About some problems arising from large scale parallelization of NP class combinatorial problems

Bogdán Zaválnij

Alfréd Rényi Institute of Mathematics
Hungarian Academy of Sciences

Zuse Institute Berlin, 2019

Table of Contents

Example class: the Maximum Clique Problem

Let G be a finite, simple graph: $G = (V, E)$, and C be a subgraph, which has the nodes $\Delta \subseteq V$, and C is spanned by Δ.

- We call the subgraph C a clique, if all of its nodes are connected to each other: $\forall v_i, \forall v_j \in \Delta, i \neq j : (v_i, v_j) \in E$
- We call the size of the clique the number of the nodes in the clique
- we call the clique size of the graph the size of its biggest clique (maximum clique), and denote it by $\omega(G)$.

We can search for the size of a maximum clique, or ask if a given graph has a clique of size k:

- The k-clique decision problem is a well known NP-complete problem
- The maximum clique optimization problem is NP-hard

Is there any real difference between the two problem?

Each algorithm solves the other problem as well:

- finding a maximum clique of size ω will answer if there is a clique of size k (is k smaller or equal, or is it bigger?)
- Using k-clique search and an upper bound (coloring) one can construct a trivial maximum clique search algorithm:

Require: k =an upper bound

```
1:  function kCLIQUE-SEQ(G = (V, E))
2:      FOUND ← false
3:      while ¬FOUND do
4:          FOUND ← kCLIQUE(V, k)
5:          if ¬FOUND then
6:              k ← k − 1
7:          end if
8:      end while
9:      return k
10: end function
```

Yes, we think there are huge differences!

For exact solution of the maximum clique problem a backtracking algorithm used.

For example the Carraghan–Pardalos algorithm is a classical Branch-and-Bound technique. We take the nodes of the graph each by one, and reduce the graph to their neighborhood.
If the reduced graph is *"not satisfactory"* we go back, if it is *"satisfactory"* we do the same (go forward).

- branching: we try several different nodes, if they should be in a maximum clique
- bound: we try to prune the branches of the search tree (number of nodes, coloring, Lovász' θ)

Carraghan–Pardalos Algorithm (1990)

C is the nodes of the clique we are building, C^* are the nodes of the biggest clique found till now. P are the prospective nodes.

Require: $C = \emptyset, C^* = \emptyset, P = V$

```
1:  function CP(C, P)
2:      if |C| > |C*| then
3:          C* ← C
4:      end if
5:      if |C| + |P| > |C*| then
6:          for all vertex p ∈ P do
7:              CP(C ∪ {p}, P ∩ N(p))
8:              P ← P \ {p}
9:          end for
10:     end if
11: end function
```

Carraghan–Pardalos Algorithm (1990)

C is the nodes of the clique we are building, C^* are the nodes of the biggest clique found till now. P are the prospective nodes.

Require: $C = \emptyset, C^* = \emptyset, P = V$

```
1:  function CP(C, P)
2:      if |C| > |C*| then
3:          C* ← C
4:      end if
5:      if |C| + |P| > |C*| then
6:          for all vertex p ∈ P do
7:              CP(C ∪ {p}, P ∩ N(p))
8:              P ← P \ {p}
9:          end for
10:     end if
11: end function
```

Problems

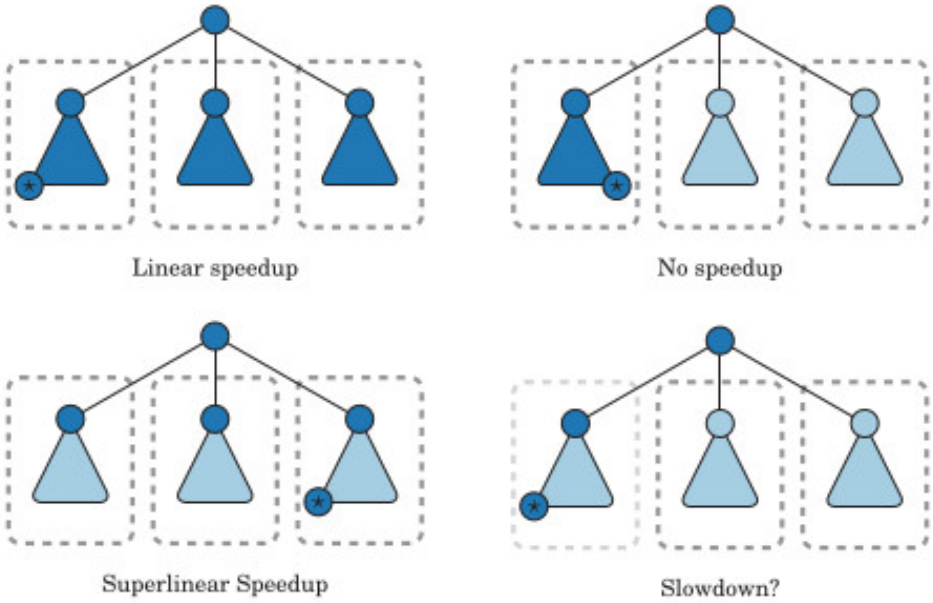
The maximum clique problem has several disadvantages:

1. We branch on the whole P.
2. Finding early a good (big) C^* is crucial.
   (Why do algorithms not searching it heuristically in the beginning?)
3. Heuristics (usually node ordering) for finding a big clique and proving the nonexistence of a bigger one differ.
4. Basically there are two goals at once contradicting each other.
   (Finding a solution and proving the nonexistence.)

Parallel – uneven distribution, superlinear speedup

Using parallel branching leads to extremely uneven distribution.
$\rightarrow$ Difference in several magnitudes.
Branching on first level is good, on second level acceptable, on third level usually does not work. Is other parallelization possible?

Sometimes we see a superlinear speedup. Is it good?

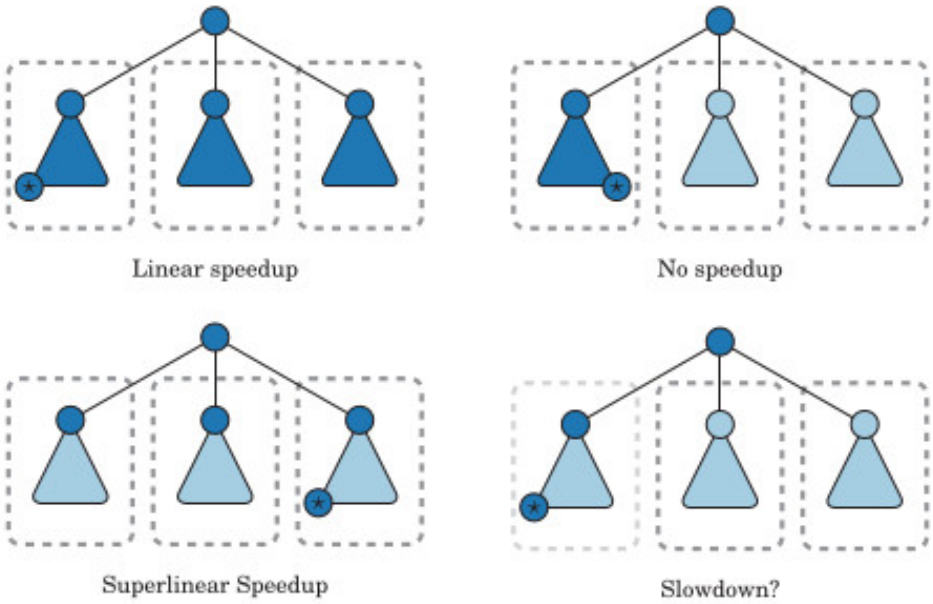


"Superlinear speedup of efficient sequential algorithm is not possible"
1986. Faber, Lubeck, White.

Parallel – uneven distribution, superlinear speedup

Using parallel branching leads to extremely uneven distribution.
$\rightarrow$ Difference in several magnitudes.
Branching on first level is good, on second level acceptable, on third level usually does not work. Is other parallelization possible?

Sometimes we see a superlinear speedup. Is it good?



"Superlinear speedup of efficient sequential algorithm is not possible"
1986. Faber, Lubeck, White.

$k$clique – advantages

```
1:  function kCLIQUE(P, k)
2:      if k = 0 then return true
3:      end if
4:      KCCNS ← construct a k-clique covering node set
5:      for all vertex p ∈ KCCNS do
6:          if kCLIQUE(P ∩ N(p), k − 1) then return true
7:          end if
8:          P ← P \ {p}
9:      end for
10:     return false
11: end function
```

1. Only nonexistence is the goal
2. We can do a better branching (smaller branching factor – Knuth)
3. There is a good estimate for the size of the search tree (Knuth)
4. Can use good ordering of nodes $\rightarrow$ reduce search tree (SAT)

Compared to the 3 best state-of-the-art programs

name	$\|V\|$	%	$\omega(G)$	*k*clique -seq	*k*clique, $k = \omega(G)+1$	BBMC -X	M-clq -13	mcqd -dyn
brock800_3	800	65	25	8837	1955	2452	5561	4290
brock800_4	800	65	26	7277	1543	1787	7037	3072
latin_square_10	900	76	90	213	131	*	*	1180
keller5	776	75	27	53	46	*	238	18098
MANN_a45	1035	99	345	*	*	82	123	2058
sanr200_0.9	200	90	42	141	55	21	2	30
sanr400_0.7	400	70	21	419	140	105	117	110
monoton-7	343	79	19	12	3	31	6	72
monoton-8	512	82	23	846	576	15049	1279	19272
1dc.256	256	88	30	8	2	2	5	22
2dc.1024	1024	68	16	178	112	40	199	146
frb30-15-1	450	82	30	0	0	1613	0	2541
frb35-17-1	595	84	35	2	0	*	1	*
frb40-19-1	760	86	40	17	0	*	1	*
frb45-21-1	945	87	45	839	0	*	119	*
frb50-23-1	1150	88	50	1351	0	*	764	*
frb53-24-1	1272	88	53	42161	0	*	4771	*
frb59-26-1	1534	89	59	*	0	*	*	*
evil-myc11x14	154	94	28	14396	14256	66	235	11563
evil-myc5x36	180	97	72	590	396	2	0	6
evil-myc23x8	184	90	16	645	624	88	23434	1390
evil-s3m25x8	200	92	32	*	*	38987	1206	18148

Table: Running time results in seconds. The "*" sign indicates that the running times are exceeding the 12 hour limit.

Compared to the 3 best state-of-the-art programs

name	$\|V\|$	%	$\omega(G)$	$k$clique -seq	$k$clique, $k = \omega(G)+1$	BBMC -X	M-clq -13	mcqd -dyn
brock800_3	800	65	25	8837	1955	2452	5561	4290
brock800_4	800	65	26	7277	1543	1787	7037	3072
latin_square_10	900	76	90	213	131	*	*	1180
keller5	776	75	27	53	46	*	238	18098
MANN_a45	1035	99	345	*	*	82	123	2058
sanr200_0.9	200	90	42	141	55	21	2	30
sanr400_0.7	400	70	21	419	140	105	117	110
monoton-7	343	79	19	12	3	31	6	72
monoton-8	512	82	23	846	576	15049	1279	19272
1dc.256	256	88	30	8	2	2	5	22
2dc.1024	1024	68	16	178	112	40	199	146
frb30-15-1	450	82	30	0	0	1613	0	2541
frb35-17-1	595	84	35	2	0	*	1	*
frb40-19-1	760	86	40	17	0	*	1	*
frb45-21-1	945	87	45	839	0	*	119	*
frb50-23-1	1150	88	50	1351	0	*	764	*
frb53-24-1	1272	88	53	42161	0	*	4771	*
frb59-26-1	1534	89	59	*	0	*	*	*
evil-myc11x14	154	94	28	14396	14256	66	235	11563
evil-myc5x36	180	97	72	590	396	2	0	6
evil-myc23x8	184	90	16	645	624	88	23434	1390
evil-s3m25x8	200	92	32	*	*	38987	1206	18148

Table: Running time results in seconds. The "*" sign indicates that the running times are exceeding the 12 hour limit.

A k-clique covering node set can be used for parallelization as well!

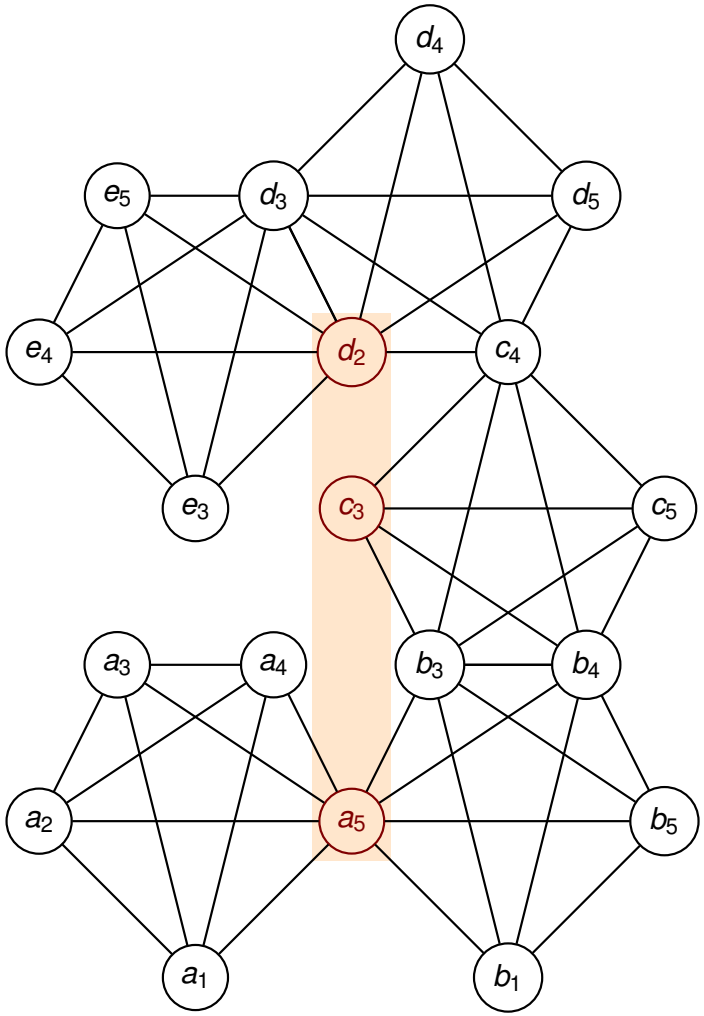


Figure: 5-clique covering node set.

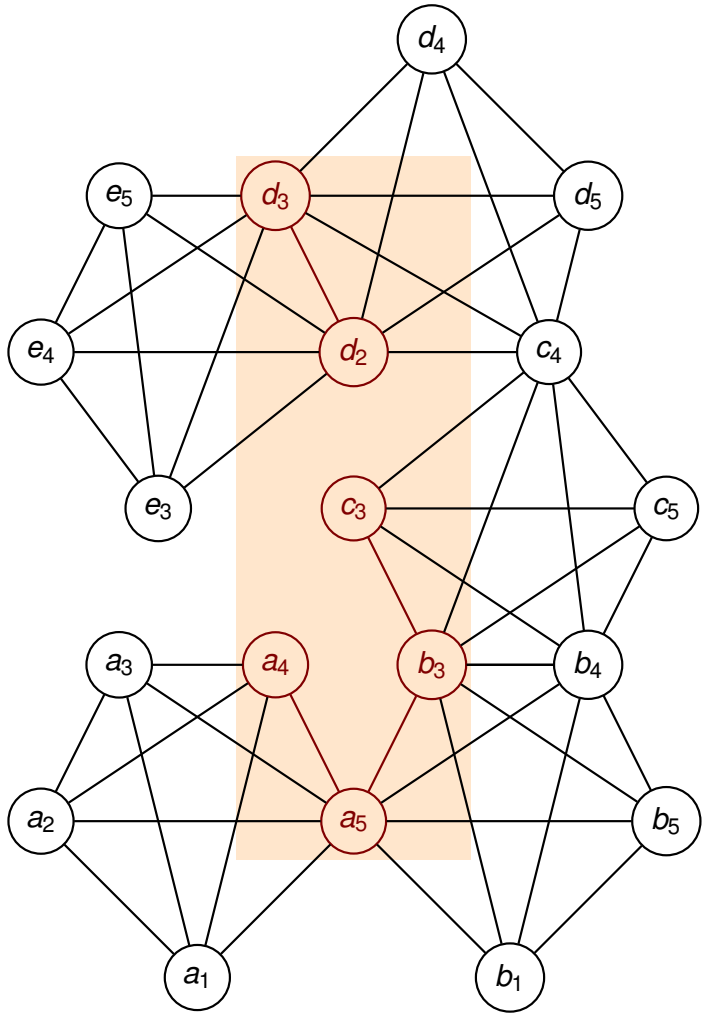


Figure: 5-clique covering edge set.

Estimation of tree size

Knuth, 1975.
Monte Carlo method: random walks in the tree.
Only in k-clique search!
(The tree does not change, as bounds do not change.)

- to know in advantage if the problem is solvable;
- to know the advancement of the program;
- to know which are the hard problems
  - for splitting up
  - for applying more preconditioning.

Las Vegas ordering of sub-problems

The solution of one sub-problem – *there is no* $(k-2)$ *clique in the subgraph of* $N(a,b)$ – effects other sub-problems $\rightarrow$ the edge $\{a,b\}$ can be deleted from all other problems!

The exact sequence of the sub-problems alters the size of the search space. Heuristically it is better to solve the easier problems first.

Question: which problems are easy?

The Las Vegas approach: start all sub-problems parallelly, and those who finish early will "help" others (deletion of a node or edge).
$\rightarrow$ "Work-Help parallelization"

(Other method: see tree size estimation!)

Las Vegas ordering – example

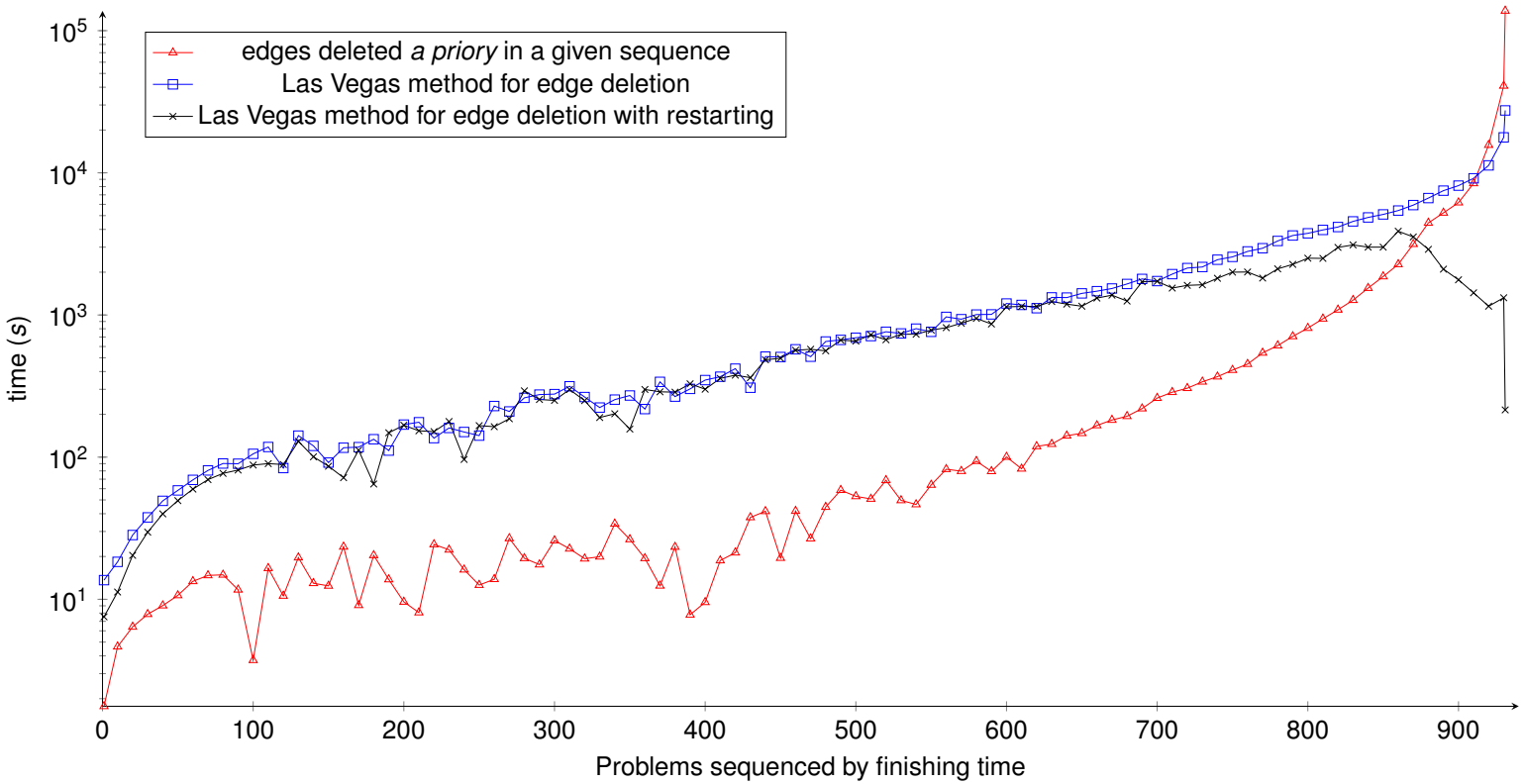


Figure: The time sequence of running times of the `monoton-9` subproblems.

Las Vegas ordering – example

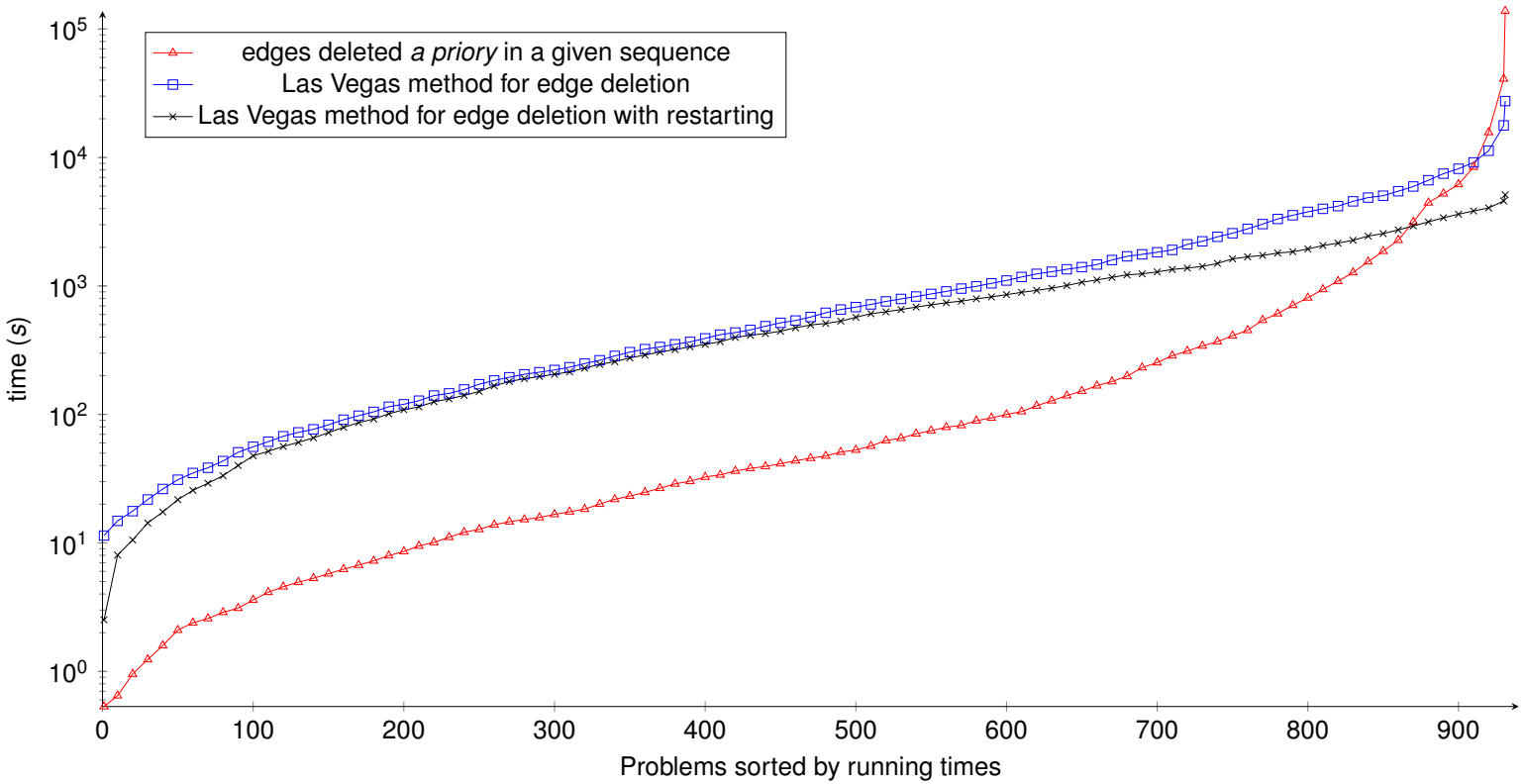


Figure: The sorted running times of the `monoton-9` subproblems.

Conclusion

- Proved to be extremely efficient against state-of-the art programs that use a much better upper bound;
- May be used for solving other combinatorial optimization problems – coloring, scheduling, subgraph isomorphy;
- Some problems better modelled for k-clique;
- Opens new possibilities for subproblem generation $\rightarrow$ new ways of parallelization;
- Was used efficiently for parallelizing on up to 500 cores;
- No superlinear speedup!
- Additional techniques allowed for parallelization on up to 70k cores,

Thank you for your attention!

Questions?

Thank you for your attention!

Questions?