SEA 2012 – Bordeaux

Implementation and Comparison of Heuristics for the Vertex Cover Problem on Huge Graphs*

Eric Angel [1] **Romain Campigotto** [2] Christian Laforest [3]

[1] IBISC, Université d'Évry-Val d'Essonne

[2] LAMSADE, CNRS – Université Paris-Dauphine

[3] LIMOS, CNRS – Université Blaise Pascal (Clermont-Ferrand)

Friday, June 8th, 2012

*Work supported by the project ANR-09-EMER-010 *TODO*

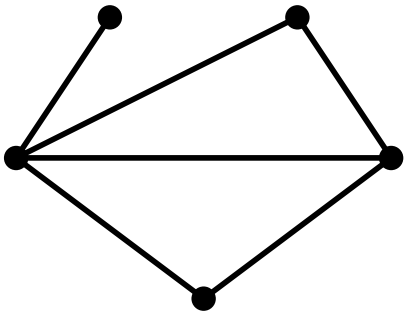
The Vertex Cover problem...

Definition (Cover)

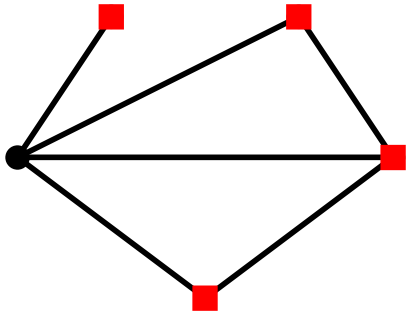
Let $G = (V, E)$ be any simple graph. C is a cover *if $C \subseteq V$ and $\forall e = uv \in E$, $u \in C$ or $v \in C$ (or both).*

Find a *cover* of minimum size

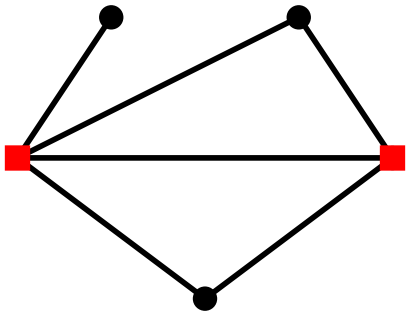
$\rightarrow$ well-known **NP**-complete optimization graph problem



A graph



A cover



An optimal cover

... on huge graphs

It appears in various applied problems:

- network monitoring;
- SNP haplotype assembly problem *(DNA sequence alignment)*;
- etc.

where data sizes can be very important.

> *"The amount of data doubles every 20 months!"*

Previous works

Theoretical study:

→ *Analysis and Comparison of Three Algorithms for the Vertex Cover Problem on Large Graphs with Low Memory Capacities*, Algo. Op. Res., Vol. 6, 2011

Previous works (experimental)

Theoretical study:

→ *Analysis and Comparison of Three Algorithms for the Vertex Cover Problem on Large Graphs with Low Memory Capacities*, Algo. Op. Res., Vol. 6, 2011

Experiments on graphs with $53 \cdot 10^6$ vertices and $170 \cdot 10^6$ edges for the Max Clique problem

→ J. Abello et al.: *On Maximum Clique Problems in Very Large Graphs*, DIMACS, Vol. 50, American Mathematical Society, 1999

Previous works (experimental)

Theoretical study:

→ *Analysis and Comparison of Three Algorithms for the Vertex Cover Problem on Large Graphs with Low Memory Capacities*, Algo. Op. Res., Vol. 6, 2011

Experiments on graphs with $53 \cdot 10^6$ vertices and $170 \cdot 10^6$ edges for the Max Clique problem

→ J. Abello et al.: *On Maximum Clique Problems in Very Large Graphs*, DIMACS, Vol. 50, American Mathematical Society, 1999

Experiments for the Vertex Cover problem:

- sparse graphs with less than 10,000 vertices

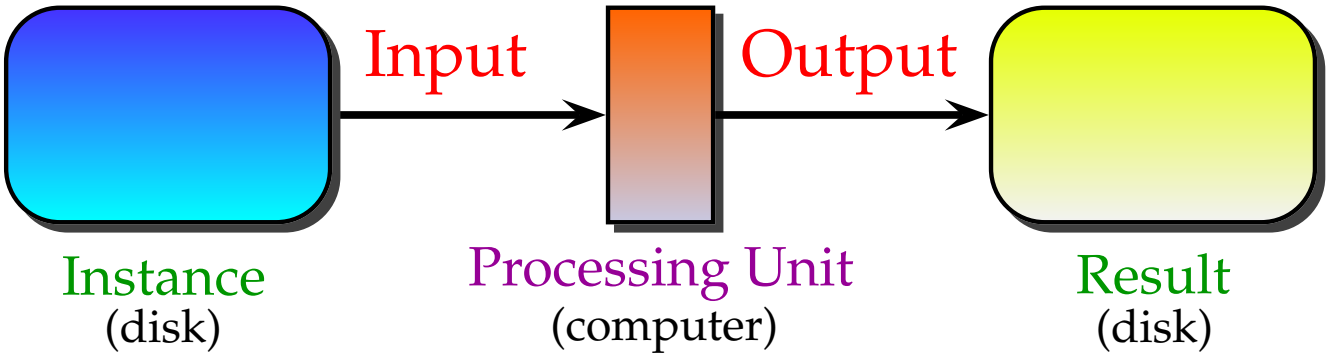
Max sizes $\approx 100 \cdot 10^9$ in our experiments!

Outline

Outline

Implementation

Experimental model



- **Instance:** stored on a external hard disk (*read only* access)
- **Processing unit:** a standard computer (*a laptop!*)

→ low memory capacities

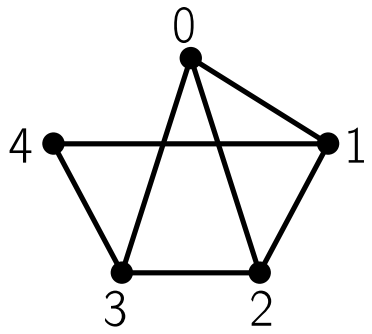
- **Result:** written *as soon as it is produced* on the external disk

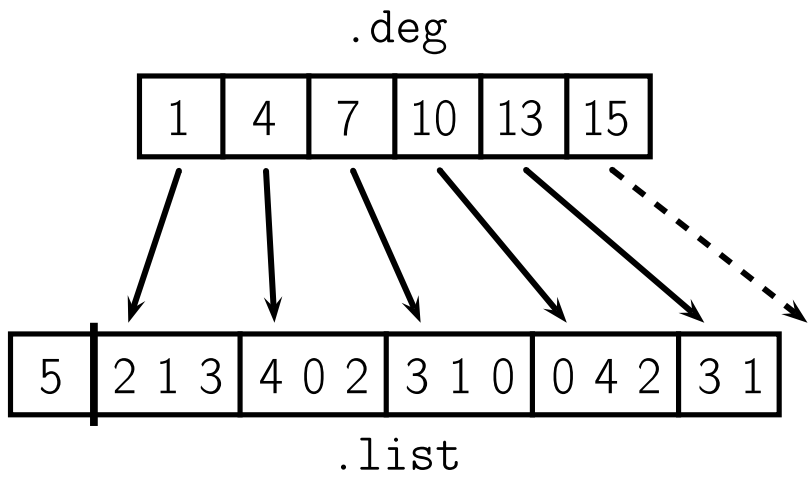
Storage and reading of graphs

With two files

`.list`: list of neighbors of vertices

`.deg`: *pointers* delimiting the neighborhood of each vertex





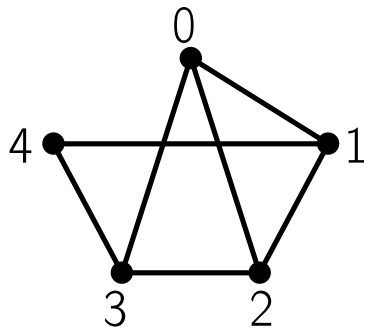
- reading of `.list` file helped by the `.deg` file

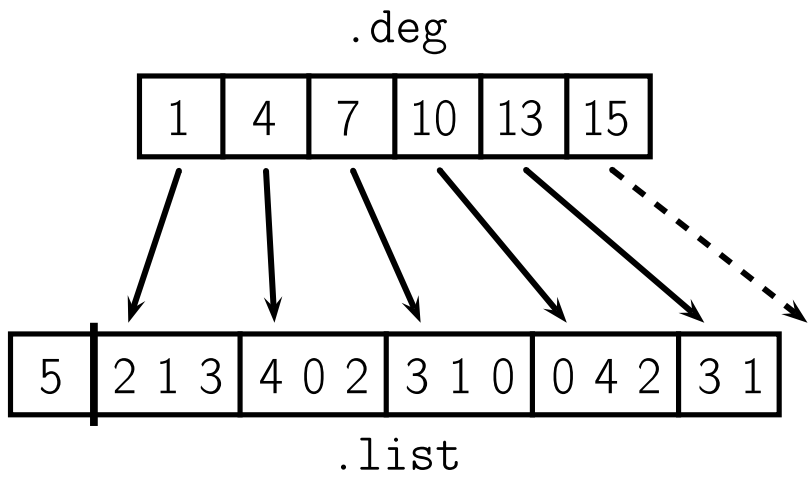
Storage and reading of graphs

With two files

`.list`: list of neighbors of vertices (value n at the beginning) $\rightarrow 2m + 1$ values

`.deg`: *pointers* delimiting the neighborhood of each vertex (degrees computing) $\rightarrow n + 1$ values





- reading of `.list` file helped by the `.deg` file

Description of the six implemented algorithms

Let $G = (V, E)$ be a graph C the cover under construction.

For each vertex $u \in V$,

LR: if $u \notin C$, then $\{v \mid uv \in E \wedge v \notin C\}$ is put in C;

Description of the six implemented algorithms

Let $G = (V, E)$ be a graph C the cover under construction.

For each vertex $u \in V$,

LR: if $u \notin C$, then $\{v \mid uv \in E \wedge v \notin C\}$ is put in C;

ED: if $u \notin C$ and if $\exists v \in N(u) \mid v \notin C$, u and v go to C;

Description of the six implemented algorithms

Let $G = (V, E)$ be a graph C the cover under construction.

For each vertex $u \in V$,

LR: if $u \notin C$, then $\{v \mid uv \in E \wedge v \notin C\}$ is put in C;

ED: if $u \notin C$ and if $\exists v \in N(u) \mid v \notin C$, u and v go to C;

S-Pitt: if $u \notin C$ and if $\exists v \in N(u) \mid v \notin C$, either u or v is put in C with probability $\frac{1}{2}$;

Description of the six implemented algorithms

Let $G = (V, E)$ be a graph C the cover under construction.

For each vertex $u \in V$,

LR: if $u \notin C$, then $\{v \mid uv \in E \wedge v \notin C\}$ is put in C;

ED: if $u \notin C$ and if $\exists v \in N(u) \mid v \notin C$, u and v go to C;

S-Pitt: if $u \notin C$ and if $\exists v \in N(u) \mid v \notin C$, either u or v is put in C with probability $\frac{1}{2}$;

LL: u is put in C iff $\exists v \in N(u)$ such that $v > u$;

Description of the six implemented algorithms

Let $G = (V, E)$ be a graph C the cover under construction.

For each vertex $u \in V$,

LR: if $u \notin C$, then $\{v \mid uv \in E \wedge v \notin C\}$ is put in C;

ED: if $u \notin C$ and if $\exists v \in N(u) \mid v \notin C$, u and v go to C;

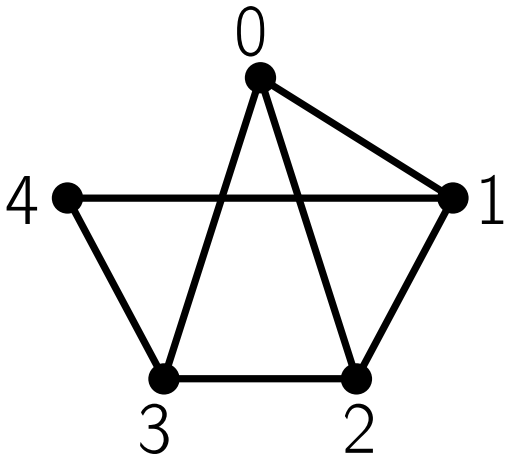
S-Pitt: if $u \notin C$ and if $\exists v \in N(u) \mid v \notin C$, either u or v is put in C with probability $\frac{1}{2}$;

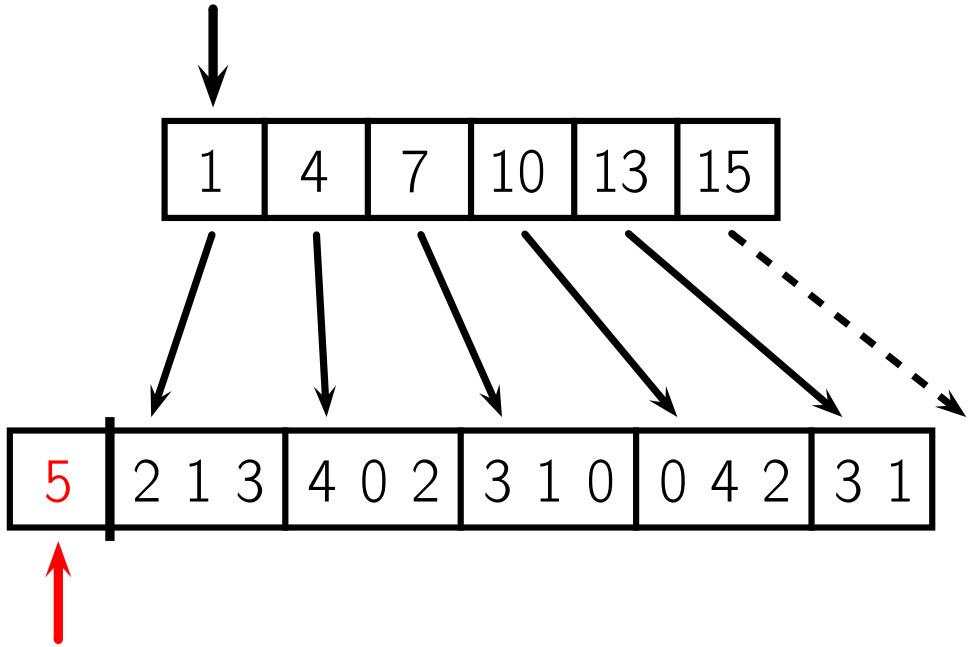
LL: u is put in C iff $\exists v \in N(u)$ such that $v > u$;

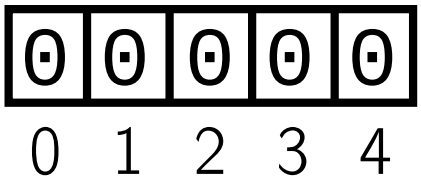
SLL: u is put in C iff $\exists v \in N(u)$ such that $d(v) < d(u)$ or $d(v) = d(u)$ and $v > u$;

ASLL: u is put in C iff $\exists v \in N(u)$ such that $d(v) > d(u)$ or $d(v) = d(u)$ and $v < u$.

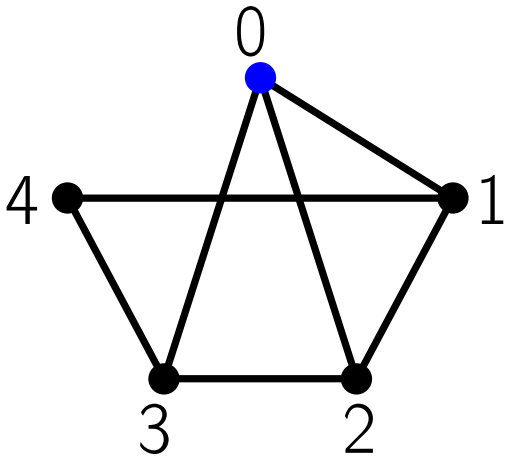
Example of execution of algorithm LR

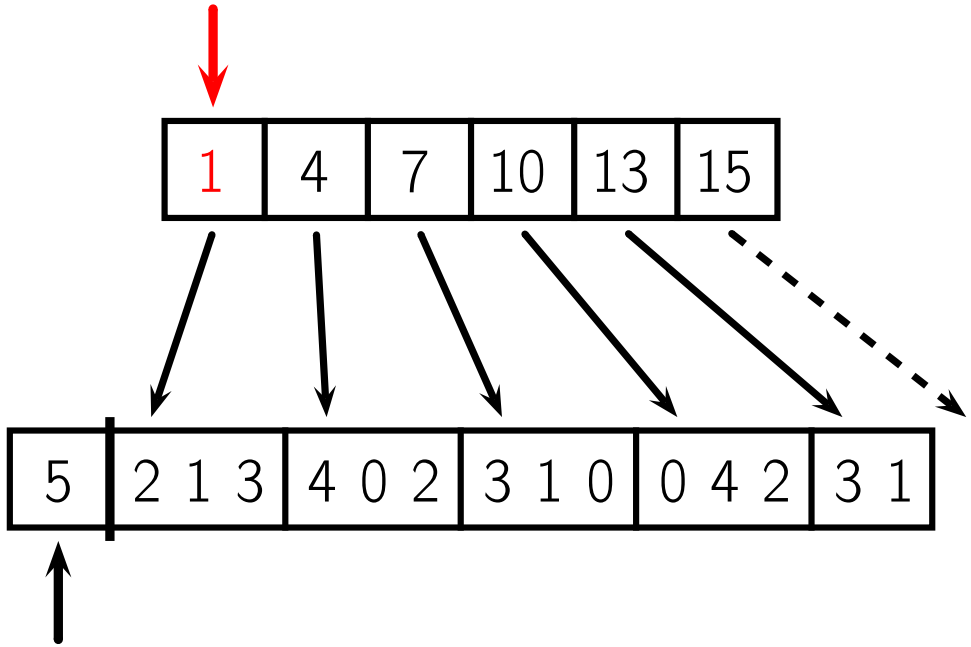


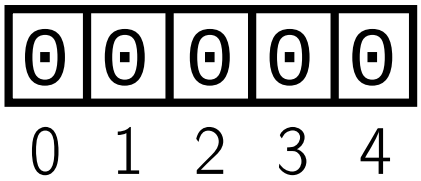




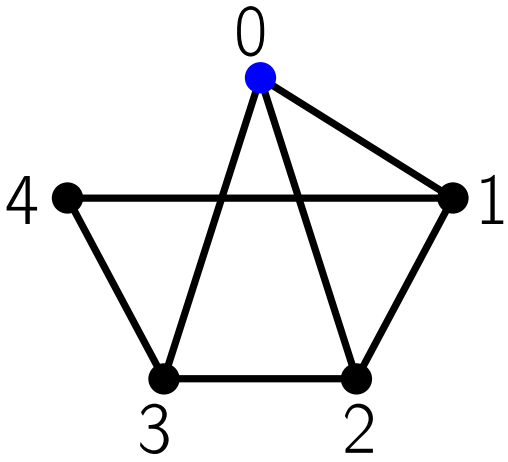
Example of execution of algorithm LR

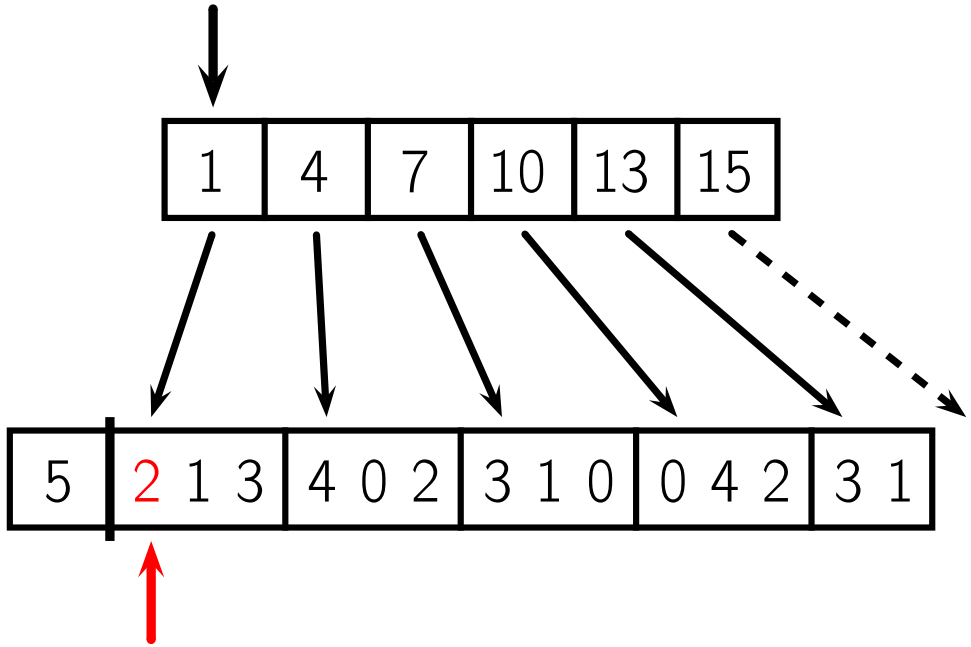


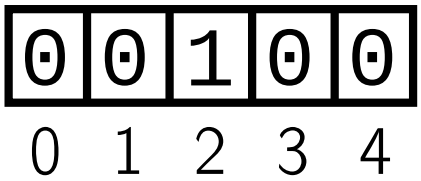




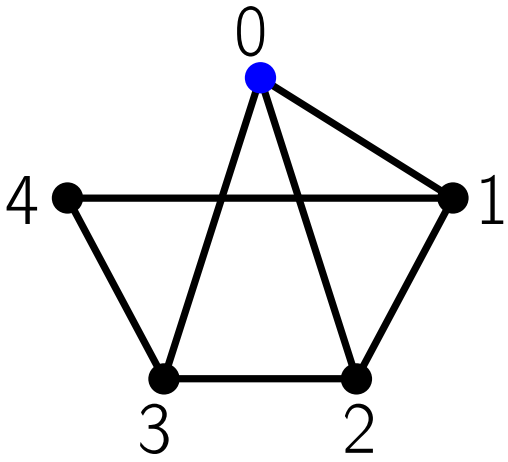
Example of execution of algorithm LR

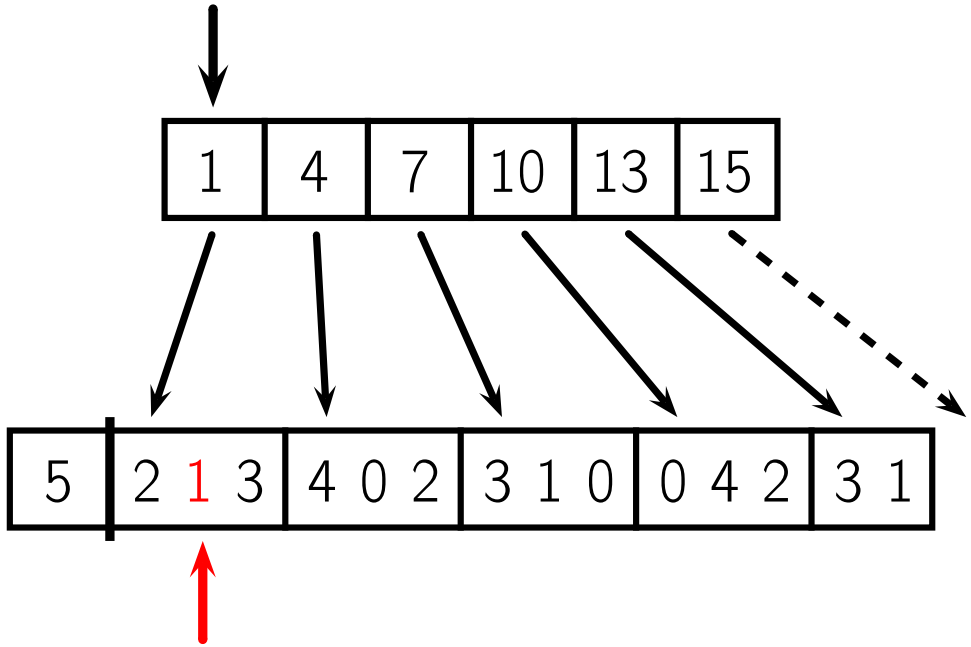


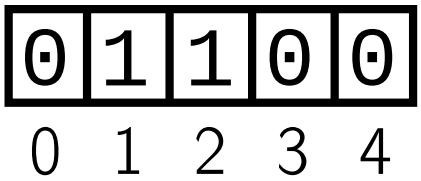




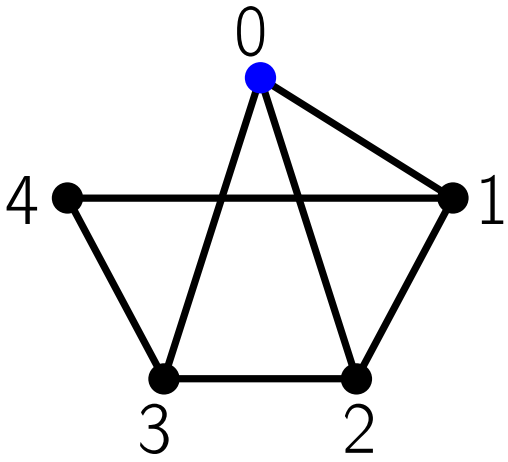
Example of execution of algorithm LR

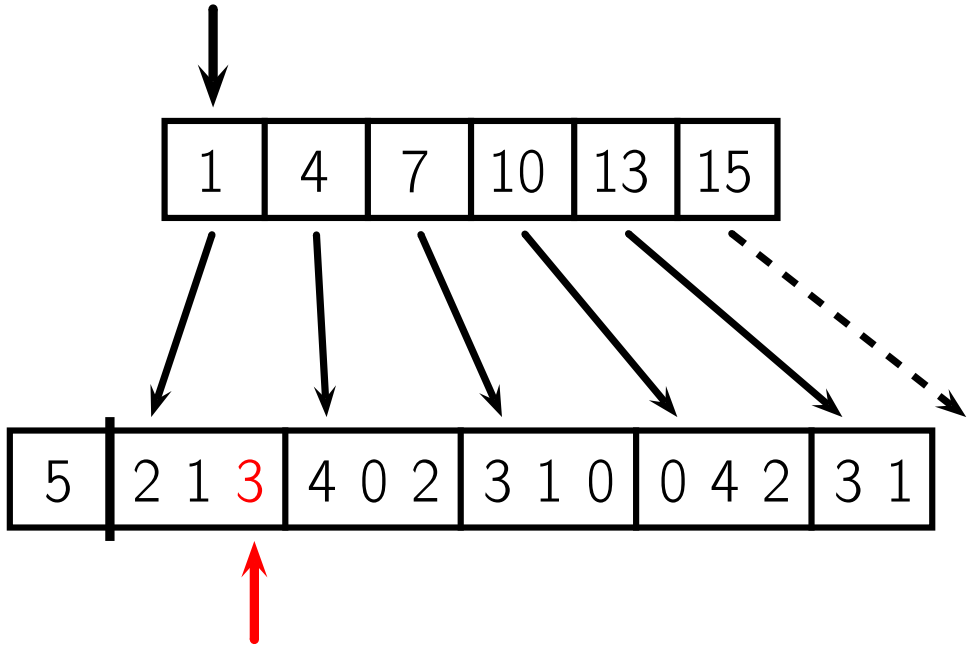


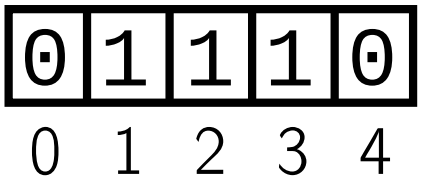




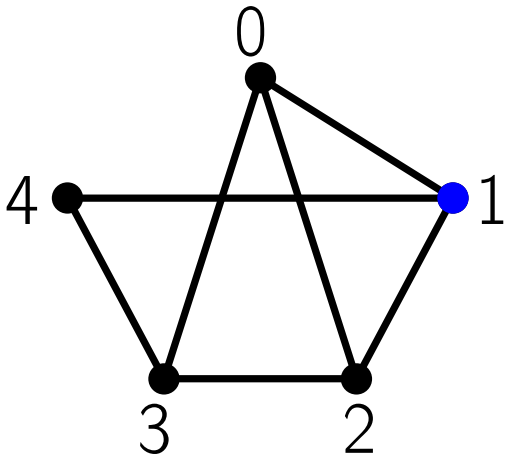
Example of execution of algorithm LR

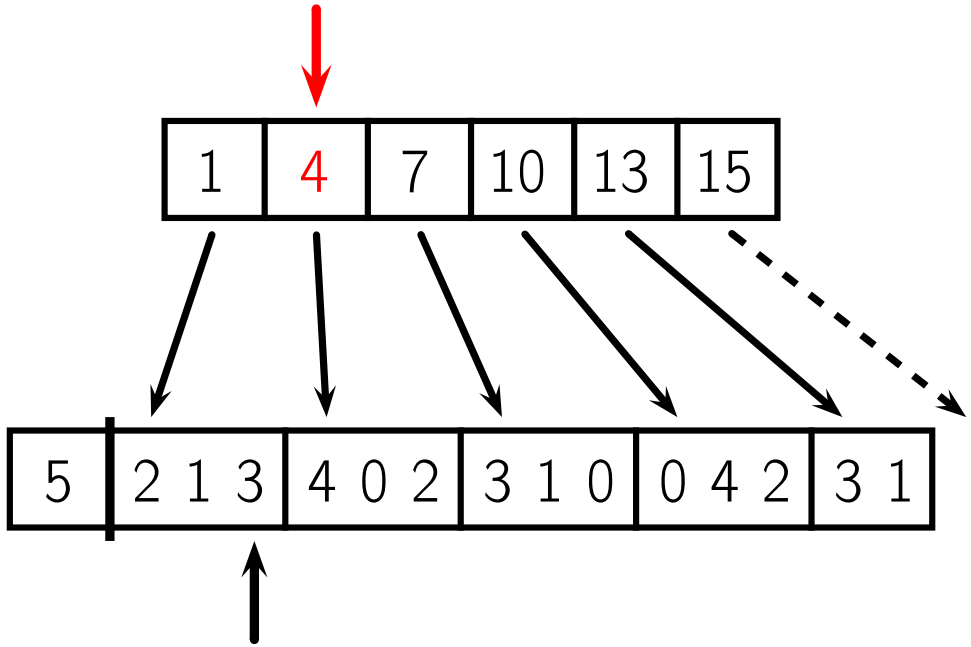


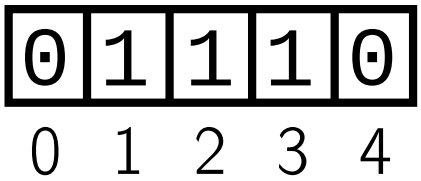




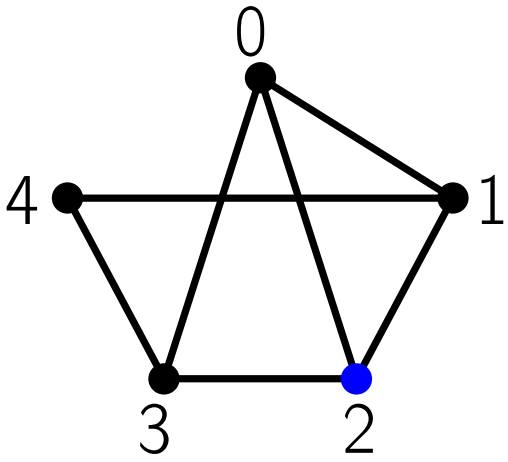
Example of execution of algorithm LR

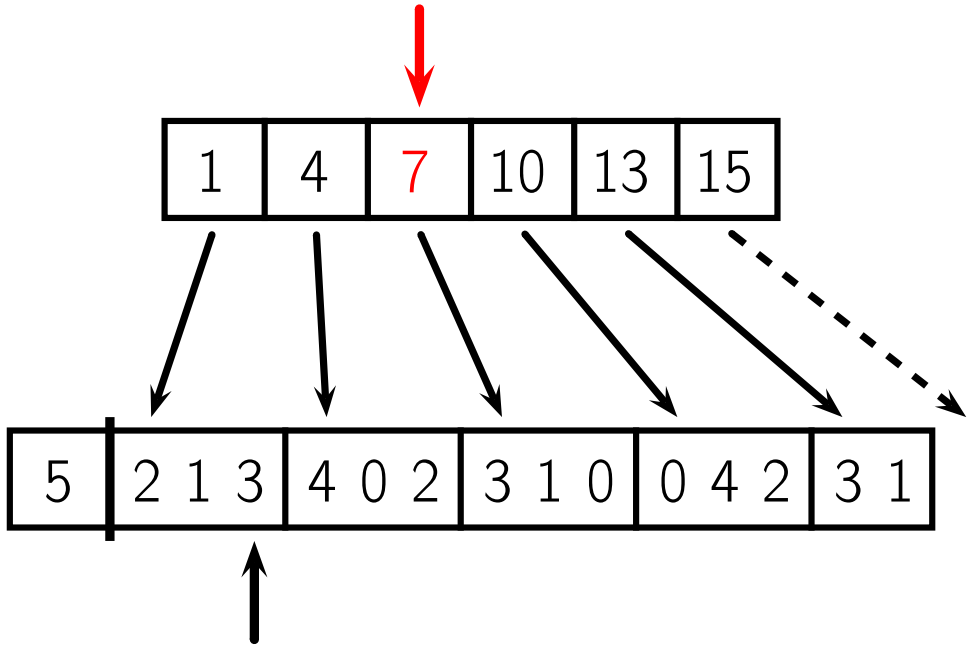


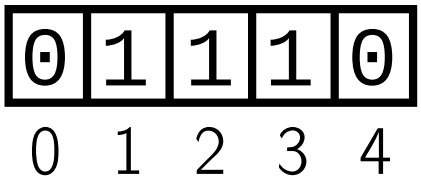




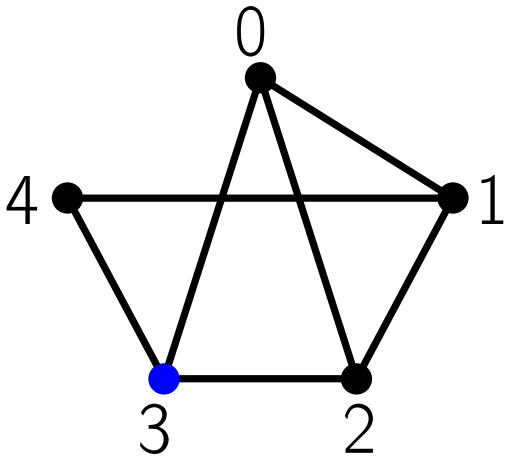
Example of execution of algorithm LR

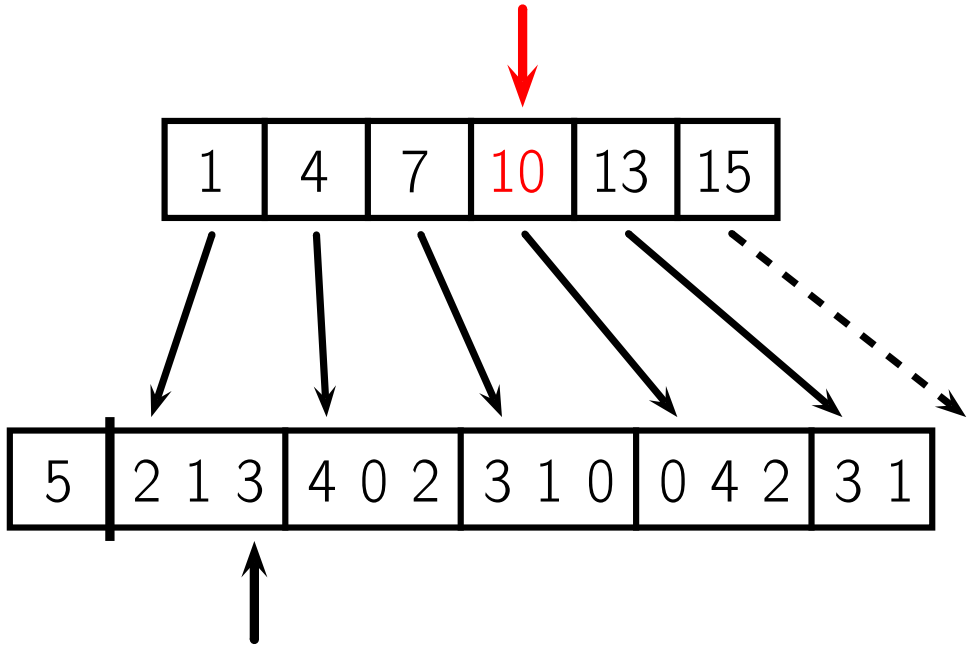


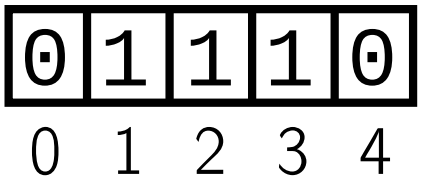




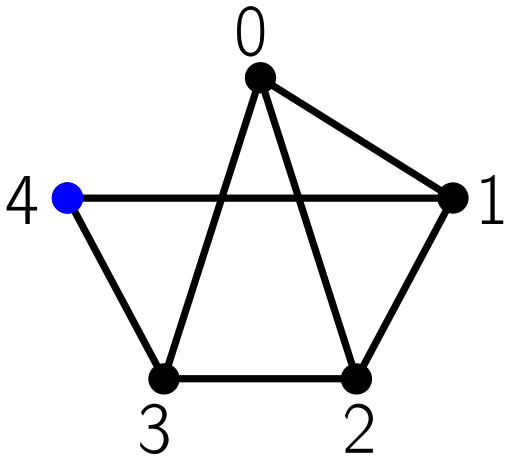
Example of execution of algorithm LR

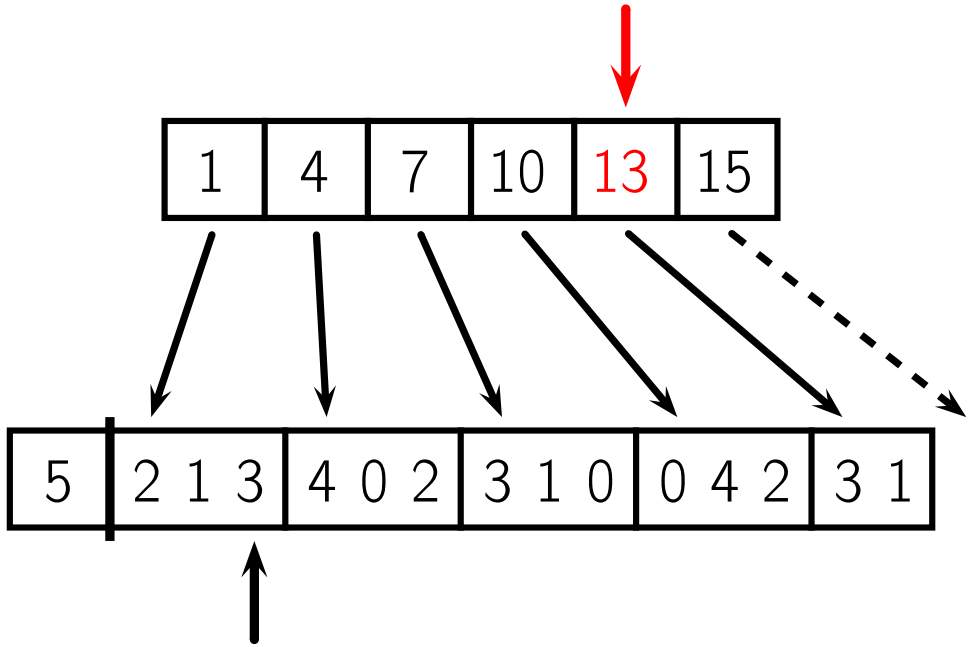


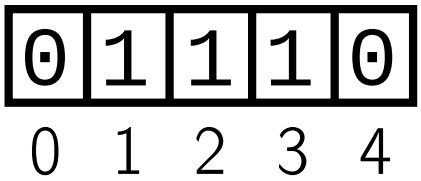




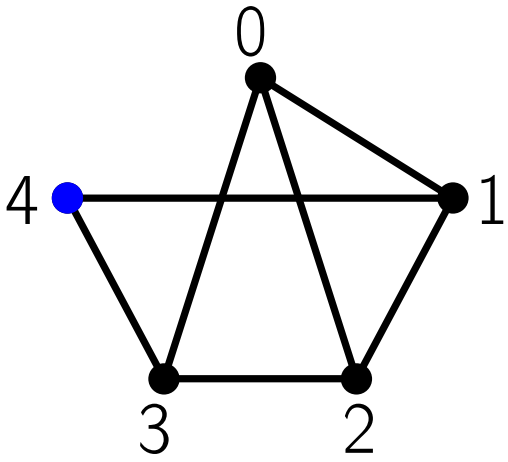
Example of execution of algorithm LR

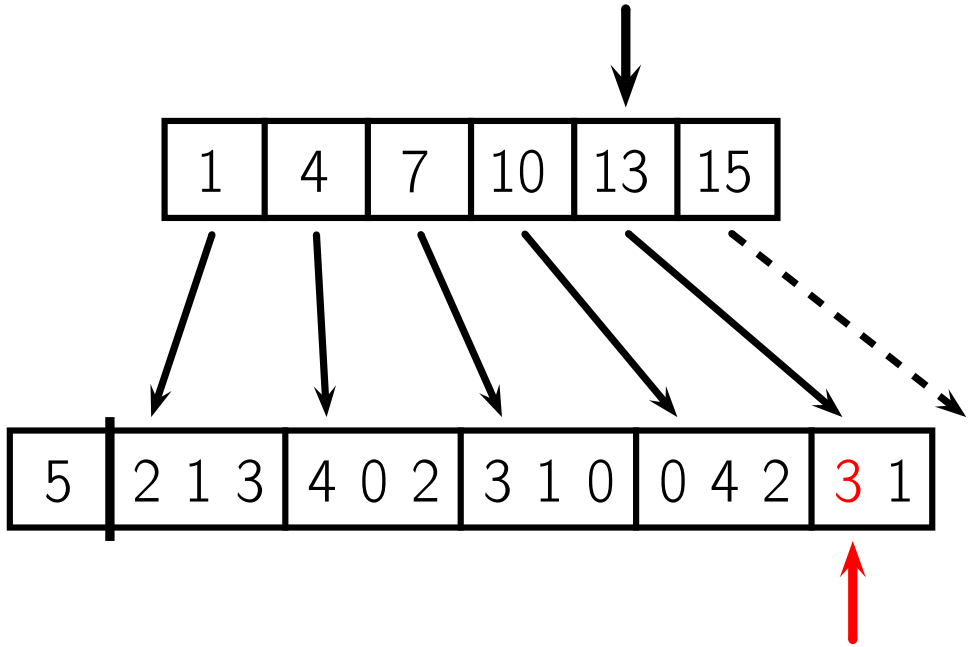


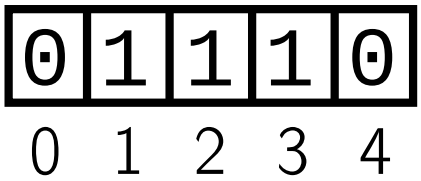




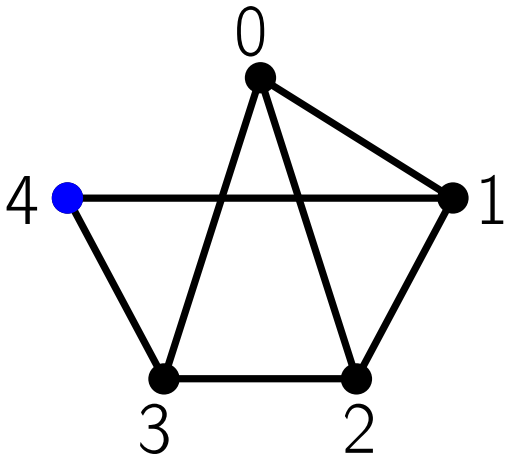
Example of execution of algorithm LR

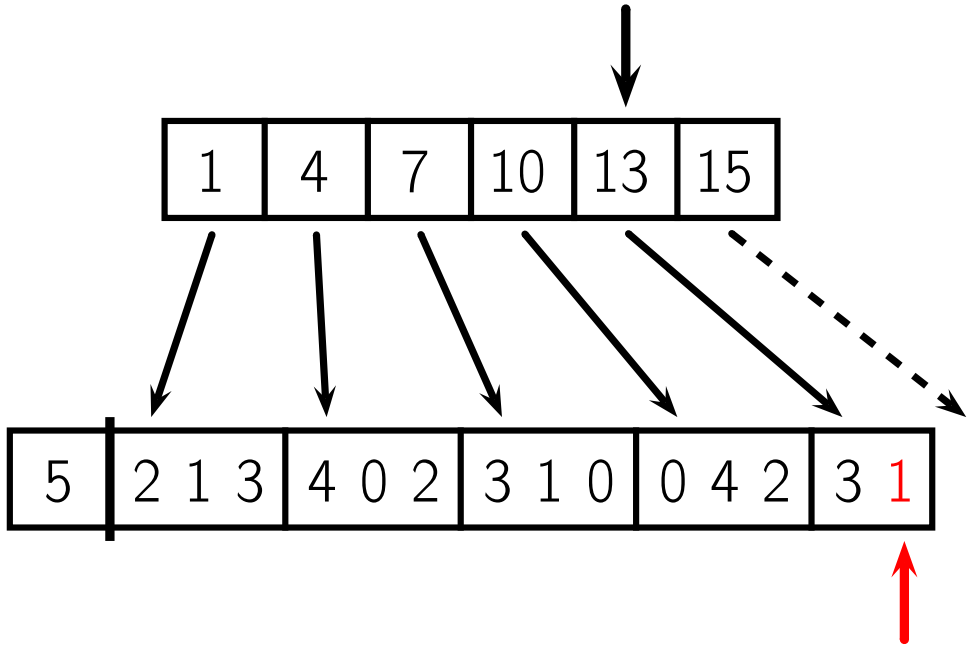


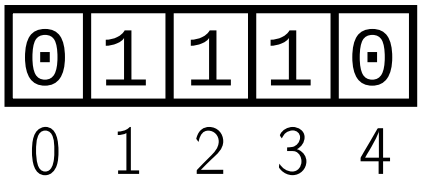




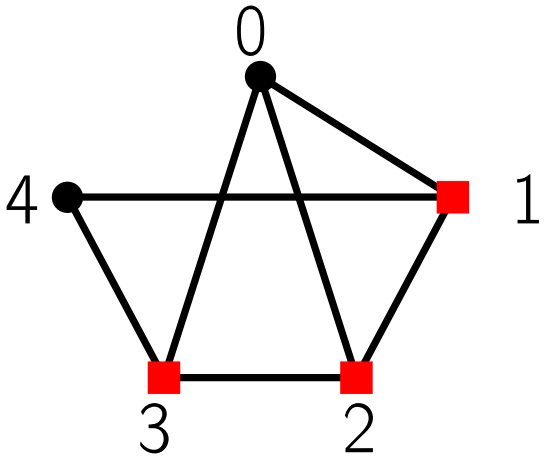
Example of execution of algorithm LR

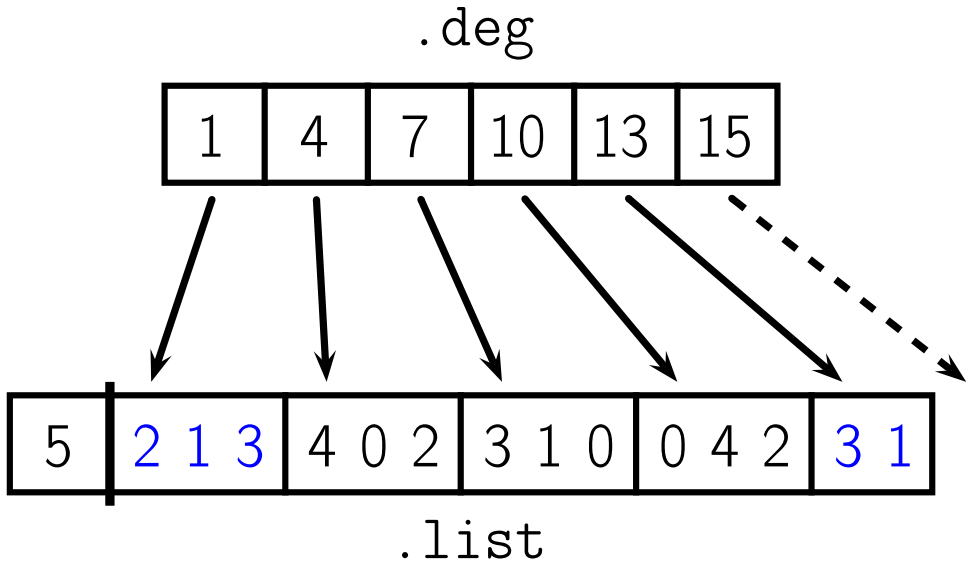






Example of execution of algorithm LR





Cover produced on disk: 2, 1, 3

Number of vertices scanned in the `.list` file: five (over 14)

→ We can "step over" neighbors in the `.list` file.

Algorithms properties

Technical properties

1. `LR`, `ED` and `S-Pitt` need to allocate an n bits array.
2. `SLL` and `ASLL` need to compute degrees of neighbors.

Algorithms properties

Technical properties

1. `LR`, `ED` and `S-Pitt` need to allocate an n bits array.
2. `SLL` and `ASLL` need to compute degrees of neighbors.

Approximation ratio

`LR`: Δ

`ED`: 2 (matching algorithm)

`S-Pitt`: 2 *in expectation*

L. Pitt: *A Simple Probabilistic Approximation Algorithm for Vertex Cover*, Technical Report 404, Yale, 1985

`SLL`: $\frac{\sqrt{\Delta}}{2} + \frac{3}{2}$

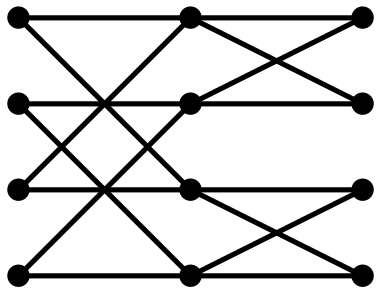
D. Avis et al.: *A List Heuristic for Vertex Cover*, ORL 2006

`LL` and `ASLL`: at least Δ

Graph families used

Sparse graphs (where $m \in \mathcal{O}(n)$):

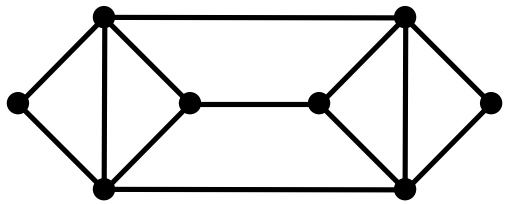
- *ButterFly*
- *de Bruijn* graphs
- grid graphs



ButterFly of dimension 2

Dense graphs (where $m \in \Theta(n^2)$):

- hypercubes
- complete bipartite graphs
- complete *split graphs*



de Bruijn of dimension 3

Why have we chosen these graphs?

1. They can be easily constructed (*with low memory capacities*).
2. Often, $OPT(G) \leq \frac{n}{2}$

Outline

Preamble

34 instances generated:

1. 13 of size $\approx 200 \cdot 10^6$ (several Gigabytes)
2. 6 of size $\approx 30 \cdot 10^9$ ($>$ 100 Gb)
3. 2 of size $\approx 100 \cdot 10^9$ (about 1.5 Tb)

Also: 13 of size $\approx$ 300,000 (several Mb on disk)

$\rightarrow$ *Random power law graphs* for low size levels

Preamble

34 instances generated:

1. 13 of size $\approx 200 \cdot 10^6$ (several Gigabytes)
2. 6 of size $\approx 30 \cdot 10^9$ ($>$ 100 Gb)
3. 2 of size $\approx 100 \cdot 10^9$ (about 1.5 Tb)

   Also: 13 of size $\approx$ 300,000 (several Mb on disk)

$\rightarrow$ *Random power law graphs* for low size levels

Evaluated criteria:

- quality of solutions
- complexity in *number of requests*

$\rightarrow$ the number of neighbors read in the `.list` file

- CPU running times

Results obtained on sparse graphs

Example with instance `butterfly-28`

n	m	OPT
7,784,628,224	15,032,385,540	3,758,096,384

Size on hard disk: 282 Gb

	Quality ($\times OPT$)	Complexity ($\times m$)	Times (CPU)
LR	**1**	0.99	$\approx$ 1h12
ED	2	**0.61**	$\approx$ 1h16
S-Pitt	1.63	0.91	$\approx$ 1h19
LL	1.93	1.03	$\approx$ 1h21
SLL	1.92	1.01	$\approx$ 5h11
ASLL	2	0.72	$\approx$ 3h39

Results obtained on dense graphs

Example with instance `compbip-35000.500000`

n	m	OPT
535,000	17,500,000,000	35,000

Size on hard disk: 261 Gb

	Quality ($\times OPT$)	Complexity ($\times m$)	Times (CPU)
LR	14.28	1	$\approx$ 24 min
ED	2	0.93	$\approx$ 23 min
S-Pitt	2.01	**0.92**	$\approx$ 22 min
LL	**1**	1.001	$\approx$ 23 min
SLL	**1**	1.001	$\approx$ 6h12
ASLL	14.28	1.002	$\approx$ 6h12

Observations

Quality of solutions

LR is almost always the best.

$\rightarrow$ It often returns an optimal solution.

SLL offers good performance, especially on *random power law graphs*.

Observations

Quality of solutions

`LR` is almost always the best.

→ It often returns an optimal solution.

`SLL` offers good performance, especially on *random power law graphs*.

Global performance of `S-Pitt` and `LL` is intermediate.

→ `LL` "fluctuates" more than `S-Pitt`.

Observations

Quality of solutions

`LR` is almost always the best.

→ It often returns an optimal solution.

`SLL` offers good performance, especially on *random power law graphs*.

Global performance of `S-Pitt` and `LL` is intermediate.

→ `LL` "fluctuates" more than `S-Pitt`.

`ED` and `ASLL` are overall the worst.

→ Approximation ratio of 2 often reached for `ED`

(confirmation of observations made by *F. Delbot et al.*)

- *Analytical and Experimental Comparison of Six Algorithms for the Vertex Cover*, ACM Journal of Experimental Algorithmics, 2010

Observations

Complexity in number of requests

ED is almost always the best.

$\rightarrow$ It performs less than m requests.

Observations

Complexity in number of requests

ED is almost always the best.

→ It performs less than m requests.

LR often reaches m requests.

→ It cannot perform worse!

Better ($< m$) when it returns bad solution.

LL, SLL et ASLL can perform more than m requests.

→ ASLL better on complete split graphs!

Observations

Complexity in number of requests

ED is almost always the best.

$\rightarrow$ It performs less than m requests.

LR often reaches m requests.

$\rightarrow$ It cannot perform worse!

Better ($< m$) when it returns bad solution.

LL, SLL et ASLL can perform more than m requests.

$\rightarrow$ ASLL better on complete split graphs!

Correlation between *quality of solutions* and *number of requests*!

Observations

CPU running times

`SLL` and `ASLL` can be longer.

$\rightarrow$ Computing degrees of retrieved neighbors

Observations

CPU running times

`SLL` and `ASLL` can be longer.

$\rightarrow$ Computing degrees of retrieved neighbors

On sparse graphs (where $m \in \mathcal{O}(n)$), partially influenced by:

- quality of solutions *written*
- number of requests produced

On dense graphs, influenced by number of requests

Observations

CPU running times

`SLL` and `ASLL` can be longer.

$\rightarrow$ Computing degrees of retrieved neighbors

On sparse graphs (where $m \in \mathcal{O}(n)$), partially influenced by:

- quality of solutions *written*
- number of requests produced

On dense graphs, influenced by number of requests

Another technical aspects involved:

- hard drive access times, buffers management, etc.

$\rightarrow$ characteristics highlighted in the *I/O-efficient* model!

Limits encountered

On instance `compbip-250000.380000`, of size:

- $n = 630{,}000$ and $m = 95{,}000{,}000{,}000$ (1.41 Tb on disk),

we can run all the algorithms (except `SLL` and `ASLL`).

Limits encountered

On instance `compbip-250000.380000`, of size:

- $n = 630{,}000$ and $m = 95{,}000{,}000{,}000$ (1.41 Tb on disk),

we can run all the algorithms (except `SLL` and `ASLL`).

But, on instance `butterfly-30`:

n	m
33,285,996,544	64,424,509,440

Size on disk: 1.17 Tb

$\rightarrow$ only `LL` can be run on our computer (4 Gb RAM)!

(CPU running times $\approx$ 5h48)

Outline

Conclusion

Despite its performance, the *well-adapted* algorithm is LL:

- no need to compute degrees of neighbors
- no need to allocate an n bits array

Conclusion

Despite its performance, the *well-adapted* algorithm is LL:

- no need to compute degrees of neighbors
- no need to allocate an n bits array

Moreover, it has interesting properties.

1. Easily parallelizable $\rightarrow$ CPU running times improvement
2. $\forall$ graph, $\exists$ a vertex labeling such that LL returns OPT

Conclusion and perspectives

Despite its performance, the *well-adapted* algorithm is LL:

- no need to compute degrees of neighbors
- no need to allocate an n bits array

Moreover, it has interesting properties.

1. Easily parallelizable $\rightarrow$ CPU running times improvement
2. $\forall$ graph, $\exists$ a vertex labeling such that LL returns OPT

We could then:

- run it several times (with different vertex labelings)

$\rightarrow$ improve quality of its results

Thanks!

Questions?