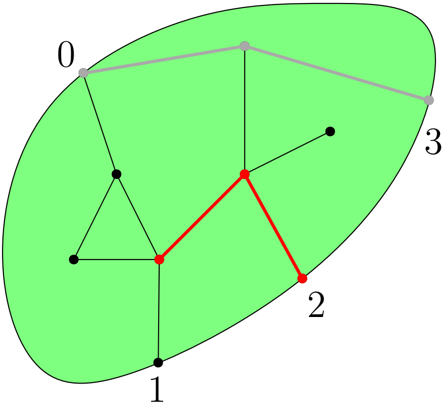
Solving a Level 0 Block



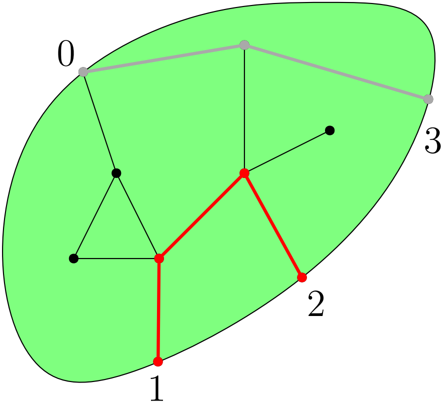
Solving a Level 0 Block



Solving a Level 0 Block

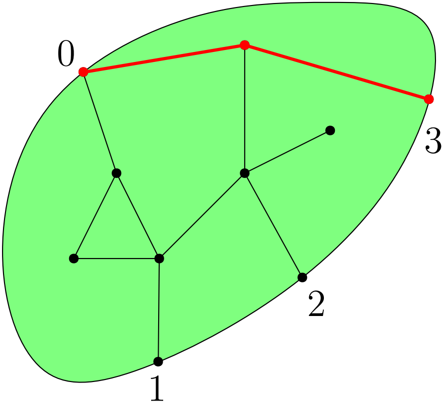


Solving a Level 0 Block

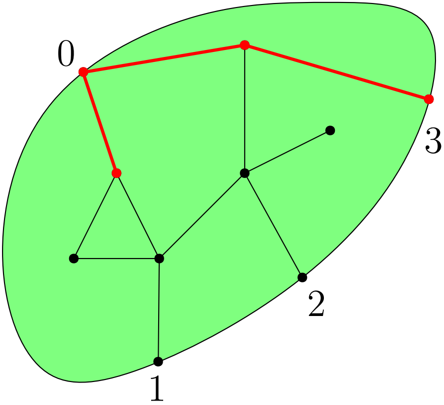


$P = \{\{0, 3\}, \{1, 2\}\}$

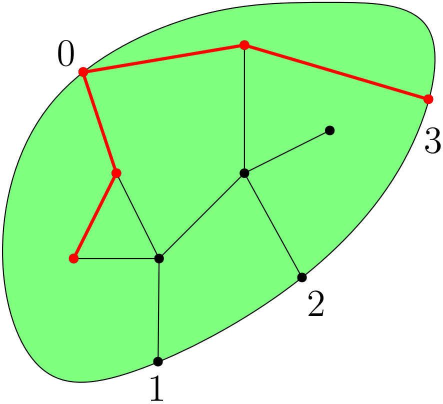
Solving a Level 0 Block



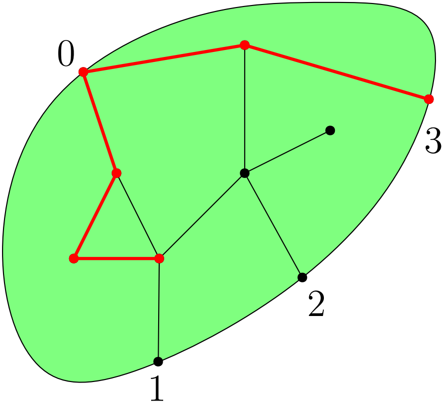
Solving a Level 0 Block



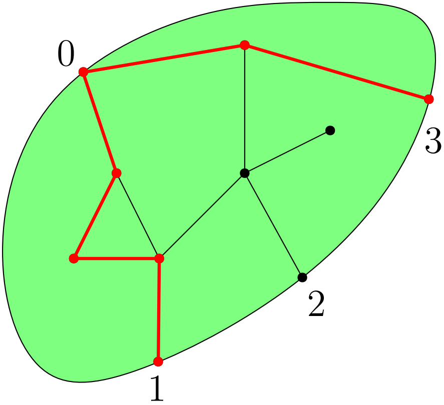
Solving a Level 0 Block



Solving a Level 0 Block

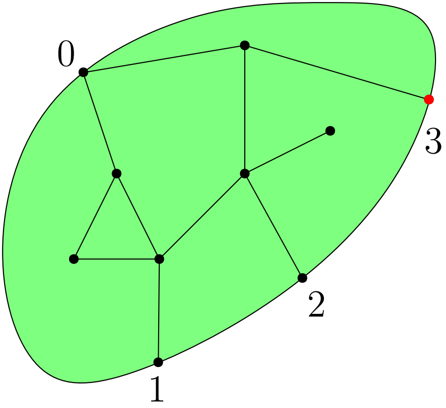


Solving a Level 0 Block



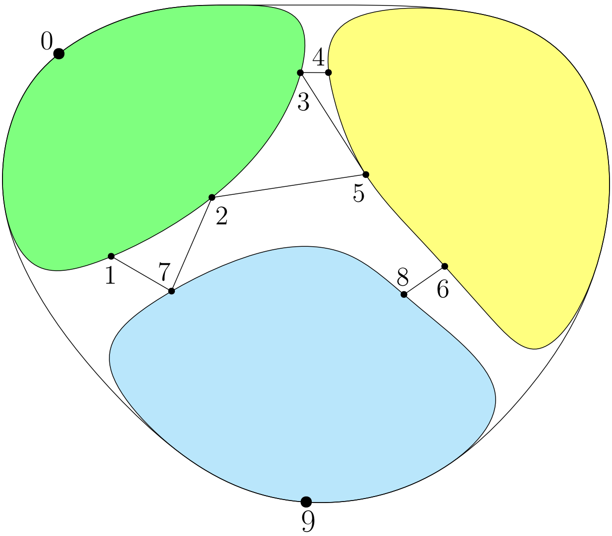
$P = \{\{1, 3\}\}$

Solving a Level 0 Block

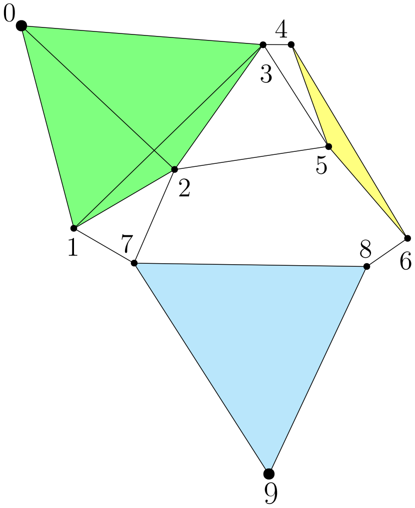


$\{\{0\}\}, \{\{1\}\}, \{\{2\}\}, \{\{3\}\},$
$\{\{0\}, \{1\}\}, \{\{0\}, \{2\}\}$ ...
$\{\{0\}, \{1\}, \{2\}\}, \{\{0\}, \{1\}, \{3\}\}$ ...
$\{\{0\}, \{1\}, \{2\}, \{3\}\},$
$\{\{0\}, \{1, 2\}\}, \{\{0\}, \{1, 3\}\},$
$\{\{0\}, \{2, 3\}\}, \{\{1\}, \{0, 2\}\},$
$\{\{1\}, \{0, 3\}\}, \{\{1\}, \{2, 3\}\},$
...
$\{\{0, 1\}, \{2, 3\}\}, \{\{1, 2\}, \{0, 3\}\},$
$\{\{0, 2\}, \{1, 3\}\}$

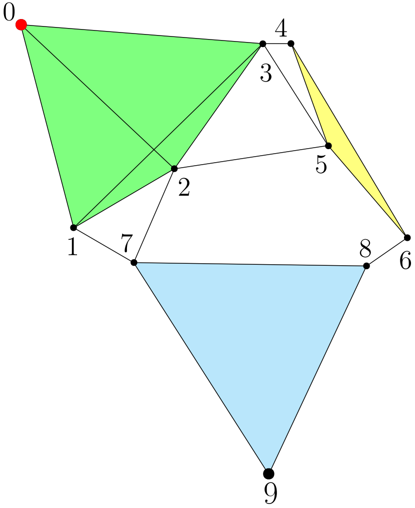
Combining Solutions



Combining Solutions



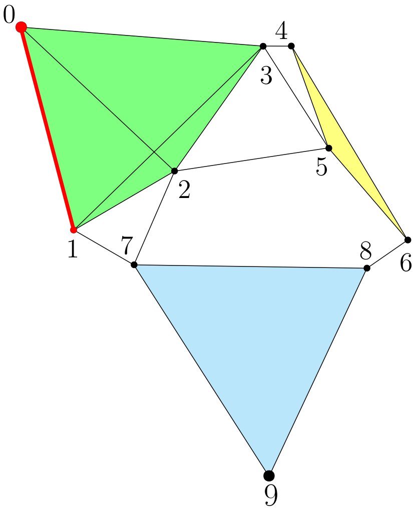
Combining Solutions



$$P = \{\}$$

$$P_{green} = \{\{0, 0\}\}$$
$$P_{yellow} = \{\}$$
$$P_{blue} = \{\}$$

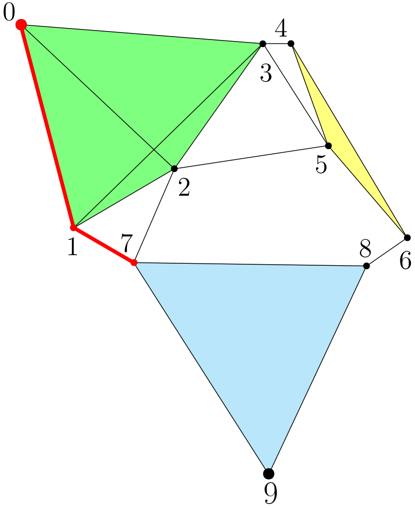
Combining Solutions



$P = \{\}$

$P_{green} = \{\{0, 1\}\}$
$P_{yellow} = \{\}$
$P_{blue} = \{\}$

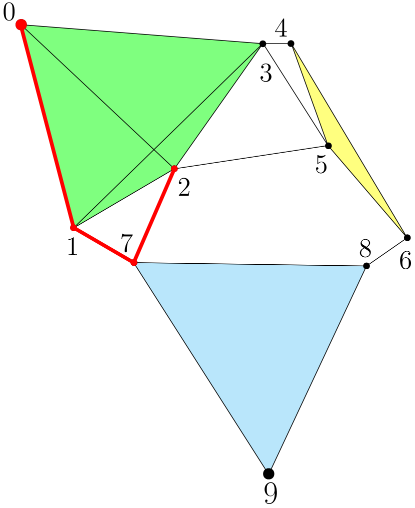
Combining Solutions



$P = \{\}$

$P_{green} = \{\{0, 1\}\}$
$P_{yellow} = \{\}$
$P_{blue} = \{\{7, 7\}\}$

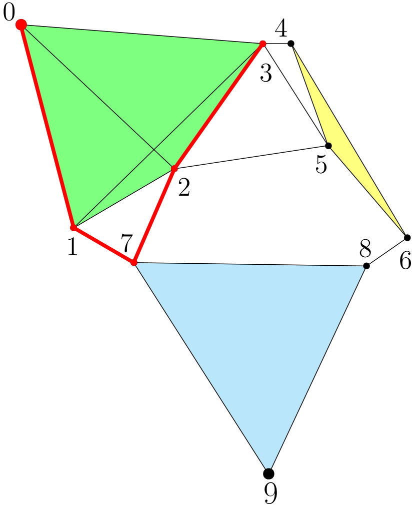
Combining Solutions



$$P = \{\}$$

$$P_{green} = \{\{0, 1\}, \{2, 2\}\}$$
$$P_{yellow} = \{\}$$
$$P_{blue} = \{\{7, 7\}\}$$

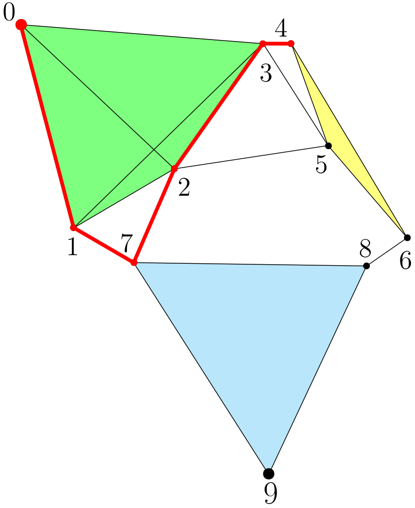
Combining Solutions



$P = \{\}$

$P_{green} = \{\{0, 1\}, \{2, 3\}\}$
$P_{yellow} = \{\}$
$P_{blue} = \{\{7, 7\}\}$

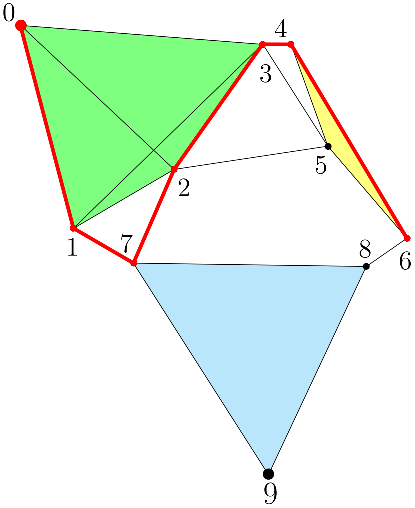
Combining Solutions



$P = \{\}$

$P_{green} = \{\{0, 1\}, \{2, 3\}\}$
$P_{yellow} = \{\{4, 4\}\}$
$P_{blue} = \{\{7, 7\}\}$

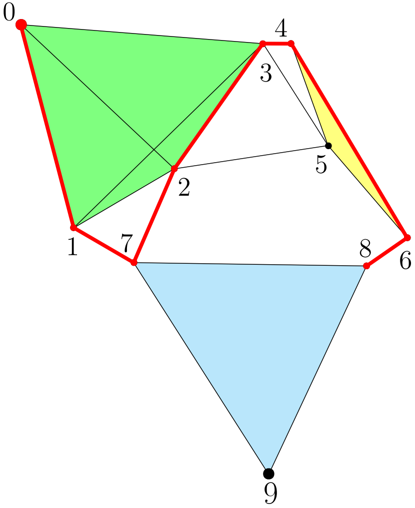
Combining Solutions



$P = \{\}$

$P_{green} = \{\{0, 1\}, \{2, 3\}\}$
$P_{yellow} = \{\{4, 6\}\}$
$P_{blue} = \{\{7, 7\}\}$

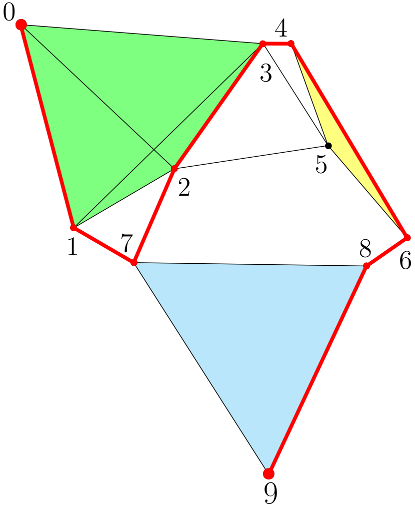
Combining Solutions



$P = \{\}$

$P_{green} = \{\{0, 1\}, \{2, 3\}\}$
$P_{yellow} = \{\{4, 6\}\}$
$P_{blue} = \{\{7, 7\}, \{8, 8\}\}$

Combining Solutions



$P = \{\{0, 9\}\}$

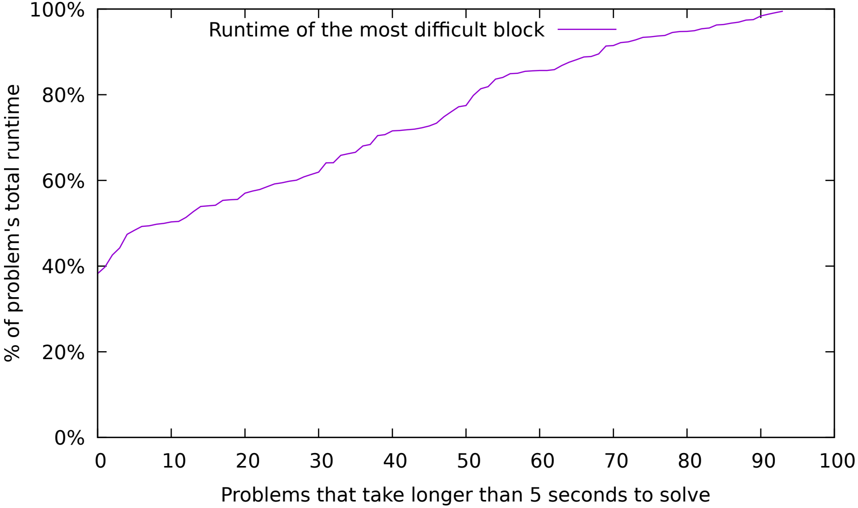
$P_{green} = \{\{0, 1\}, \{2, 3\}\}$
$P_{yellow} = \{\{4, 6\}\}$
$P_{blue} = \{\{7, 7\}, \{8, 9\}\}$

Parallelization

Solving multiple blocks in parallel

- A block can be solved once its subblocks have been solved: independence among blocks on the same level
- Simple, synchronization-free opportunity for parallelization!

Parallelization



Parallelization

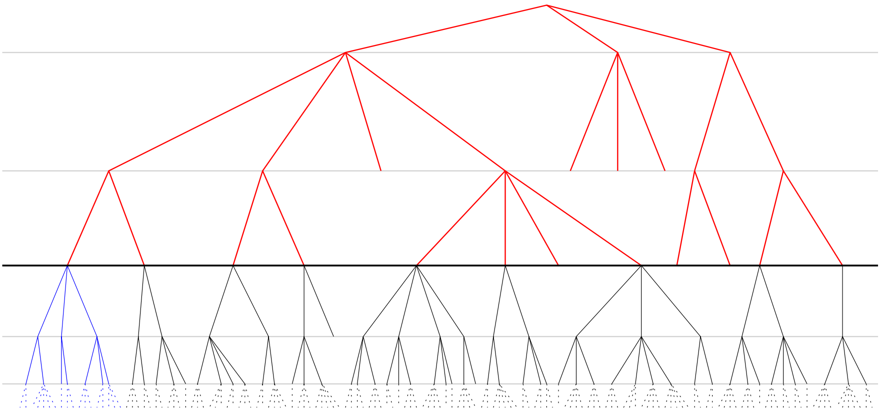
Solving multiple blocks in parallel

- A block can be solved once its subblocks have been solved: independence among blocks on the same level
- Simple, synchronization-free opportunity for parallelization!
- Inefficient: Majority of work still done sequentially

Parallelize algorithm within a single block

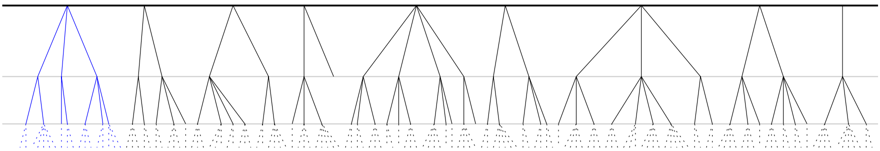
Parallelizing LPDP-Search

- Run LPDP-Search() with a limited recursion depth
- Store the current state of the search each time the limit is reached



Parallelizing LPDP-Search

- Run LPDP-Search() with a limited recursion depth
- Store the current state of the search each time the limit is reached



Parallelizing LPDP-Search

- Execute the search branches in parallel
- List of branches represents a queue
- Thread finishes a branch $\rightarrow$ get next branch from front of the queue

  $\Rightarrow$ Leads to good load balancing

  $\Rightarrow$ Requires some synchronization
  (fetching branches, updating partial solutions)

Evaluations

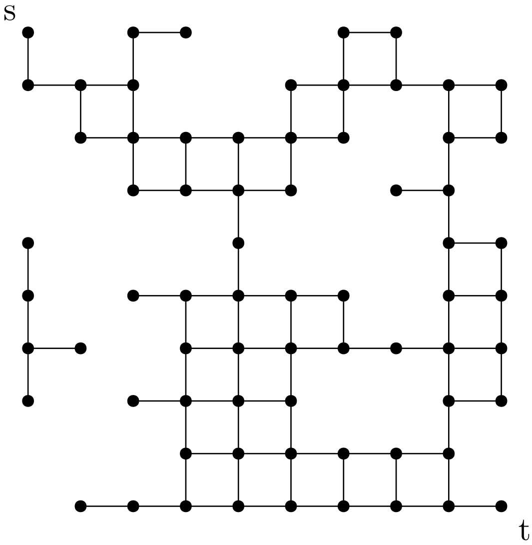
Instances

- 300 grids
  - 30%: sizes $10x10, 15x15, \ldots, 50x50$ (10 instances per size)
  - 40%: sizes $10x10, 15x15, \ldots, 120x120$ (10 instances per size)
- 150 road graphs: $2, 4, 6, \ldots, 300$ vertices
- 60 word graphs: sizes $10, 20, \ldots, 60$, 10 instances per size

Competitors

- LPDP with two configurations of graph partitioner KaHIP: eco (LPDPe) and strong/high-quality (LPDPs)
- Exhaustive Depth-first search (ExDFS)
- Stern et al. 2014 : A* and Depth-first Branch&Bound (DFBnB)

10x10 **Grid.** 30% **Deleted Vertices**



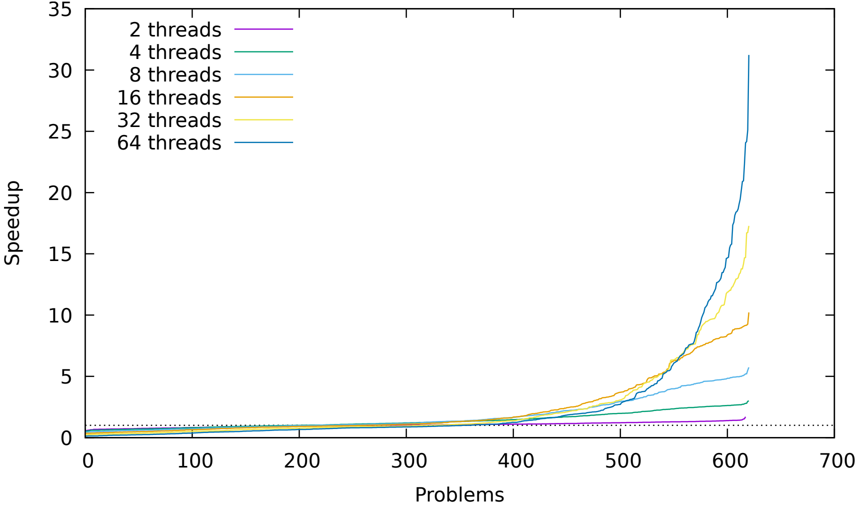
Comparison with previous algorithms

Solver	Number of Solved Instances			
	Grids	Roads	Words	Total
A*	34	70	40	144
DFBnB	37	85	46	168
ExDFS	31	72	44	147
LPDPe	296	144	58	498
LPDPs	300	144	59	503

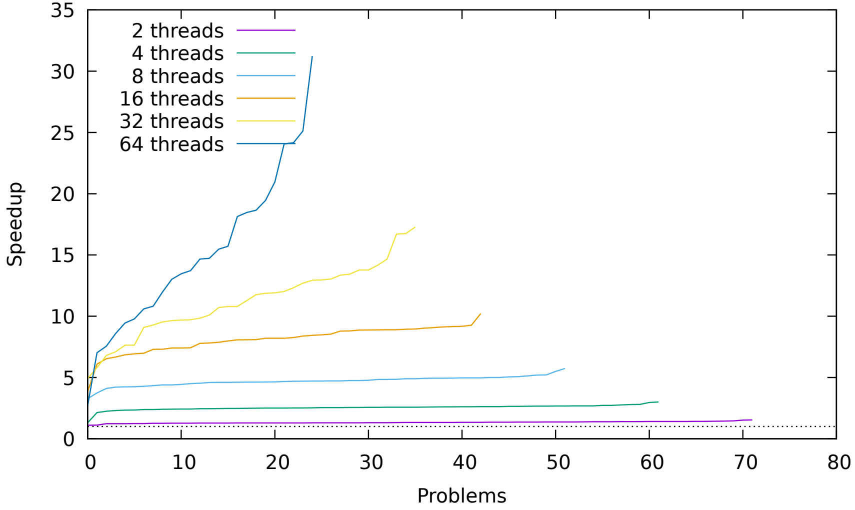
Parallel Speedups

Threads	Parallel Solved	Both Solved	Speedup All			Speedup Big		
			Avg.	Tot.	Med.	Avg.	Tot.	Med.
2	618	618	1.038	1.360	1.031	1.335	1.363	1.331
4	624	621	1.392	2.623	1.224	2.542	2.638	2.564
8	627	621	1.788	4.833	1.189	4.707	4.913	4.720
16	628	621	2.257	8.287	1.127	8.097	8.569	8.208
32	629	621	2.344	10.714	0.987	11.272	11.553	11.519
64	633	621	2.474	11.691	0.889	15.180	13.512	14.665

Parallel Speedups



“Hard” Instances (weak scaling)



Conclusion

- New exact algorithm LPDP for longest path problem by graph partitioning and dynamic programming
- Experiments: LPDP outperforms previous exact algorithms on nontrivial problems
- Efficient parallelization of our algorithm: Significant speedup for up to 64 threads

Thank you for your attention!

Worst Case Performance

Clique/Complete graph

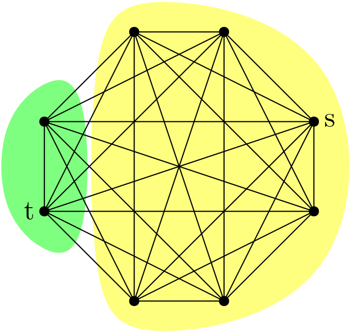
- random vertices $s \neq t$
- random edge weights $\in [0, 100]$

LPDP partitioning:

- Level 0: two blocks B_1 and B_2

  B_1 contains s and 75% of all vertices

  B_2 contains t and 25% of all vertices
- Level 1: one block $B = V$



Worst Case Performance

vertices	exhDFS	LPDP	
	runtime [s]	runtime [s]	speedup
8	$<$0.001	0.001	0.102
9	$<$0.001	0.002	0.262
10	0.005	0.007	0.758
11	0.034	0.031	1.098
12	0.278	0.254	1.095
13	3.107	0.527	5.895
14	37.761	3.682	10.255
15	535.457	40.667	13.167
16	N/A	568.151	N/A
17	N/A	1081.140	N/A

Worst Case Performance

vertices	exhDFS	LPDP	
	steps	steps	speedup
8	3 914	5 828	0.672
9	27 400	11 532	2.376
10	219 202	62 552	3.504
11	1 972 820	478 234	4.125
12	19 728 202	4 760 073	4.145
13	217 010 224	8 659 923	25.059
14	2 604 122 690	66 442 136	39.194
15	33 853 594 972	738 510 944	45.840
16	N/A	9 995 560 186	N/A
17	N/A	16 468 631 539	N/A