Norges miljø- og biovitenskapelige universitet

Institutt for datavitenskap

Digitalisering på Ås

INF205

Resource-efficient programming

4 Production

4.1 Image compression topic
4.2 Subgraph matching
4.3 Groups 2 and 26
4.4 Hard spheres

9th November 2022

Scheduling project presentations

Norwegian University of Life Sciences

pixels to circles

Five groups (+ two from topic 4)
15th December, 10.00 – 13.50

paths in labelled graphs

Six groups
1st December, 10.30 – 13.50

minimize overlap between spheres

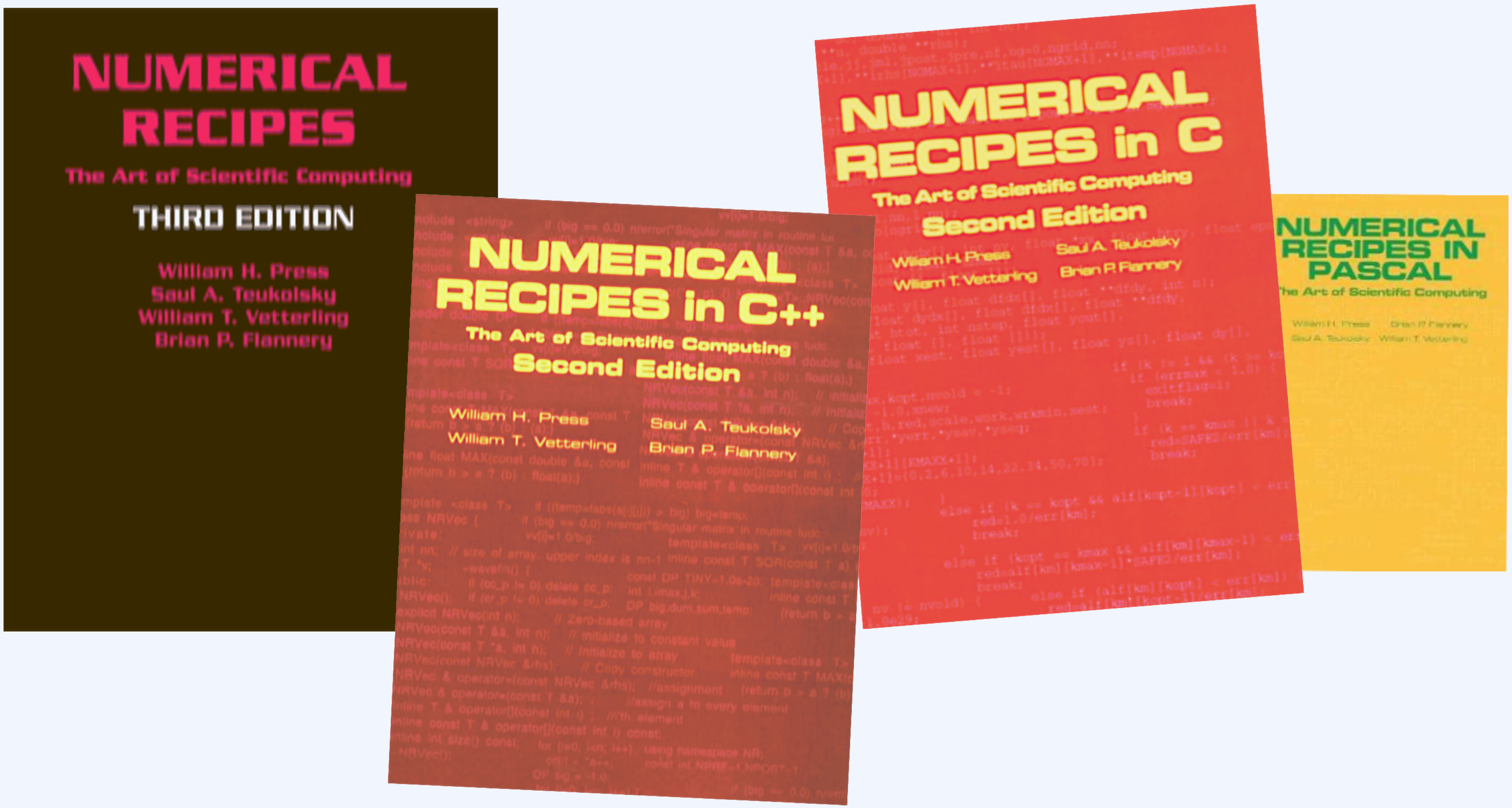
Six groups
8th December, 10.30 – 13.50

binary classifica-tion by ANNs

Two groups (+ five from topic 1)
15th December, 10.00 – 13.50

Norwegian University of Life Sciences

Numerical recipes[1, 2, …] (1985–2007)



[1]http://numerical.recipes/

[2]W. H. Press *et al.*, *Numerical Recipes in C++*, 2nd edn., Cambridge Univ. Press (ISBN 978-0-521-75033-2), **2002**.

Remark about grading

The INF205 grade will in all cases be measured by the same criteria/"aspects". The grade must correspond to competencies in the INF205 subject matter.

Knowledge about algorithms and theories related to the specific topics is interesting and will help us all have interesting discussions and learn together. And hopefully there are some synergies with your other modules and projects.

But in-depth theoretical understanding of the topic will not improve your grade. For that, what matters is that you demonstrate that …

- … you can write clean, readable, object-oriented programs in C++,
- … you can handle memory access and allocation/deallocation safely,
- … you take time and memory efficiency considerations into account,
- … you can exploit parallel/distributed resources, *e.g.*, with MPI or ROS.

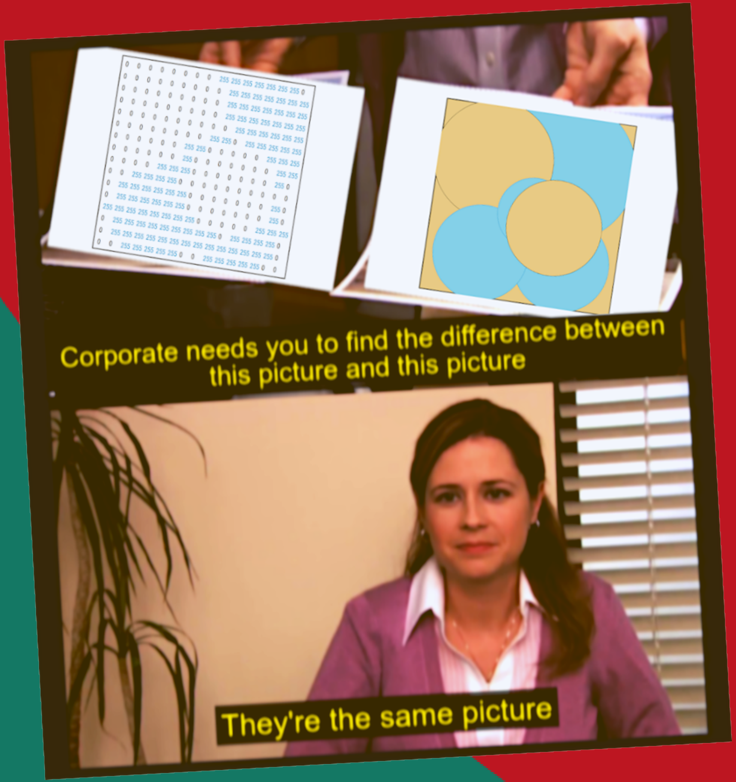
Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

4 C++ basics

4.1 Pixels to circular disks



INF205

9th November 2022

Circular disk vector for a b/w bitmap

0	0	0	0	0	0	255	255	255	255	255	255	255	255	255	0
0	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255
0	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255
0	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255
0	0	0	0	255	255	255	255	255	255	255	255	255	255	255	255
0	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255
0	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255
0	0	0	0	0	255	255	255	255	255	255	255	255	255	255	255
0	0	0	0	0	0	255	255	255	255	255	255	255	255	255	0
0	0	0	0	0	0	0	255	255	255	255	255	255	255	0	0
0	0	0	0	0	0	0	0	0	0	255	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

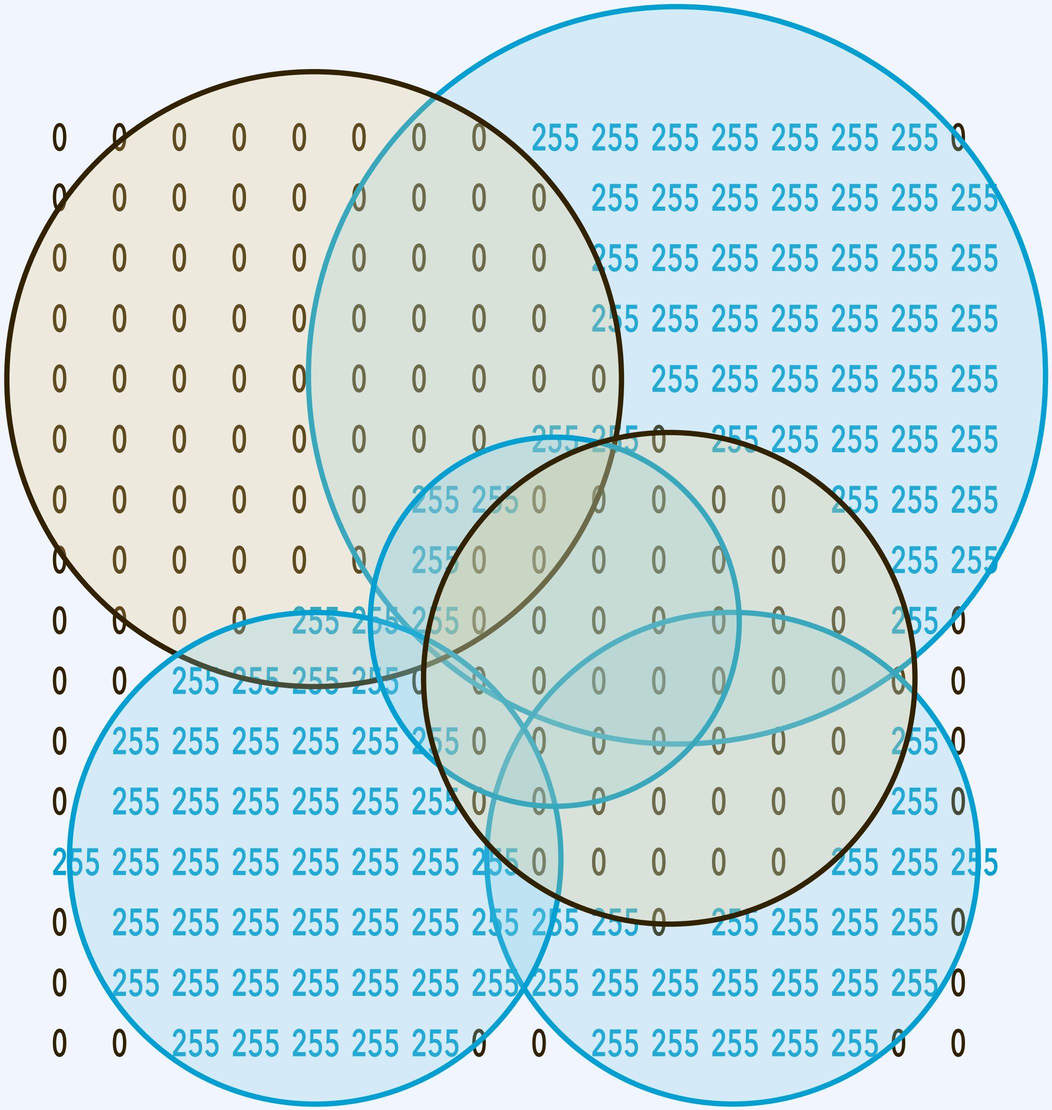
Black background

“white”, r=6, at (10, 4)

disks-to-pixels example

```
unsigned char DiskVector::get_pixel(
  int x, int y
) const
{
  char colour = background;
  for(auto i = disks.begin(); i !=disks.end(); i++)
  {
    float dd = (x - i->x)*(x – i->x)
                              + (y - i->y)*(y - i->y);
    if(i->r * i->r >= dd) colour = i->colour;
  }
  return colour;
}
```

Circular disk vector for a b/w bitmap



Black background

“white”, r=6, at (10, 4)

“black”, r=5, at (4, 4)

“white”, r=4, at (11, 12)

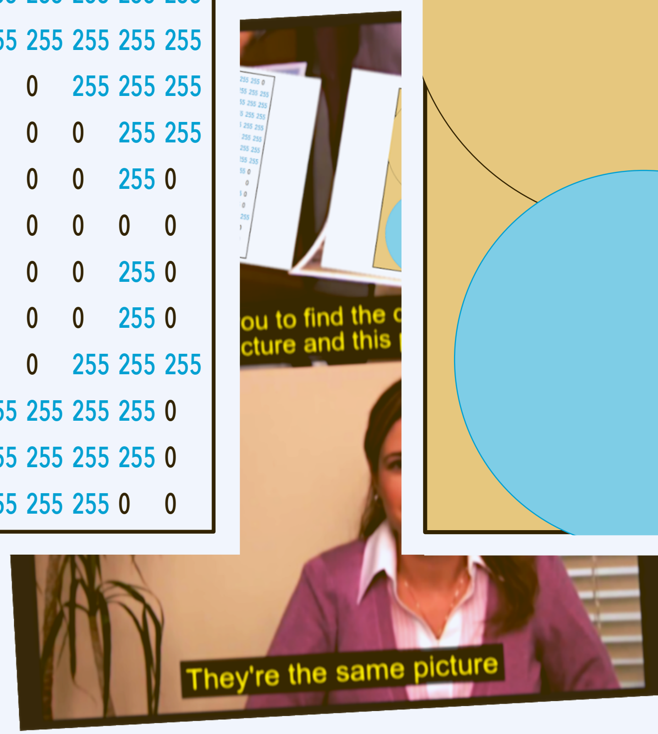
“white”, r=4, at (4, 12)

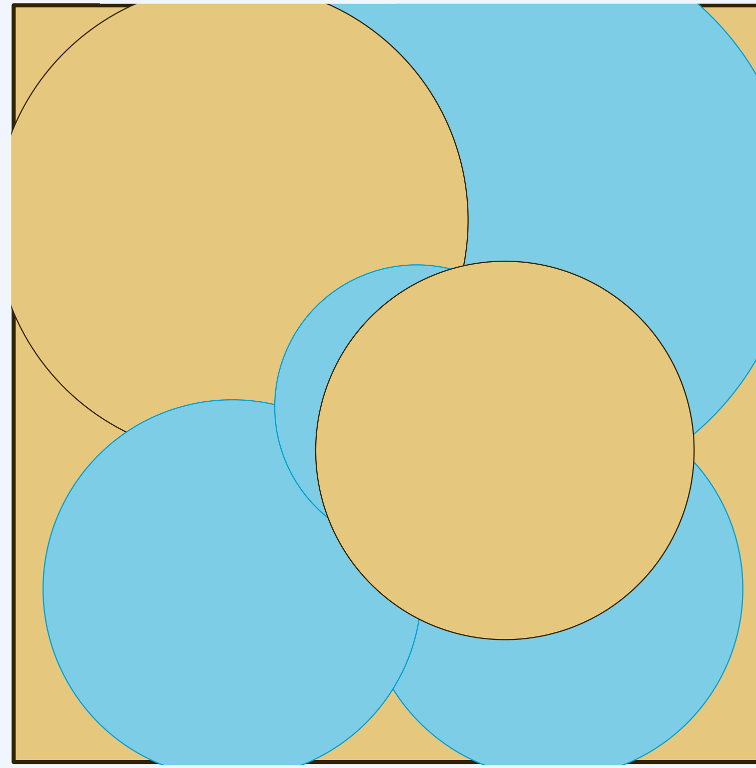
“white”, r=3, at (8, 8)

“black”, r=4, at (10, 9)

“They’re the same picture”

0	0	0	0	0	0	0	0	255	255	255	255	255	255	255	0
0	0	0	0	0	0	0	0	0	255	255	255	255	255	255	255
0	0	0	0	0	0	0	0	0	255	255	255	255	255	255	255
0	0	0	0	0	0	0	0	0	255	255	255	255	255	255	255
0	0	0	0	0	0	0	0	0	0	255	255	255	255	255	255
0	0	0	0	0	0	0	0	255	255	0	255	255	255	255	255
0	0	0	0	0	0	255	255	0	0	0	0	0	255	255	255
0	0	0	0	0	0	255	0	0	0	0	0	0	0	255	255
0	0	0	0	255	255	255	0	0	0	0	0	0	0	255	0
0	0	255	255	255	255	0	0	0	0	0	0	0	0	0	0
0	255	255	255	255	255	255	0	0	0	0	0	0	0	255	0
0	255	255	255	255	255	255	0	0	0	0	0	0	0	255	0
255	255	255	255	255	255	255	255	0	0	0	0	0	255	255	255
0	255	255	255	255	255	255	255	255	255	0	255	255	255	255	0
0	255	255	255	255	255	255	255	255	255	255	255	255	255	255	0
0	0	255	255	255	255	255	0	0	255	255	255	255	255	0	0





16 x 16 b/w values

could be 32 byte

6 x 3 floats
and 7 b/w values

could be 73 byte

Variants of the problem

- What if we allow the **vector-graphics outcome** to not only have black-and-white values, but values from 0 to 255 that can be interpreted …
  - as **greyscale** values, and we aim to minimize square deviations;
  - or as a **probability** (prediction) for black/white, like for **topic 4**.
- Or if the disks are not completely covering what is below, but **combine additively**, or with a **transparency** degree, giving us another parameter.
- They might then also not contribute a constant value to the colour in the compressed image, but **with a gradient**, decreasing from inside out.
- If we do any of the above, the **original pixel graphics** might just as well also be a greyscale image, or it could even be an RGB pixel image.
- Then problem can then even be given a **physical interpretation**; more so if we think of a potential generalization from 2D to 3D: Then it can be applied to represent a charge distribution by a set of point charges.

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

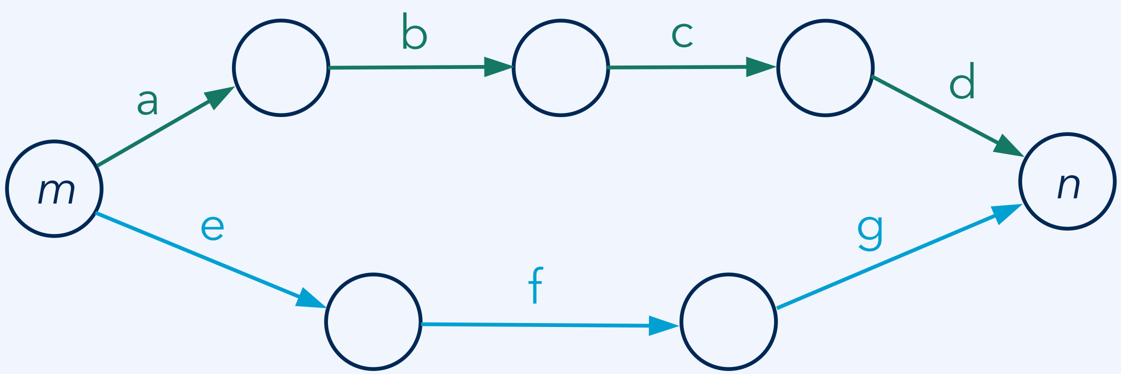
4 Production

4.1 Image compression topic

4.2 Subgraph matching

Suggested topic formulation

Given a graph g and pair of edge-label sequences (p, q), decide whether there is a pair of nodes (m, n) in g such that m is connected to n via a path out of edges labelled $p[0]$, $p[1]$, …, <u>and</u> by a path of edges labelled $q[0]$, $q[1]$, …



$p = \{a, b, c, d\}$

$q = \{e, f, g\}$

Are we considering directed or undirected graphs? For the standard use case, directed graphs are typical. So it could make sense to choose directed graphs.

What if the pattern occurs many times in the graph? It is formulated as a yes-no question (decision problem). In reality, we could be interested in all matches.

More questions and answers

If you have any additional concrete questions (also for any of the other topics), we can keep expanding the Q&A material on the topic website.[1]

- *Should the two paths have the same starting node?* Yes.
- *Is it the idea that the user provides the start node?* No, we search it.
- *Are the labels unique?* Nodes have unique labels, and edges don't.
- *Can the same node occur in both paths?* Yes.
- *What realistic queries are there that take the form of sequences p and q?*
  - "What inventors were killed by their own invention?"[3]
  - "What products are of the same colour as their packaging?"
  - "What ministers had children who became ministers of the same country?"[3]
- *How about parsing the graph data?* (Let us look at the example code.[2])

You can deviate from the "recommended" version and vary the problem a little.

[1] https://home.bawue.de/~horsch/teaching/inf205/project/paths-in-graphs_q-and-a.html
[2] That code is an adaptation of the example from Stroustrup (2018), Section 10.5, *p.* 128.
[3] See https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples/

I/O graph input example

Norwegian University of Life Sciences

```
bool Graph::generate_edge_from(std::istream* source) {
  char single_symbol = '\0';

  // read label of the first node
  std::string a_label = "";
  while(single_symbol != '<')  *source >> single_symbol;   // proceed until '<'
  for(*source >> single_symbol;  single_symbol != '>';  *source >> single_symbol)  a_label += single_symbol;

  if(node_a_label == "")  return false;  // format: empty label means that we are done

  // read string-type edge label
  std::string e_label = "";
  while(single_symbol != '<')  *source >> single_symbol;   // proceed until '<'
  for(*source >> single_symbol;  single_symbol != '>';  *source >> single_symbol)  e_label += single_symbol;

  // read label of the second node
  std::string b_label = "";
  while(single_symbol != '<')  *source >> single_symbol;   // proceed until '<'
  for(*source >> single_symbol;  single_symbol != '>';  *source >> single_symbol)  b_label += single_symbol;

  this->create_edge(a_label, e_label, b_label);
  return true;
}
```

```
void Graph::in(std::istream* source) {
  // read edge by edge
  while( this->generate_edge_from(source) ) { }
}
```

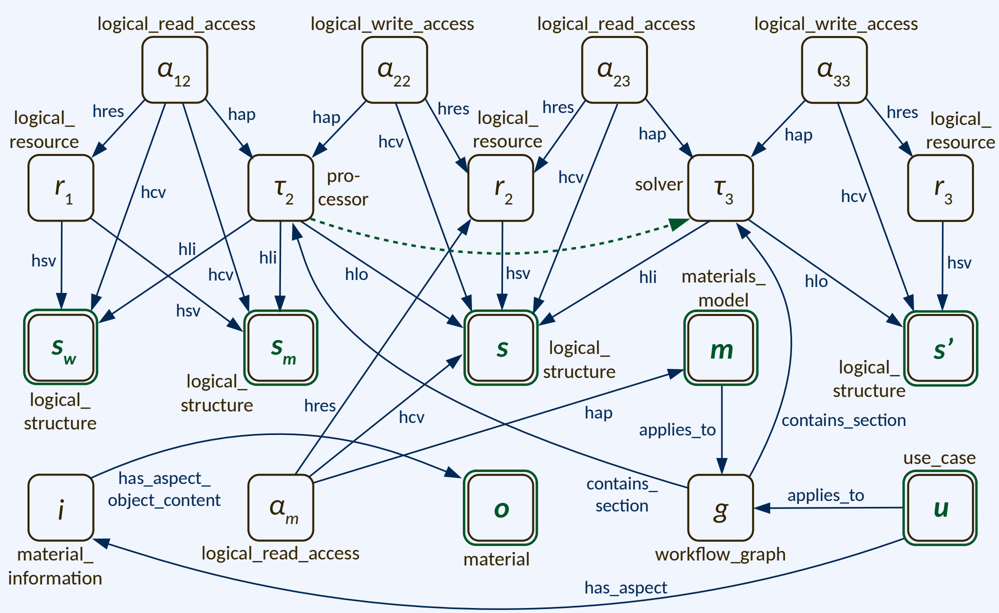
```
// read int-type edge label
int edge_label = 0;
*source >> edge_label;
```

Subgraph matching as querying

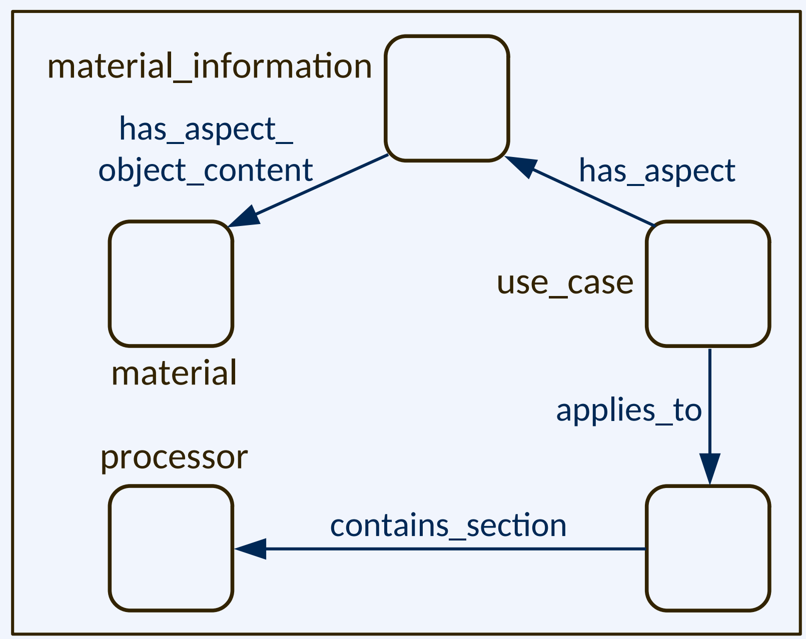
Subgraph matching problem (NP-complete):

Given a graph *G* and a pattern *H*, does G contain a subgraph isomorphic to *H*?

graph *G*



pattern *H*



(example from P. Klein et al., Proceedings of JOWO 2021)

Triples as building blocks

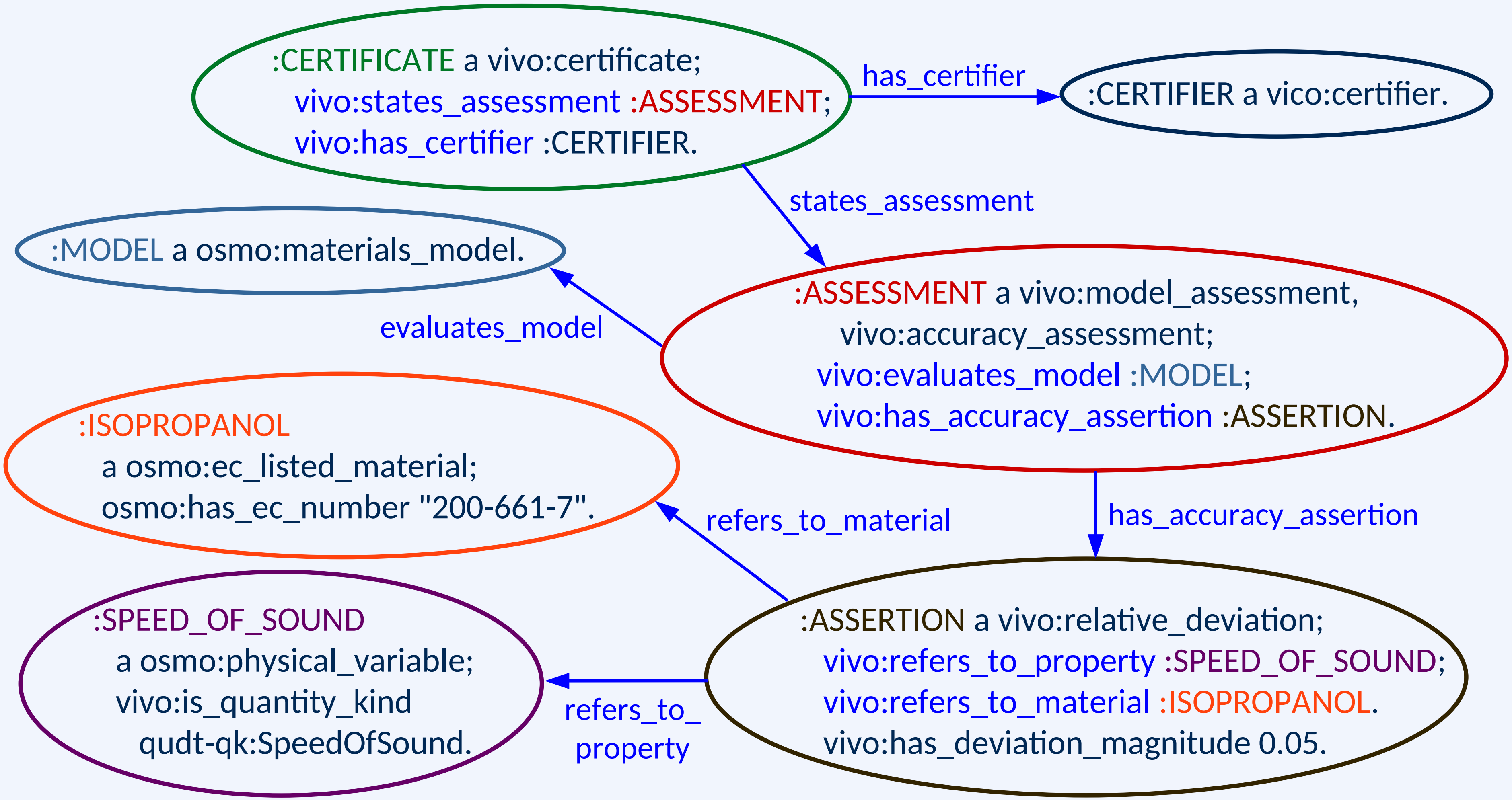
Semantic technology can facilitate the integration of data and software into a coherent framework. Ontologies characterize individuals (*i.e.*, objects), the concepts (*i.e.*, classes) to which they belong, the possible relations between them, and applicable restrictions (rules).

Triples: Individual Relation Individual. (Subject Predicate Object.)

Example: theFox eats theChicken.

- Resource description framework (RDF): Formalism for specifying triples.
- Web ontology language (OWL): Formalism for specifying ontologies, including rules that can be processed by automated reasoning.
- Terse triple language (TTL): Syntax for denoting triples from RDF and OWL.

Knowledge graphs out of triples



Querying a knowledge base: SPARQL

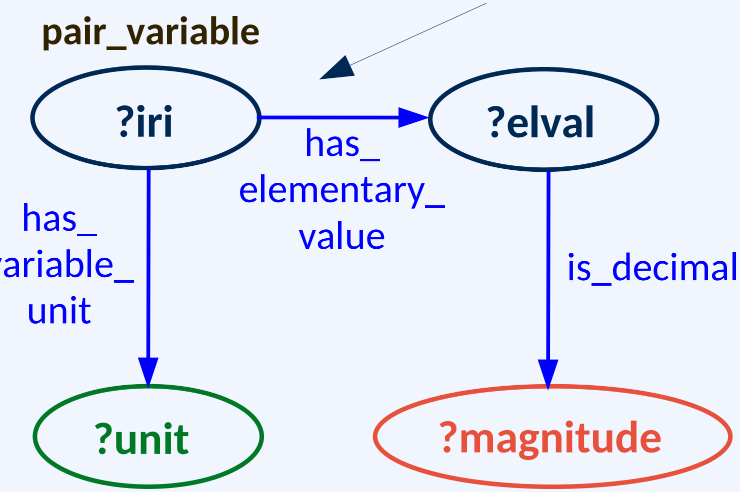
SELECT ?x ?y … WHERE {sequence of triples involving ?x, ?y, …}

selected free variables

there can be additional
(non-selected) free variables

graph pattern

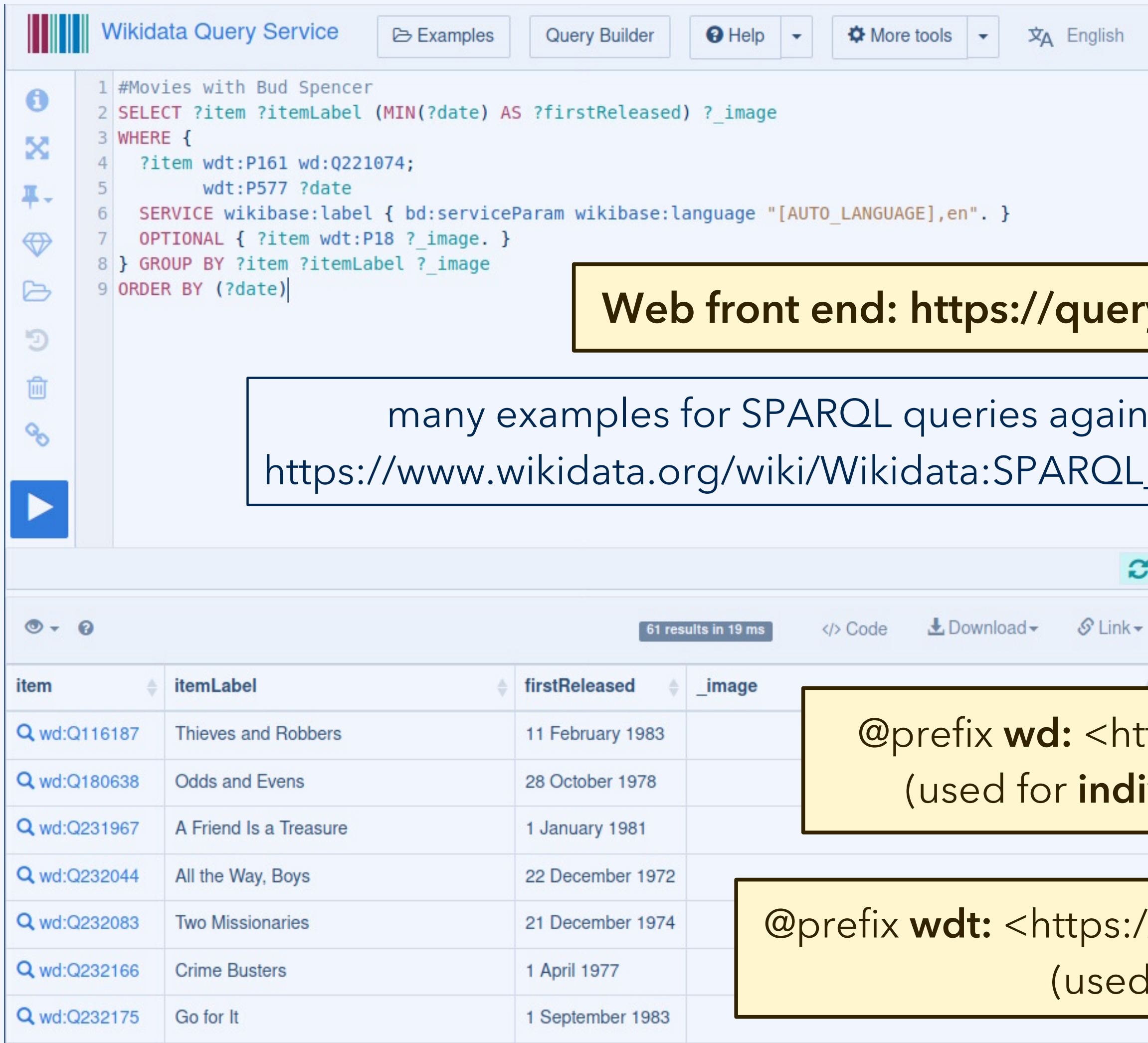
```
SELECT ?magnitude ?unit
WHERE {
  ?iri rdf:type :pair_variable.
  ?iri :has_elementary_value ?elval.
  ?elval :is_decimal ?magnitude.
  ?iri :has_variable_unit ?unit.
}
```



Observation: The WHERE clause consists of RDF triples with free variables.

[1]W3C recommendation, https://www.w3.org/TR/sparql11-query/, **2013**.

SPARQL query examples



Web front end: https://query.wikidata.org/

many examples for SPARQL queries against Wikidata are available at https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples

@prefix **wd:** <https://wikidata.org/wiki/>
(used for **individuals** and **concepts**)

@prefix **wdt:** <https://wikidata.org/wiki/Property:>
(used for **relations**)

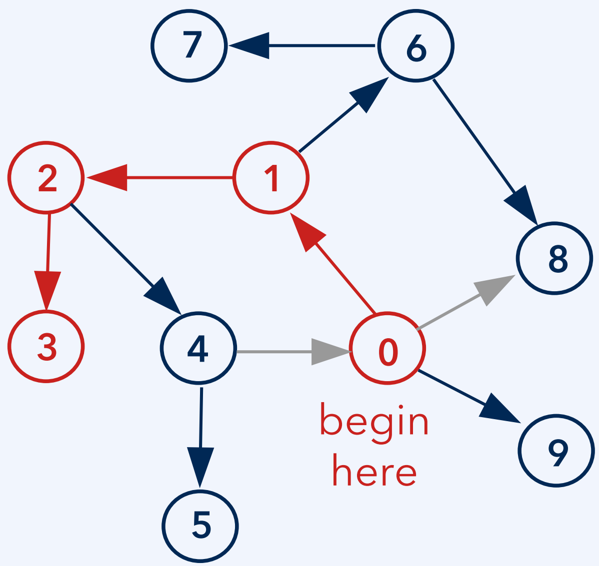
NMBU Norwegian University of Life Sciences

Simple graph traversal algorithms

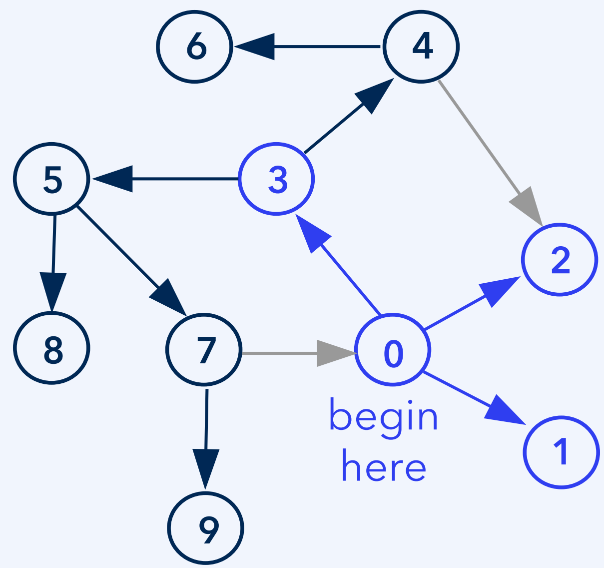
Traversal of trees and graphs: Depth-first search and breadth-first search

DFS always proceeds from the most recently detected node (LIFO).
BFS always proceeds from the node that was detected earliest (FIFO).

depth-first search (DFS)



breadth-first search (BFS)



Note: Only elements *to which there is a path from the initial node* can be found.

Variants of the problem

Other interpretations of the labels or queries:

- Instead of sequences of labels, permit sequences of sets of labels
- … or allow p and q to be regular expressions
- Include some reasoning (*e.g.*, about how distant nodes are connected)
- Numerical labels, the elements of p and q are constraints on labels

Consider other types of queries, given by other graph patterns:

- Search for patterns that are frequent,[1] but still comparably simple
- More complex patterns make the query NP-complete;
- but libraries and algorithms from the literature[2] might be employed

Require distinct nodes/edges, but forget about labels, just checking the shape.

[1]https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples/

[2]L. Kothoff *et al.*, in *Proc. LION 2016*, *pp.* 107–122, doi:10.1007/978-3-319-50349-3_8, **2016**.

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

4 Production

4.1 Image compression topic

4.2 Subgraph matching

4.3 Groups 2 and 26

Neural network parameterization

Solving a **binary classification problem** by an artificial neural network (ANN). Based on statement from group 26:

- **Input:** An *m* x *n* matrix.
- **Output:** A parameterization of the neural network and a vector of probability values generated by that network.

Based on steps formulated by group 2:

- Implement a **perceptron** that can be parameterized/trained.
- Connect perceptrons as a **feed-forward net** with back-propagation.
- Consider concurrency/parallelization.

As an application, the ANN could be a compressed representation of a black-white image. It could even better be used to represent a sequence of images.

Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

4 Production

4.1 Image compression topic

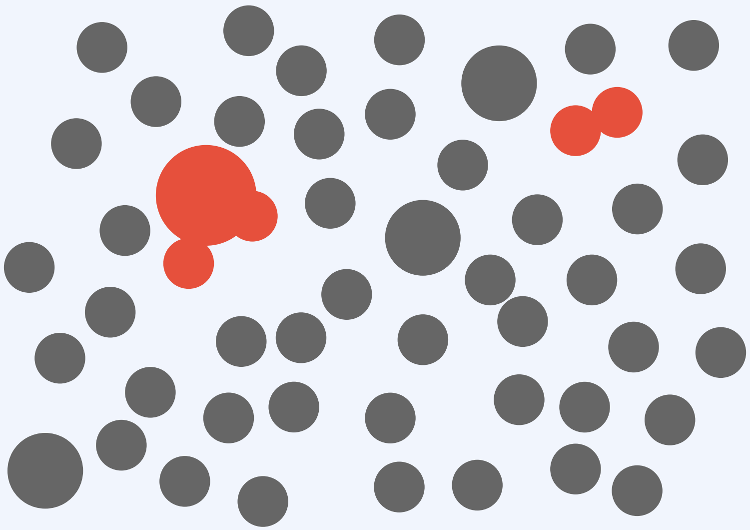
4.2 Subgraph matching

4.3 Groups 2 and 26

4.4 Hard spheres

Standard version of the problem

N spheres of different types (with different radii) are positioned in a 3D system.



Evaluate the number of overlaps.

Rearrange the spheres to make the number of overlaps as small as you can achieve it.

With a low packing fraction (density), this is simple. But it becomes a hard problem with many spheres and a very high packing fraction.

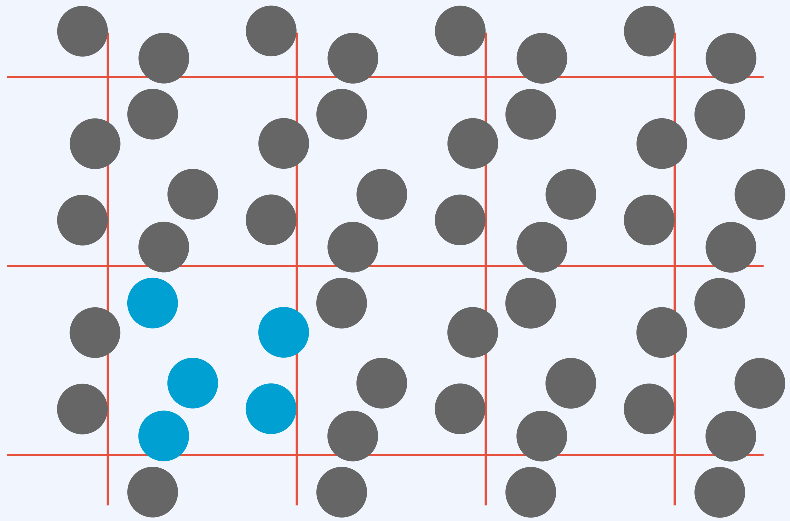
Why is this a challenging optimization problem? Why can naive methods fail in general – in what case can they fail?

What is the dimension of the configuration space for N spheres? How many values can we vary?

Norwegian University of Life Sciences

Box conventions: PBC + MIC

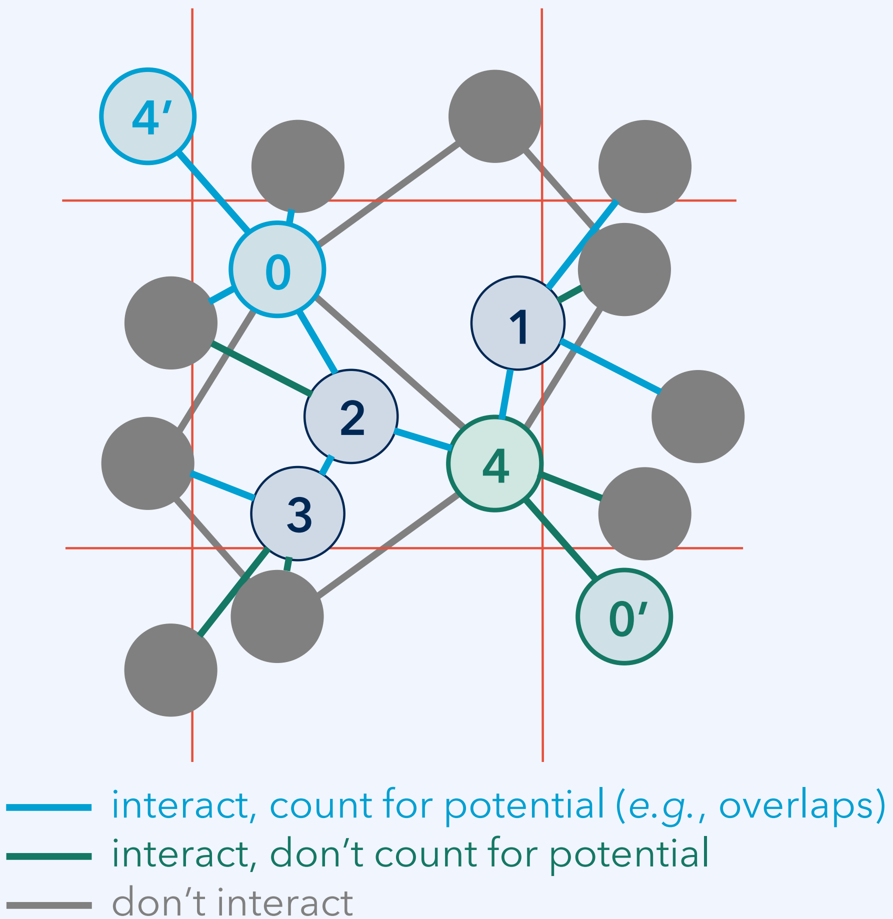
Periodic boundary condition (PBC)



PBC: Assume that the simulation box repeats periodically in all directions.

MIC: Each particle interacts only with closest replica of each other particle.

Minimum image convention (MIC)



Example code (pbc-mic)

Periodic boundary condition (PBC)

Apply modulo(-like) logic, using the box size as the modulo:

```
// now read all the particles
for(size_t i = 0; i < num_part_comp; i++)
{
  // read the coordinates
  double x[3];
  for(int d = 0; d < 3; d++)
  {
    *source >> x[d];

    // apply periodic boundary condition
    if(x[d] < 0.0) x[d] += this->extension[d];
    else if(x[d] > this->extension[d]) x[d] -= this->extension[d];
  }
  ...
}
```

1. **For configuration I/O**
2. **Whenever particles move**

box size

next, read **one** particle with diameter **1.5**

read as (0/0/**1**)

read as (0/3/**3**)

end-of-file token

```
4   4   4

1   1.5
0   1   1

7   1
0   0   0
0   0   5
0   1   0
1   0   0
1   0   1
1   1   0
1   1   1

1   2
0   3   -1

0
```

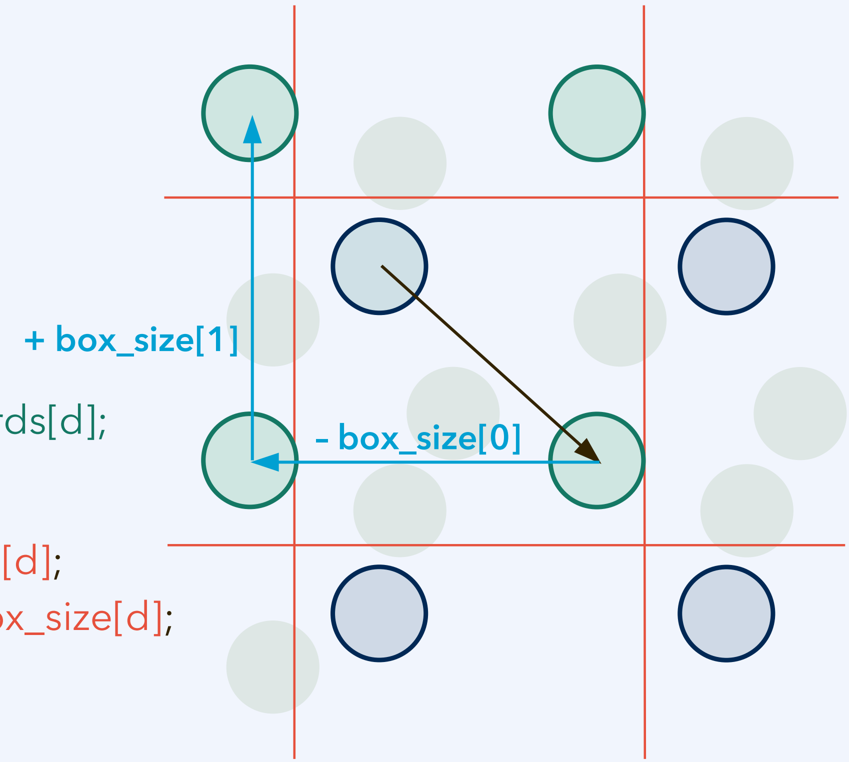
Example code (pbc-mic)

```
bool Sphere::check_collision(
  const Sphere* other, const double box_size[3]
) const
{
  // square distance between "this" and "other"
  double square_distance = 0.0;
  for(int d = 0; d < 3; d++)
  {
    double dist_d = other->coords[d] - this->coords[d];

    // apply minimum image convention
    if(dist_d > 0.5*box_size[d]) dist_d -= box_size[d];
    else if(dist_d < -0.5*box_size[d]) dist_d += box_size[d];

    square_distance += dist_d*dist_d;
  }
  double sum_of_radii = 0.5 * (this->size + other->size);
  return (square_distance < sum_of_radii*sum_of_radii);
}
```

Minimum image convention (MIC)



Norwegian University of Life Sciences

Variants of the problem

The boundary conditions or potential could be varied, *e.g.*, as follows:

- Instead of PBC, use hard walls in one or multiple spatial directions
- Or a spherical box (surrounded by a hard wall[1]), instead of a cuboid
- Non-spherical particles, or multi-centre particles
- More physically relevant interactions with attraction *and* repulsion

Solution techniques/algorithms, from simple to very advanced, may include:

- Monte Carlo (MC) method
  - In Numerical Recipes 2nd edn.,[2] see Sections 7.6 and 7.8
- simulated annealing (MC with varying temperature)
  - In Numerical Recipes 2nd edn.,[2] see Section 10.9
- Other global optimization techniques for complex problems
  - Population-based methods such as evolutionary algorithms
  - Biased sampling methods

Dynamic simulation, integrating the classical mechanical equations of motion.

[1] I. Urrutia, "Bending rigidity and higher […]," *Phys. Rev. E* **89**: 032122, arXiv:1311.5176 [cond-mat.soft], **2014**.

[2] W. H. Press *et al.*, *Numerical Recipes in C++*, 2nd edn., Cambridge Univ. Press (ISBN 978-0-521-75033-2), **2002**.

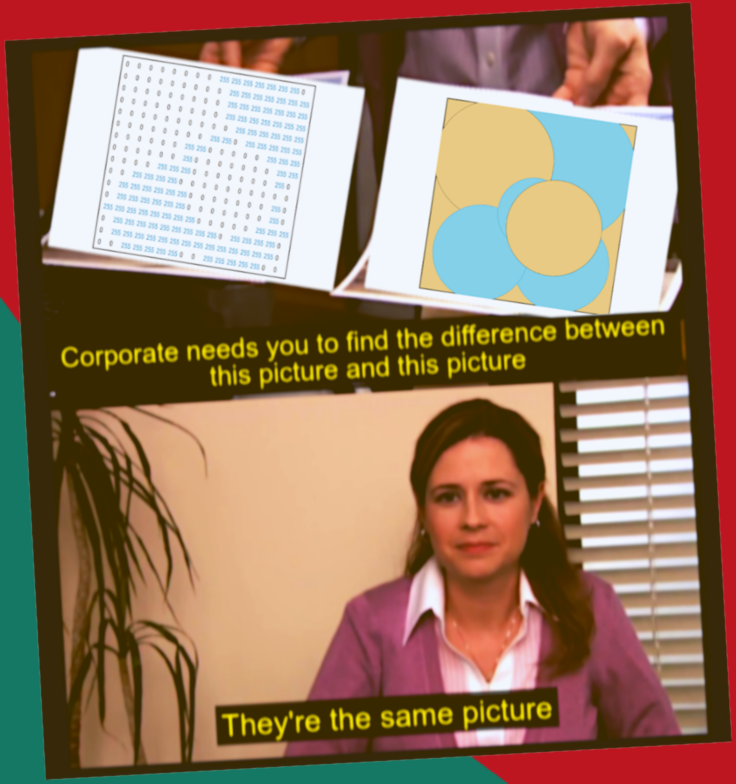
Noregs miljø- og biovitskaplege universitet

Institutt for datavitskap

Digitalisering på Ås

Conclusion

pixels to circles	paths in labelled graphs
Five groups (+ two from topic 4) 15th December, 10.00 – 13.50	Six groups 1st December, 10.30 – 13.50
minimize overlap between spheres	**binary classification by ANNs**
Six groups 8th December, 10.30 – 13.50	Two groups (+ five from topic 1) 15th December, 10.00 – 13.50



Norges miljø- og biovitenskapelige universitet

Institutt for datavitenskap

Digitalisering på Ås

INF205
Resource-efficient programming

4 Production

4.1 Image compression topic

4.2 Subgraph matching

4.3 Groups 2 and 26

4.4 Hard spheres

9th November 2022