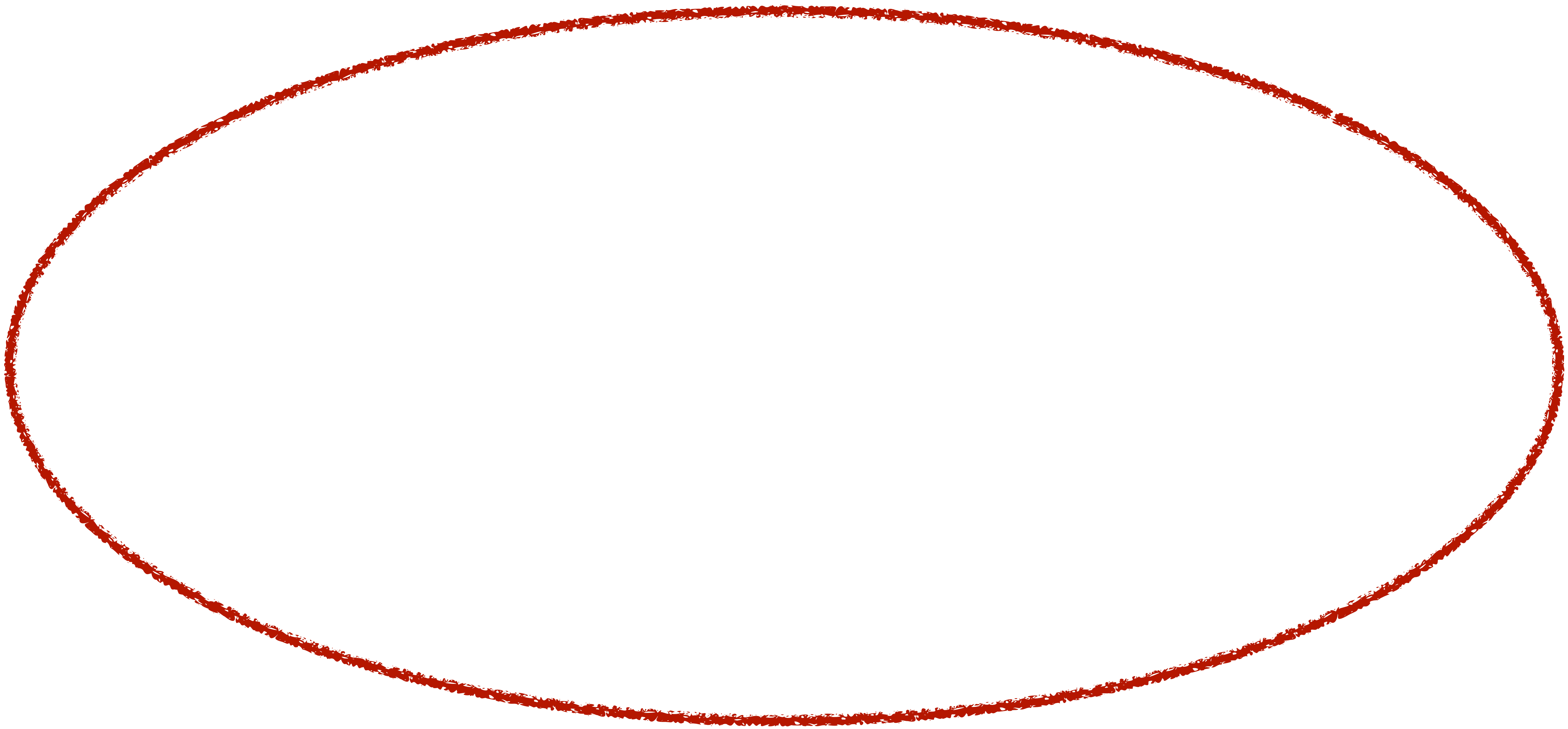
Introduction to Algorithms and Data Structures

NP-completeness: A closer look

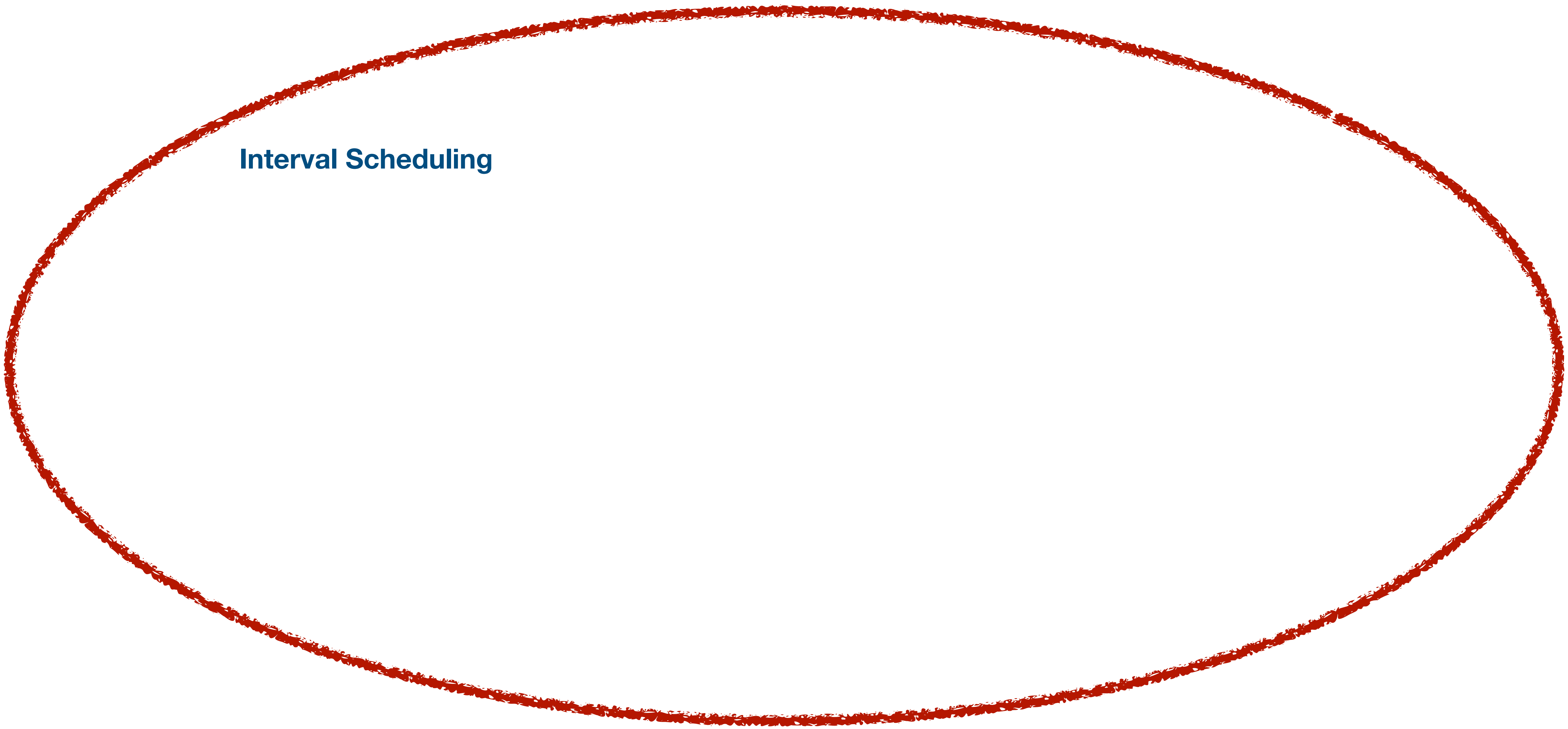
The class NP

- Stands for "*non deterministic polynomial time*".
- Problems that can be solved in polynomial time by a non-deterministic Turing machine.
- More intuitive definition:
    - Problems such that, *if a solution is given*, it can be *checked* that it is indeed a solution in polynomial time.
    - *Efficiently verifiable*.

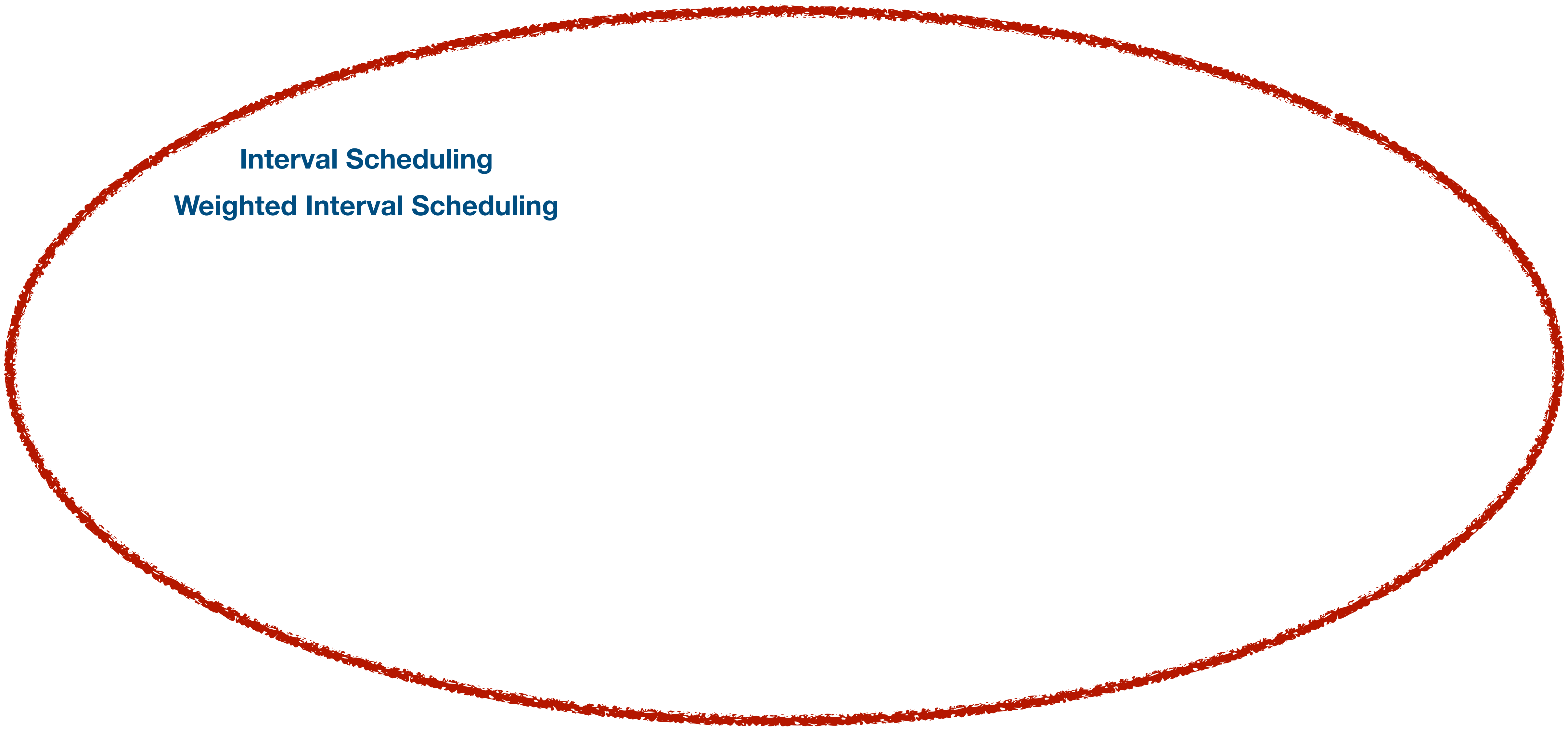
The class NP



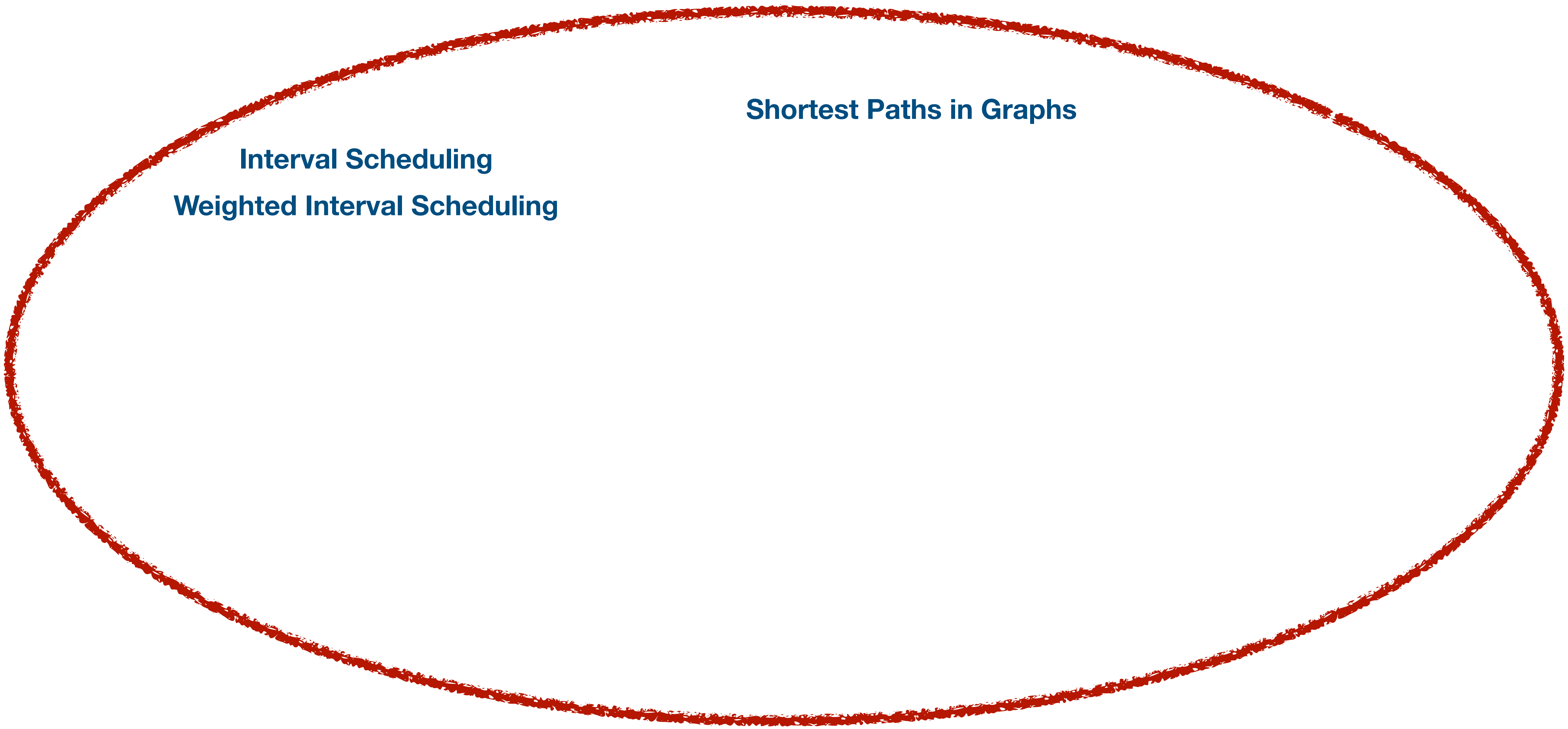
The class NP



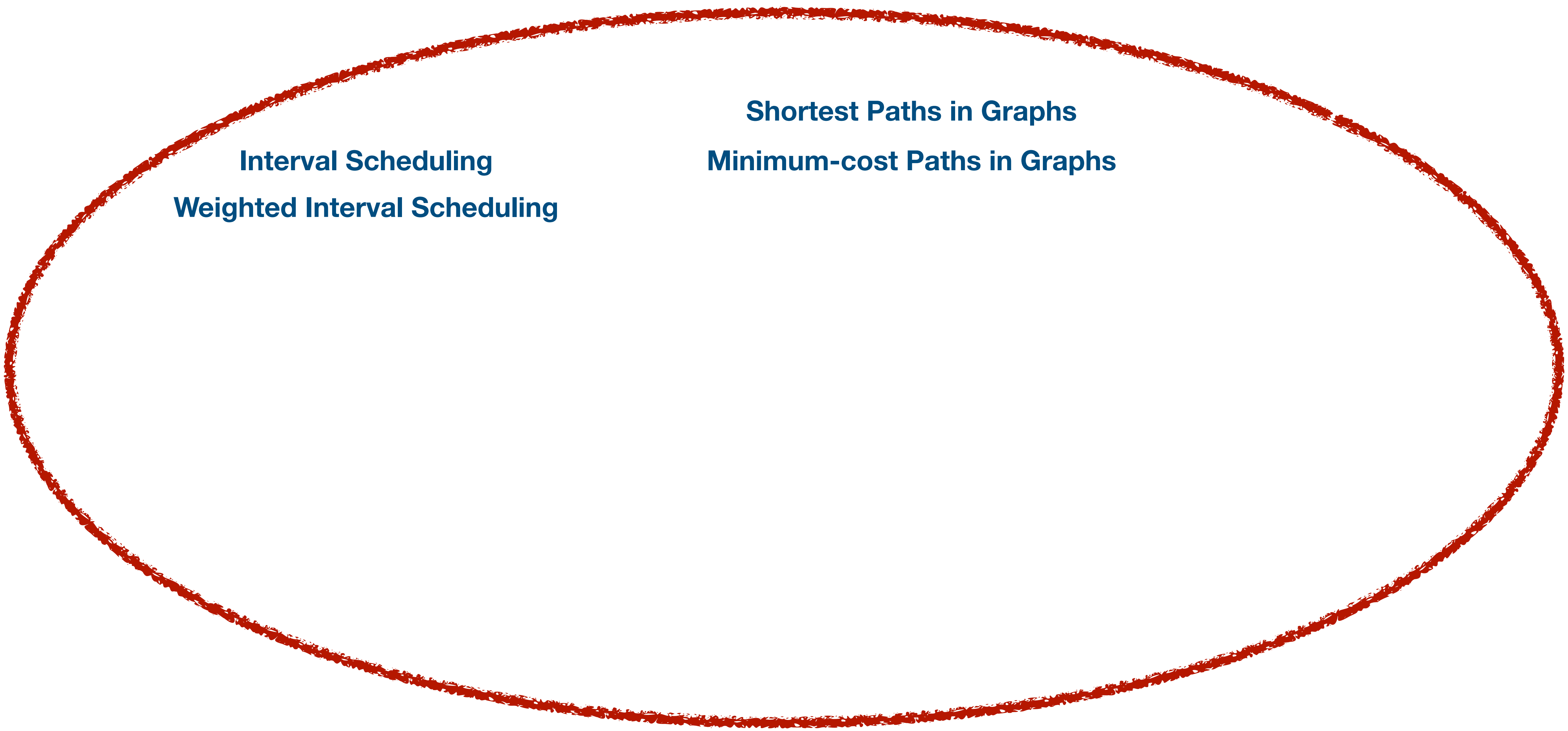
The class NP



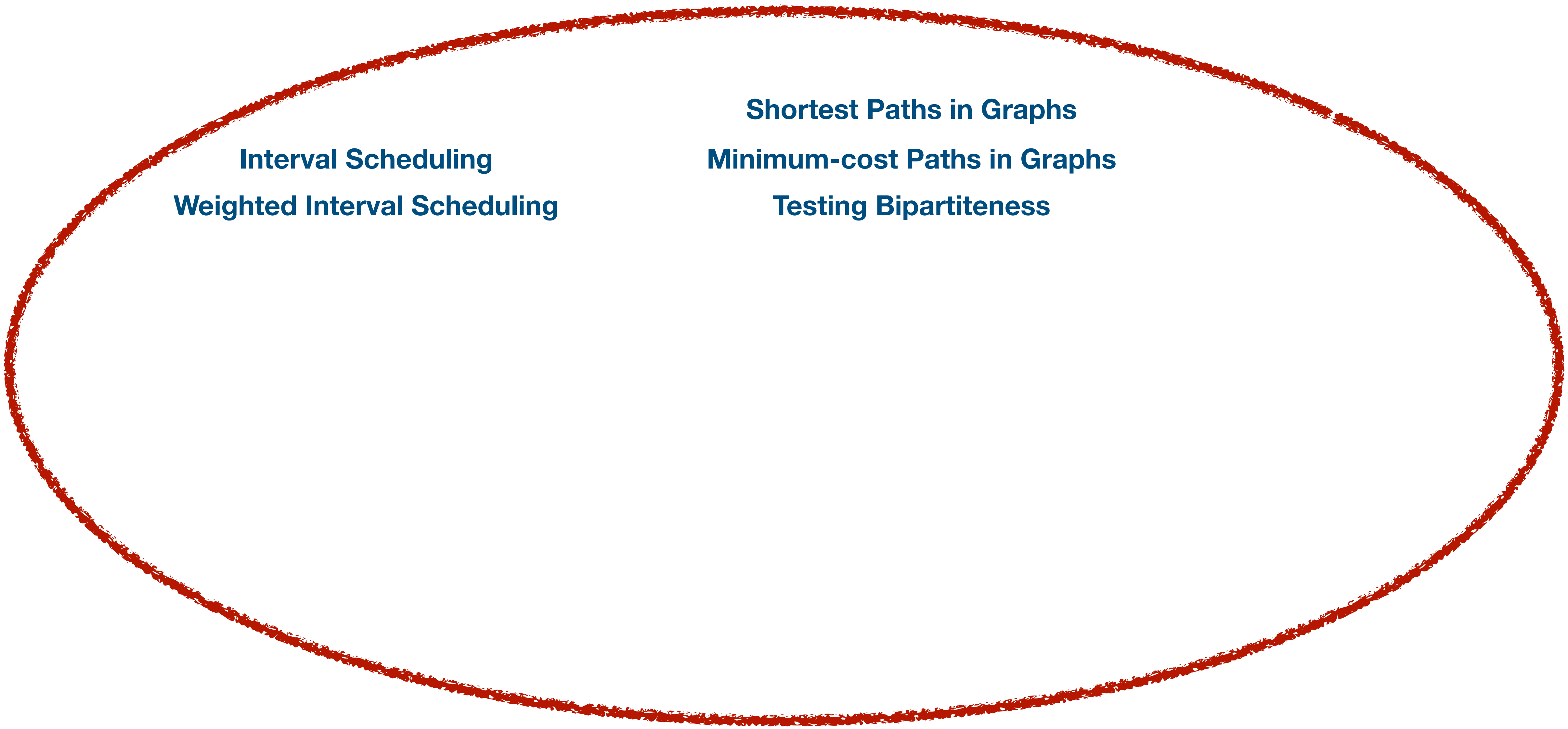
The class NP


Shortest Paths in Graphs
Interval Scheduling
Weighted Interval Scheduling

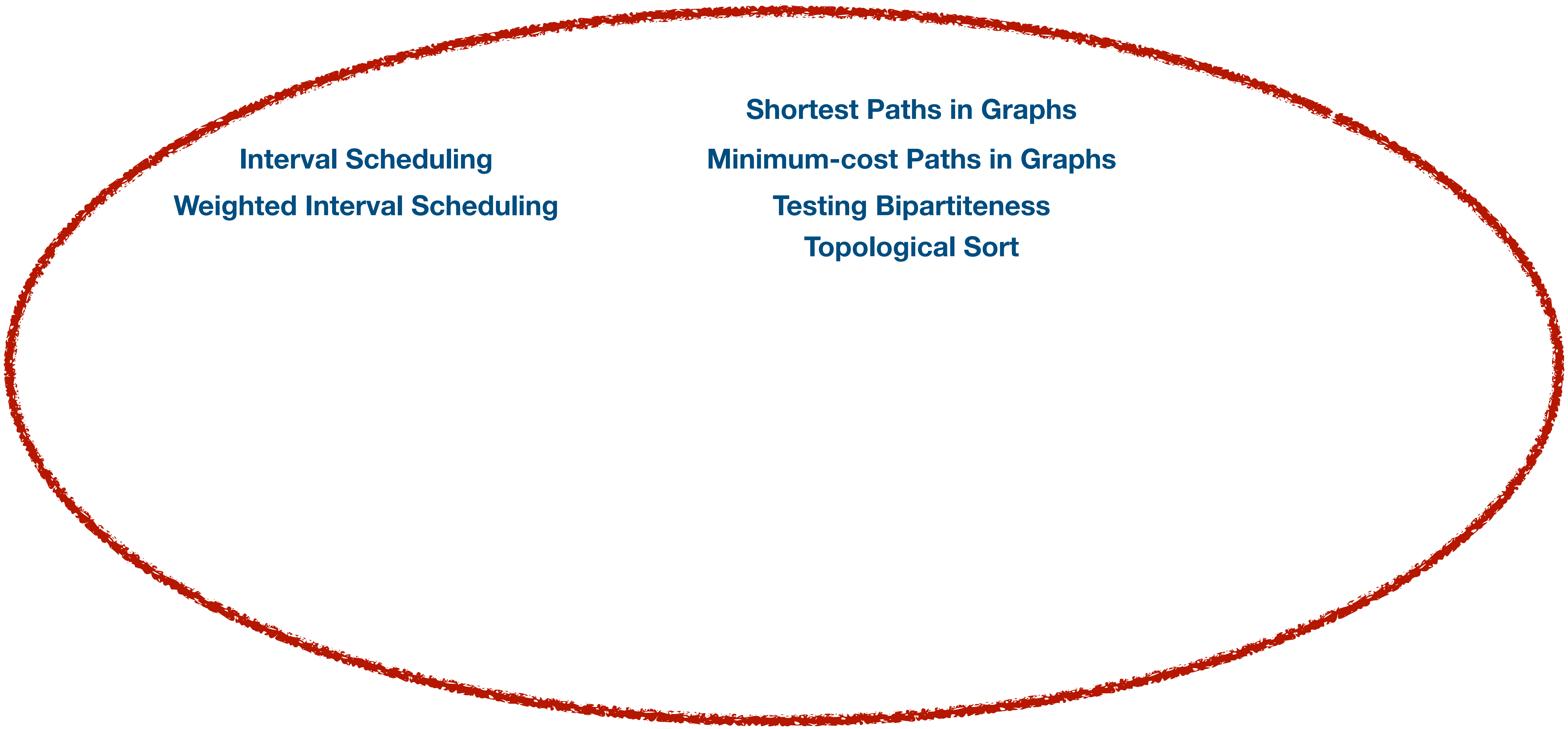
The class NP


Shortest Paths in Graphs
Interval Scheduling
Minimum-cost Paths in Graphs
Weighted Interval Scheduling

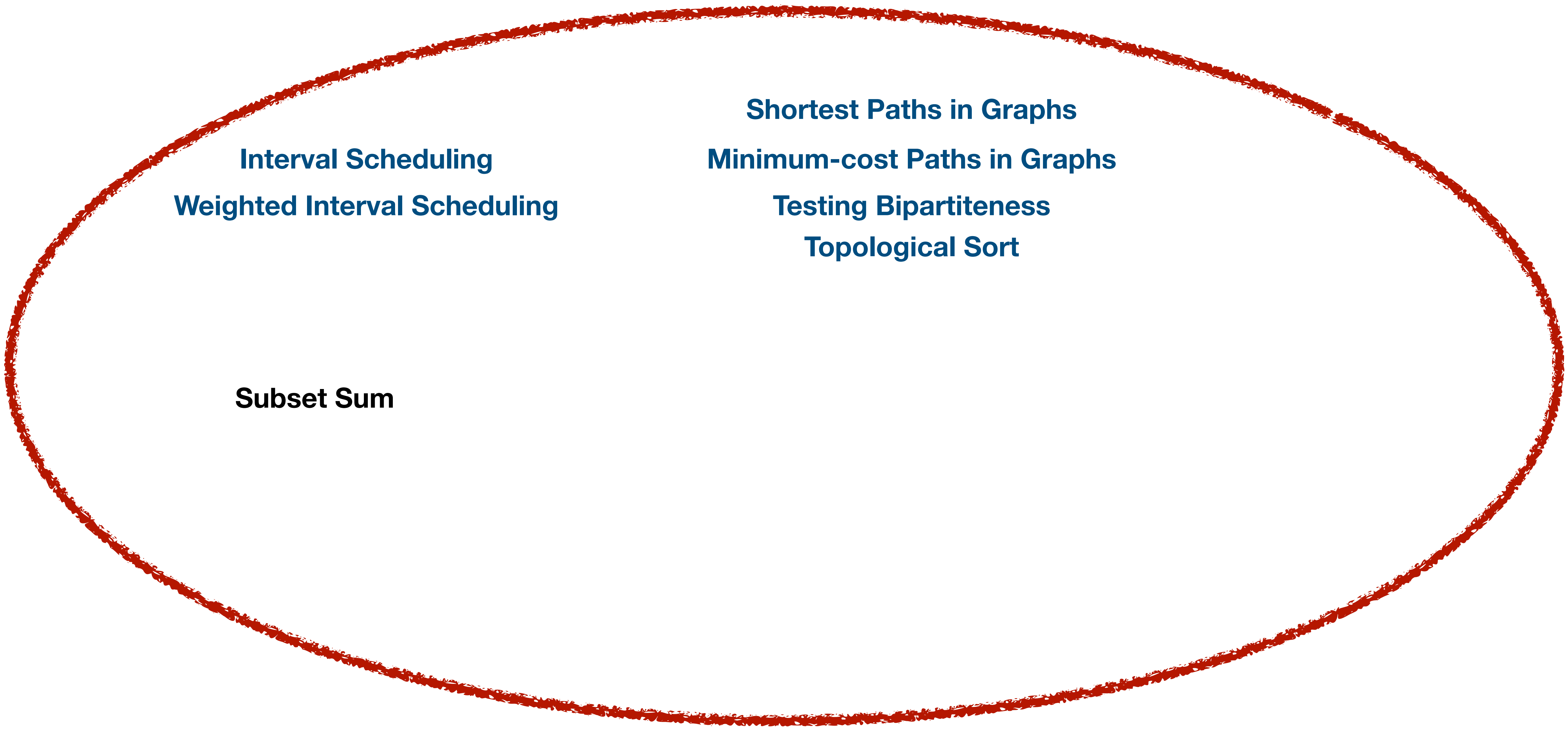
The class NP


Shortest Paths in Graphs
Interval Scheduling
Minimum-cost Paths in Graphs
Weighted Interval Scheduling
Testing Bipartiteness

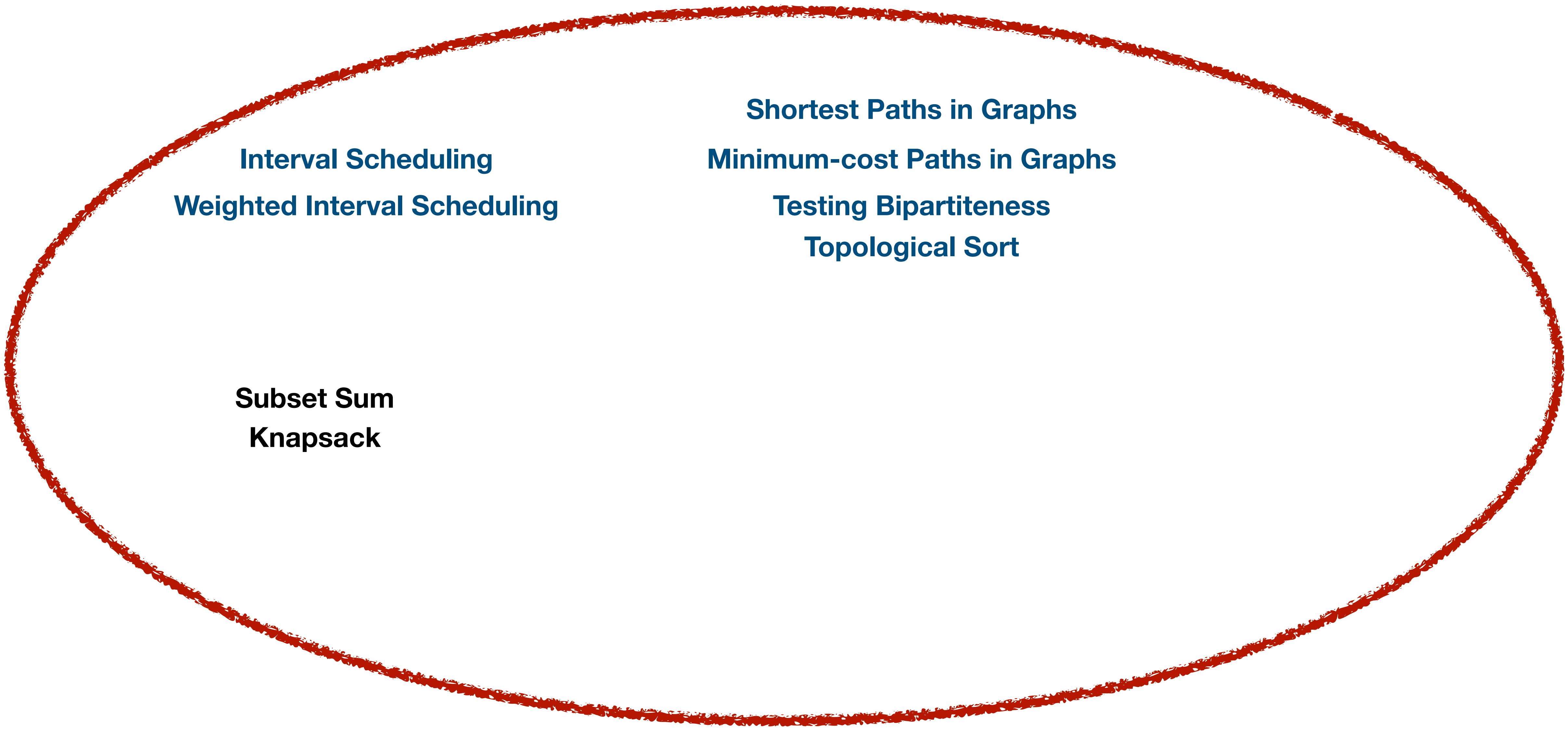
The class NP


Shortest Paths in Graphs
Interval Scheduling
Minimum-cost Paths in Graphs
Weighted Interval Scheduling
Testing Bipartiteness
Topological Sort

The class NP


Shortest Paths in Graphs
Interval Scheduling
Minimum-cost Paths in Graphs
Weighted Interval Scheduling
Testing Bipartiteness
Topological Sort
Subset Sum

The class NP


Shortest Paths in Graphs
Interval Scheduling
Minimum-cost Paths in Graphs
Weighted Interval Scheduling
Testing Bipartiteness
Topological Sort
Subset Sum
Knapsack

3 SAT

- A CNF formula with m clauses and k literals.

  $\phi = (x_1 \vee x_5 \vee x_3) \wedge (x_2 \vee x_6 \vee \neg x_5) \wedge \ldots \wedge (x_3 \vee x_8 \vee x_{12})$

- ("An AND of ORs").

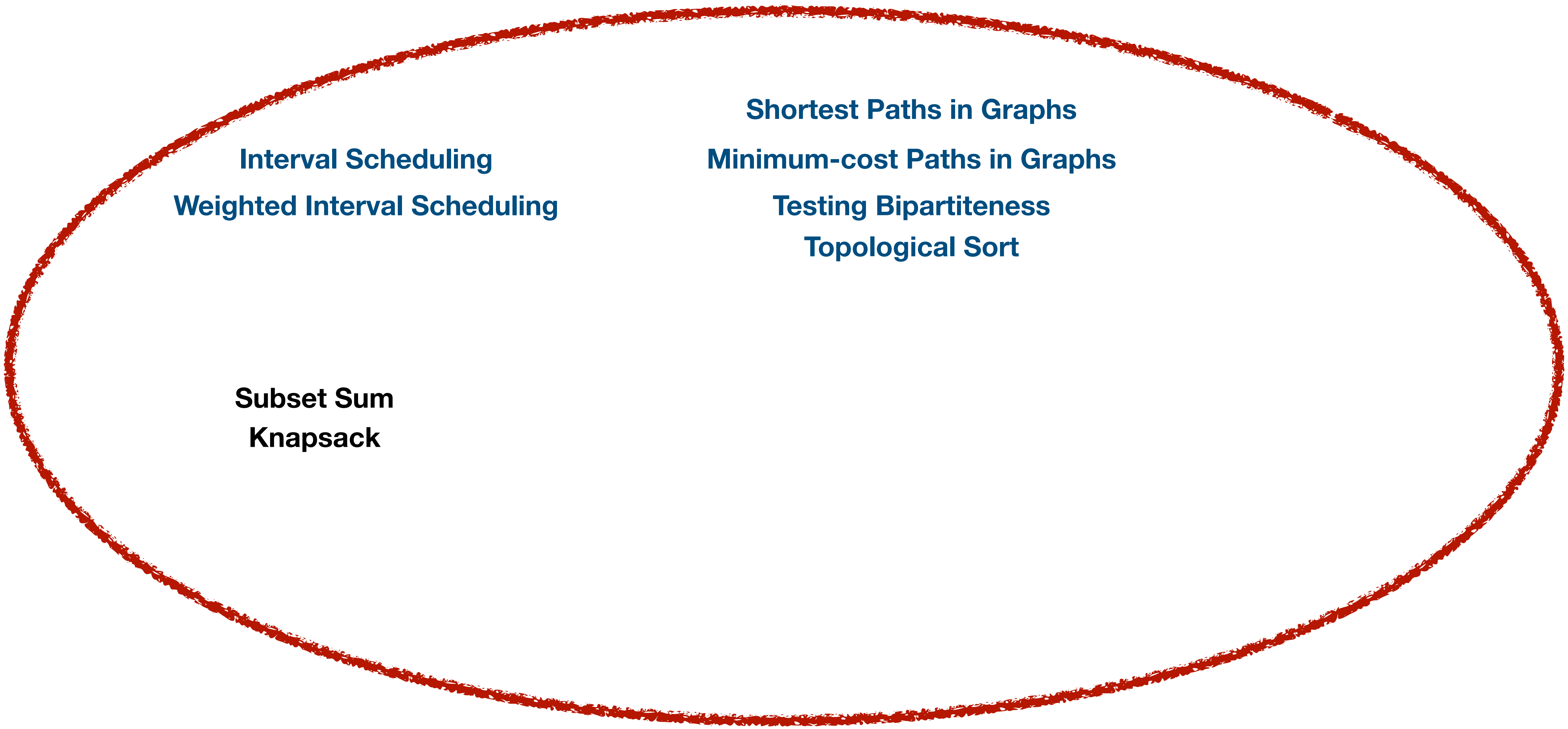
- Each clause has three literals.

- Truth assignment: A value in {0,1} for each variable x_i.

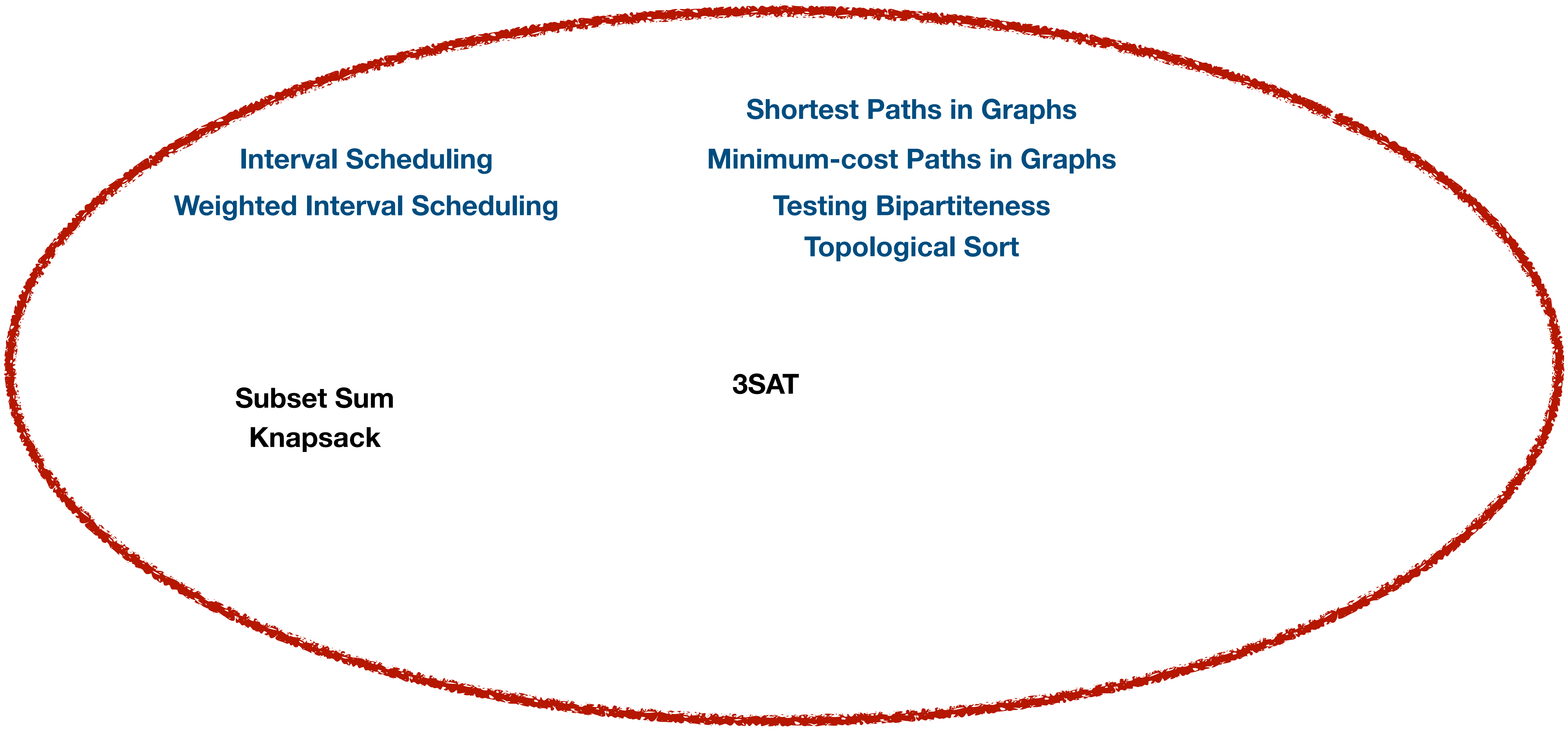
- Satisfying assignment: A truth assignment which makes the formula evaluate to 1 (= true).

- Computational problem 3SAT : Decide if the input formula ϕ has a satisfying assignment.

The class NP


Shortest Paths in Graphs
Interval Scheduling
Minimum-cost Paths in Graphs
Weighted Interval Scheduling
Testing Bipartiteness
Topological Sort
Subset Sum
Knapsack

The class NP


Shortest Paths in Graphs
Interval Scheduling
Minimum-cost Paths in Graphs
Weighted Interval Scheduling
Testing Bipartiteness
Topological Sort
3SAT
Subset Sum
Knapsack

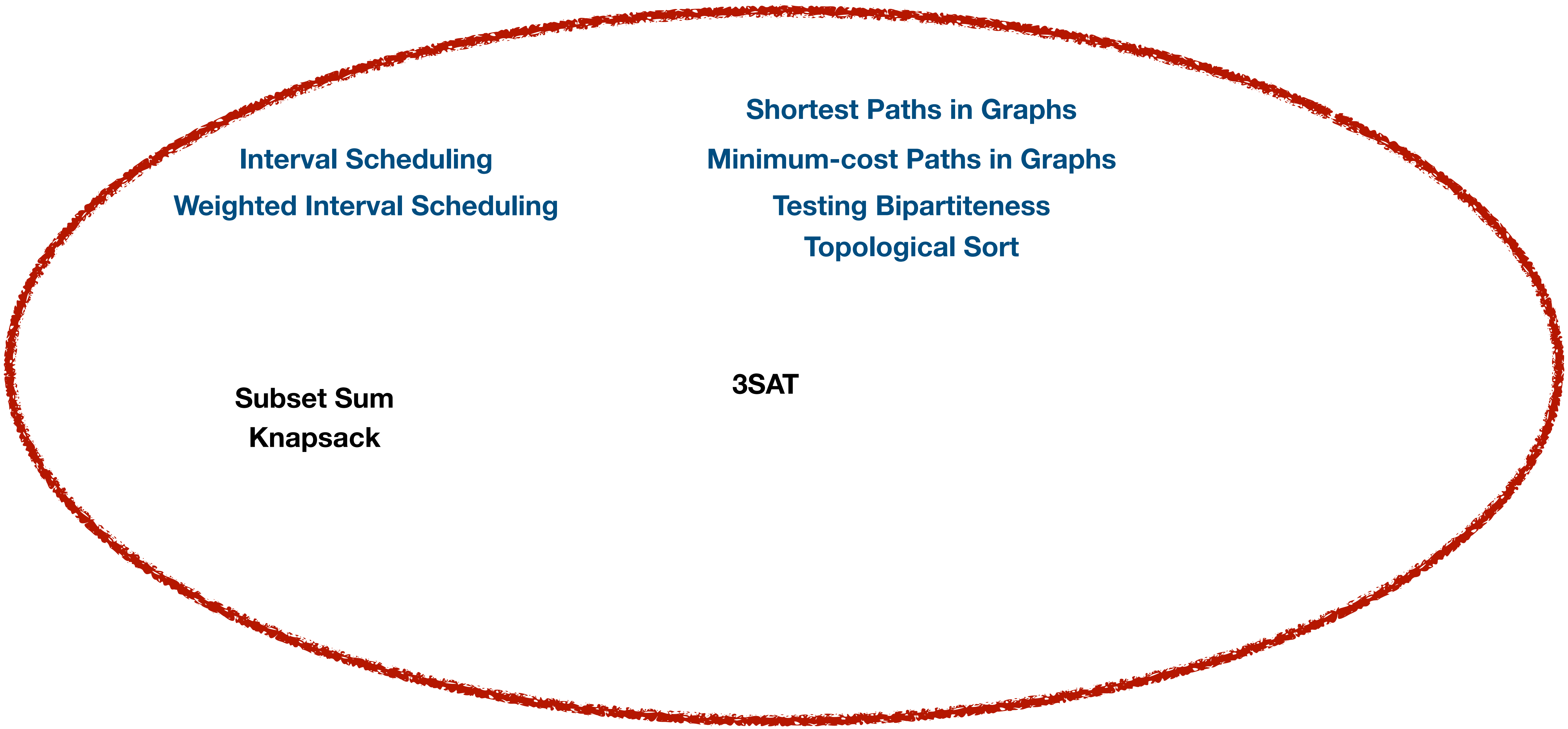
Vertex Cover decision version

- Definition: A vertex cover C of a graph $G=(V, E)$ is a subset of the nodes such that every edge e in the graph has at least one endpoint in C.

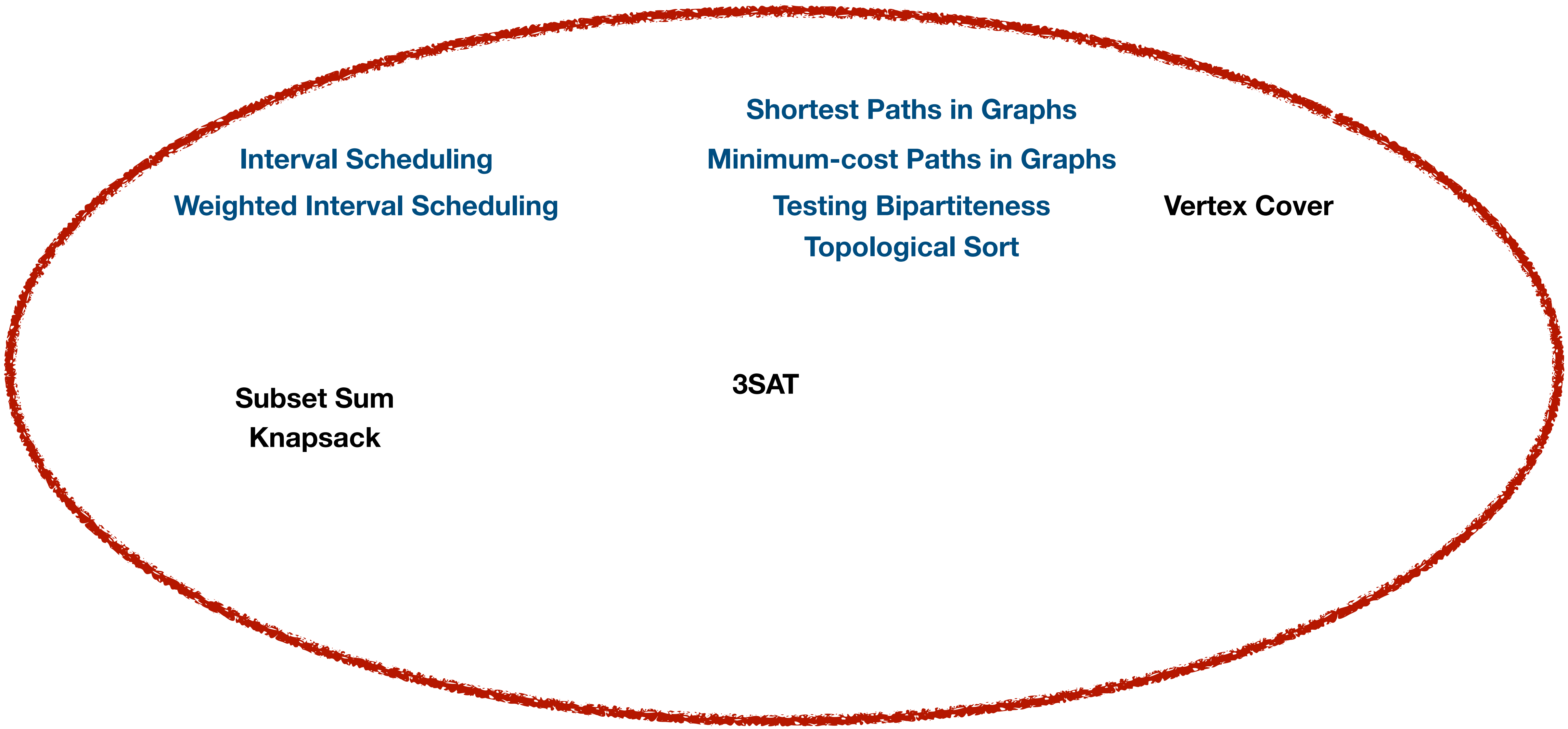
- Definition: A minimum vertex cover is a vertex cover of the smallest possible size.

- Vertex Cover
  Input: A graph $G=(V, E)$ and a number k
  Output: Is there a vertex cover of size $\leq k$?.

The class NP


Shortest Paths in Graphs
Interval Scheduling
Minimum-cost Paths in Graphs
Weighted Interval Scheduling
Testing Bipartiteness
Topological Sort
3SAT
Subset Sum
Knapsack

The class NP


Shortest Paths in Graphs
Interval Scheduling
Minimum-cost Paths in Graphs
Weighted Interval Scheduling
Testing Bipartiteness
Vertex Cover
Topological Sort
3SAT
Subset Sum
Knapsack

Other problems in NP

Other problems in NP

- Independent Set in graph G: A set of nodes in the graph, such that there is no edge between any two nodes in the set.

Other problems in NP

- Independent Set in graph G: A set of nodes in the graph, such that there is no edge between any two nodes in the set.

- Maximum Independent Set
  Given a graph G, find an independent set of maximum size.

Other problems in NP

- Independent Set in graph G: A set of nodes in the graph, such that there is no edge between any two nodes in the set.

- Maximum Independent Set
  Given a graph G, find an independent set of maximum size.

- Maximum Independent Set (decision version)
  Given a graph G, and an integer k, is there an independent set of size at least k?

Other problems in NP

Other problems in NP

- Set Packing
  Given a set U of elements, a collection $S_1, \ldots, S_m$ of subsets of U and a number k, does there exist a collection of at least k of these sets such that no two of them intersect?

Other problems in NP

- Set Packing
  Given a set U of elements, a collection $S_1, \ldots, S_m$ of subsets of U and a number k, does there exist a collection of at least k of these sets such that no two of them intersect?

- Set Cover
  Given a set U of elements, a collection $S_1, \ldots, S_m$ of subsets of U and a number k, does there exist a collection of at most k of these sets whose union is equal to U?

Other problems in NP

Other problems in NP

- 3-Dimensional Matching
  Given disjoint sets X, Y and Z each of size n, and given a set T (which is a subset of X x Y x Z) of ordered triples, does there exist a set of n triples in T, so that each element of X ∪ Y ∪ Z is contained in exactly in one of these triples?

Other problems in NP

Other problems in NP

- k-Colouring of a graph G: A function f: V → {*1*, …, *k*} so that for every edge (u, v) we have that f(u) ≠ f(v).

Other problems in NP

- k-Colouring of a graph G: A function f: $V \rightarrow \{1, \ldots, k\}$ so that for every edge (u, v) we have that $f(u) \neq f(v)$.

- 3-Colouring
  Given a graph G, does it have a 3-Colouring?

Other problems in NP

Other problems in NP

- Hamiltonian cycle in a directed graph G: A cycle in a directed graph that visits each vertex *exactly once*.

Other problems in NP

- Hamiltonian cycle in a directed graph G: A cycle in a directed graph that visits each vertex *exactly once*.

- Hamiltonian path in a directed graph G: A path in a directed graph that contains each vertex *exactly once*.

Other problems in NP

- Hamiltonian cycle in a directed graph G: A cycle in a directed graph that visits each vertex *exactly once*.

- Hamiltonian path in a directed graph G: A path in a directed graph that contains each vertex *exactly once*.

- Hamiltonian Cycle
  Given a directed graph G, does it have a Hamiltonian Cycle?

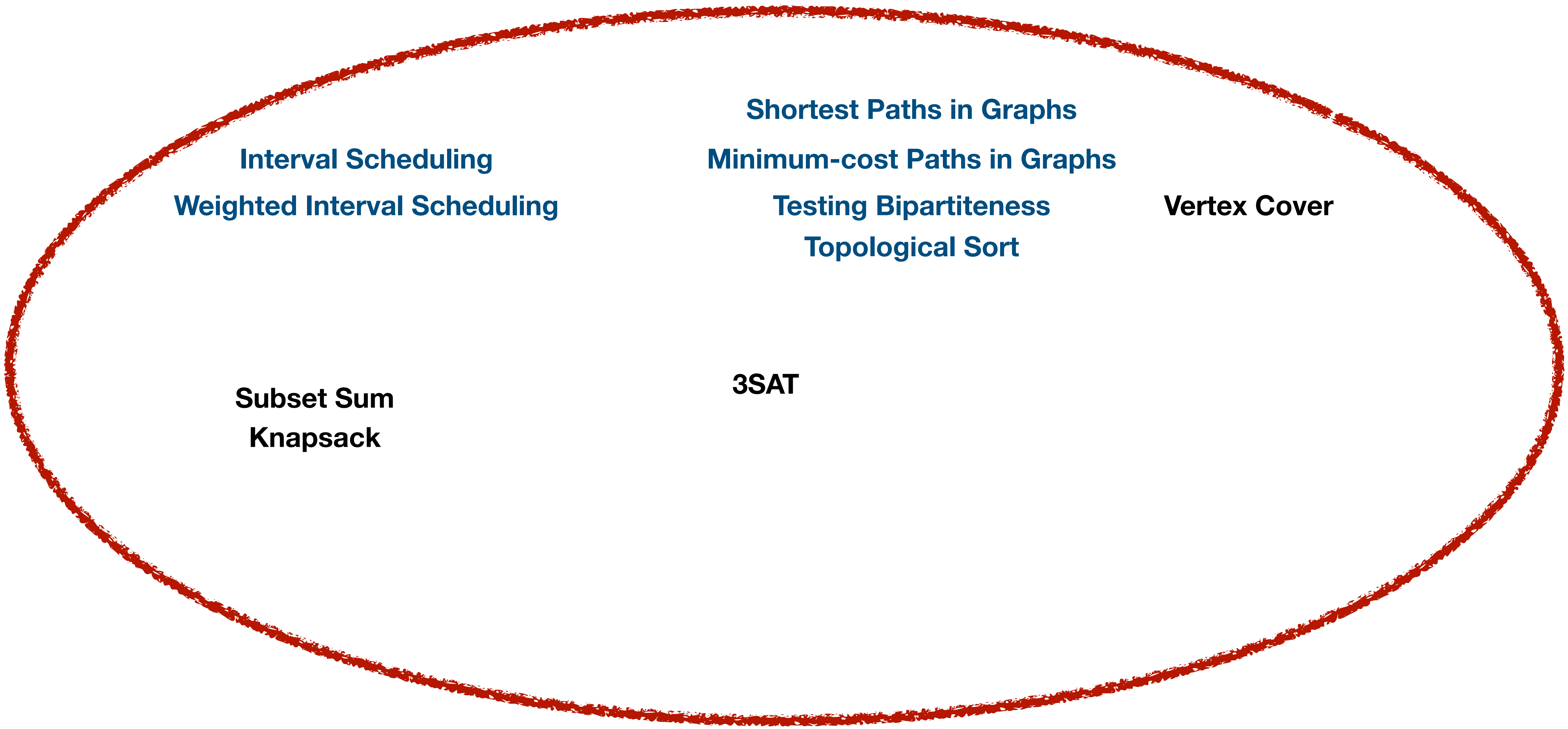
Other problems in NP

- Hamiltonian cycle in a directed graph G: A cycle in a directed graph that visits each vertex *exactly once*.
- Hamiltonian path in a directed graph G: A path in a directed graph that contains each vertex *exactly once*.
- Hamiltonian Cycle
  Given a directed graph G, does it have a Hamiltonian Cycle?
- Hamiltonian Path
  Given a directed graph G, does it have a Hamiltonian Path?

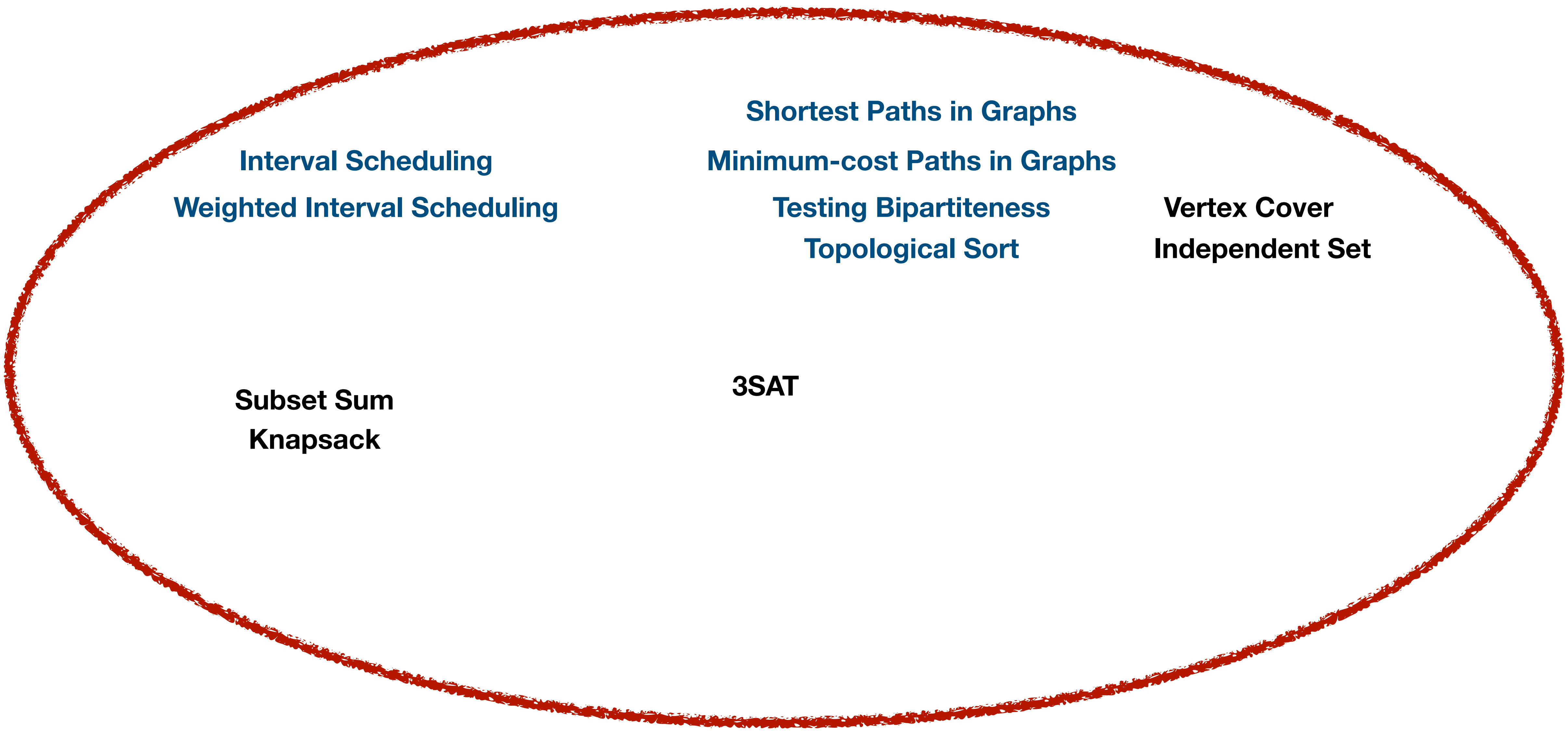
Other problems in NP

- Hamiltonian cycle in a directed graph G: A cycle in a directed graph that visits each vertex *exactly once*.
- Hamiltonian path in a directed graph G: A path in a directed graph that contains each vertex *exactly once*.
- Hamiltonian Cycle
  Given a directed graph G, does it have a Hamiltonian Cycle?
- Hamiltonian Path
  Given a directed graph G, does it have a Hamiltonian Path?
- Traveling Salesman
  (def Kleinberg and Tardos, p. 474).

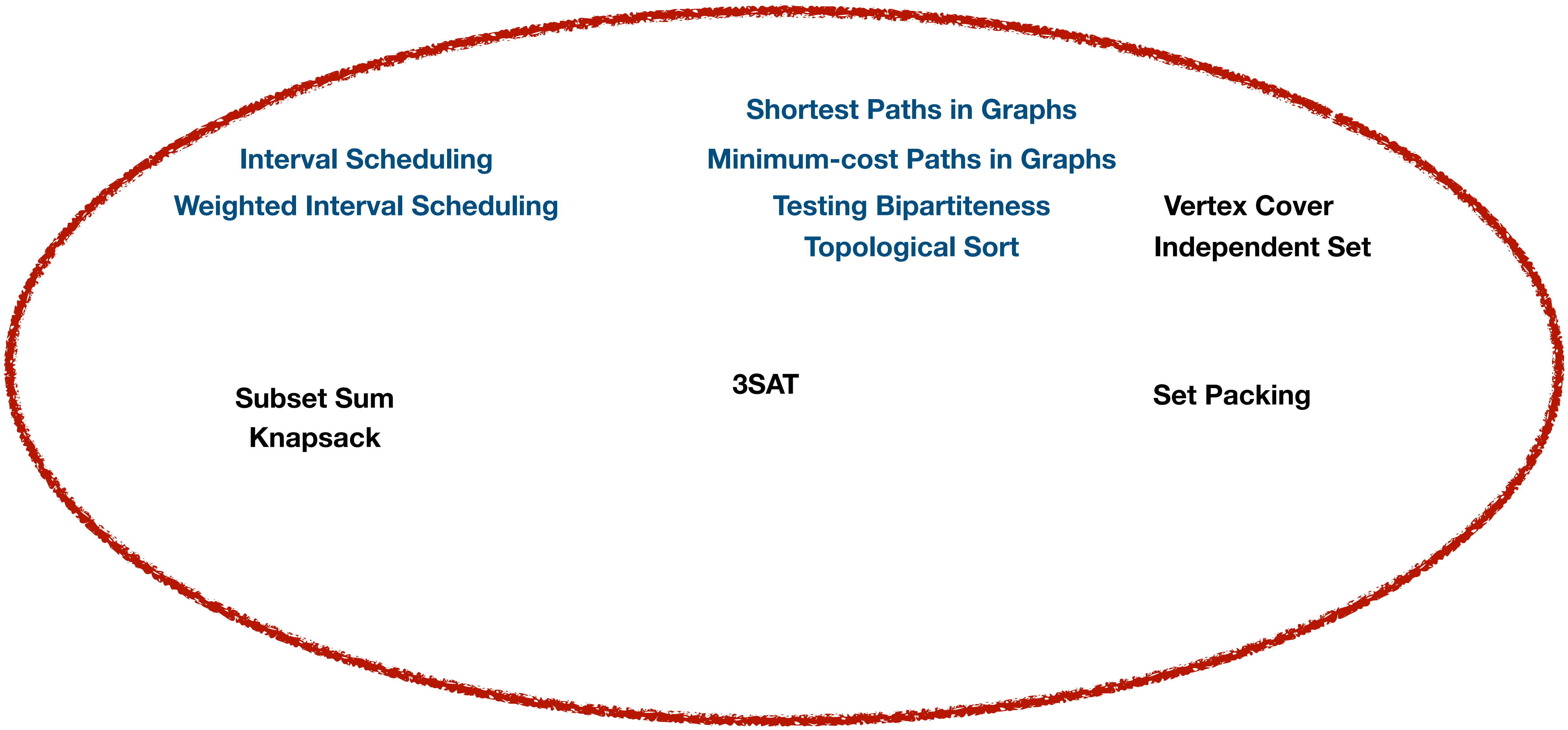
The class NP


Shortest Paths in Graphs
Interval Scheduling
Minimum-cost Paths in Graphs
Weighted Interval Scheduling
Testing Bipartiteness
Vertex Cover
Topological Sort
3SAT
Subset Sum
Knapsack

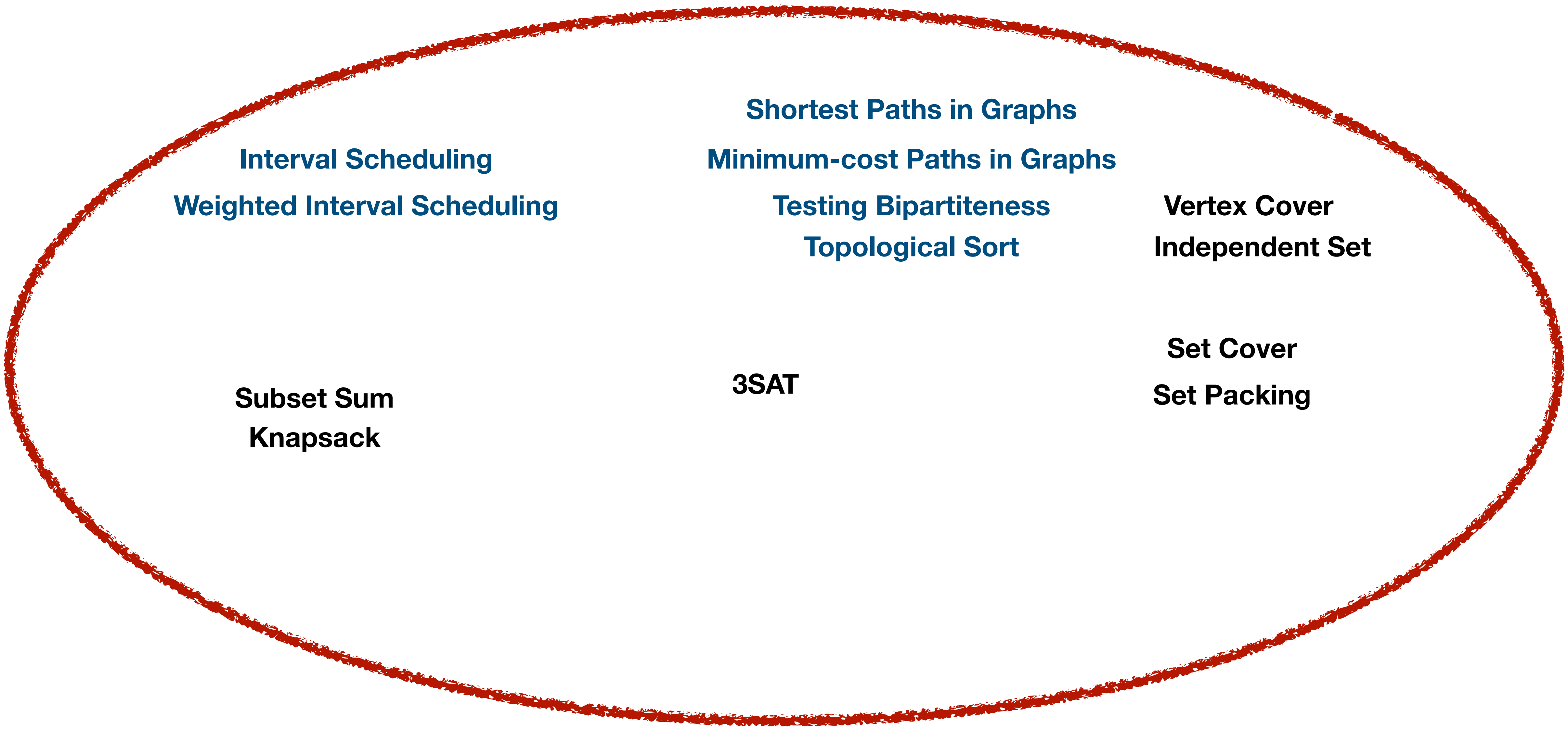
The class NP


Shortest Paths in Graphs
Interval Scheduling
Minimum-cost Paths in Graphs
Weighted Interval Scheduling
Testing Bipartiteness
Vertex Cover
Topological Sort
Independent Set
3SAT
Subset Sum
Knapsack

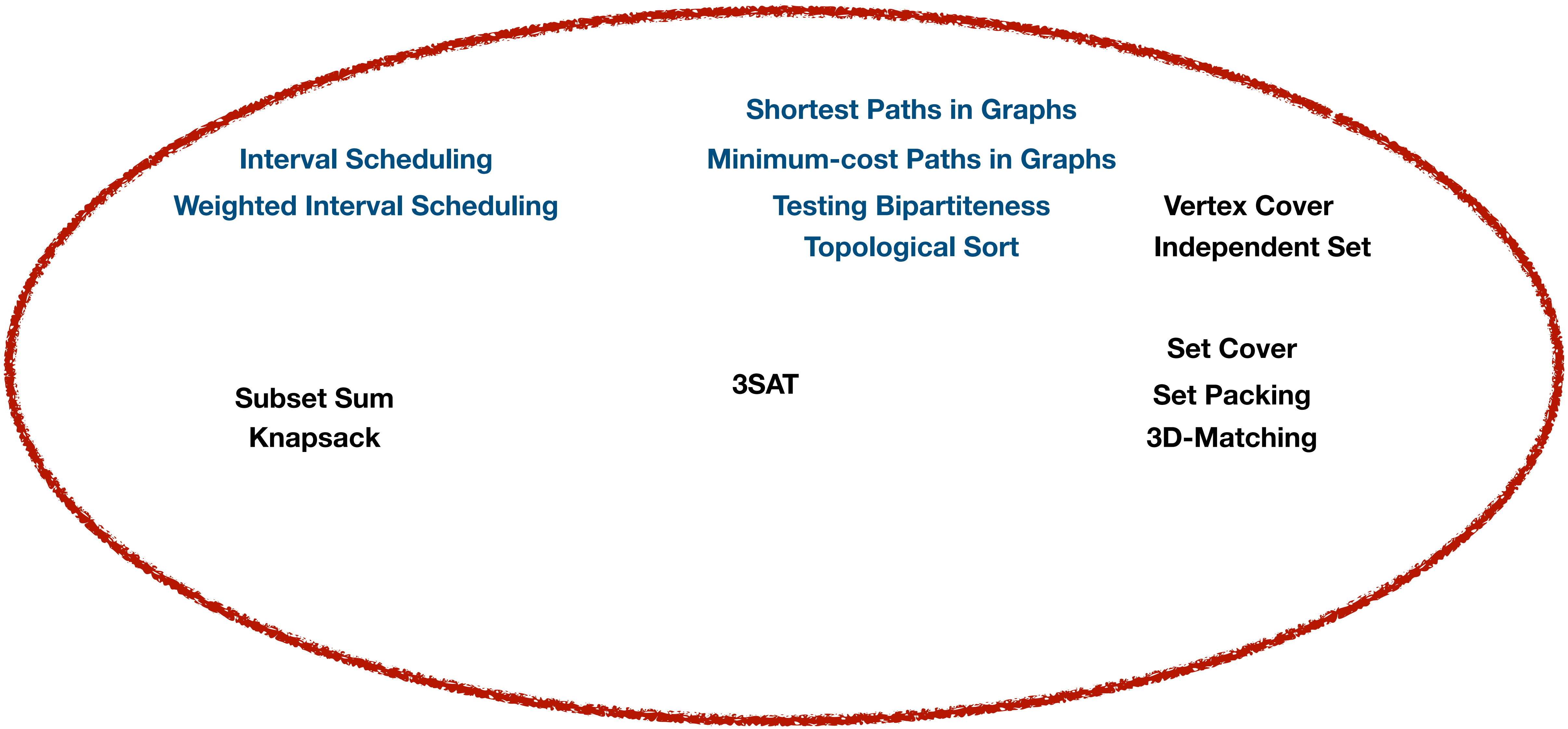
The class NP


Shortest Paths in Graphs
Interval Scheduling
Minimum-cost Paths in Graphs
Weighted Interval Scheduling
Testing Bipartiteness
Vertex Cover
Topological Sort
Independent Set
Subset Sum
Knapsack
3SAT
Set Packing

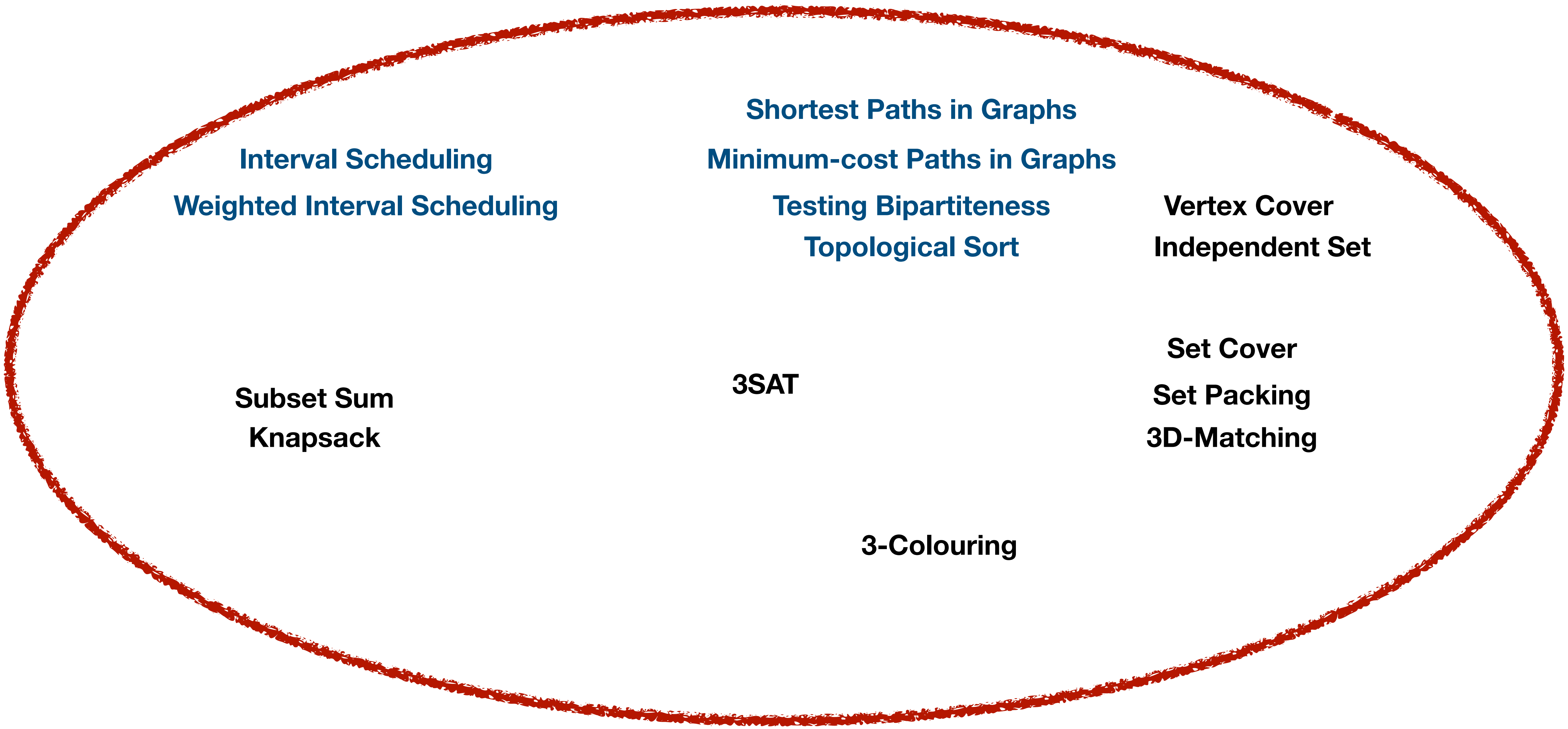
The class NP


Shortest Paths in Graphs
Interval Scheduling
Minimum-cost Paths in Graphs
Weighted Interval Scheduling
Testing Bipartiteness
Vertex Cover
Topological Sort
Independent Set
Set Cover
3SAT
Subset Sum
Set Packing
Knapsack

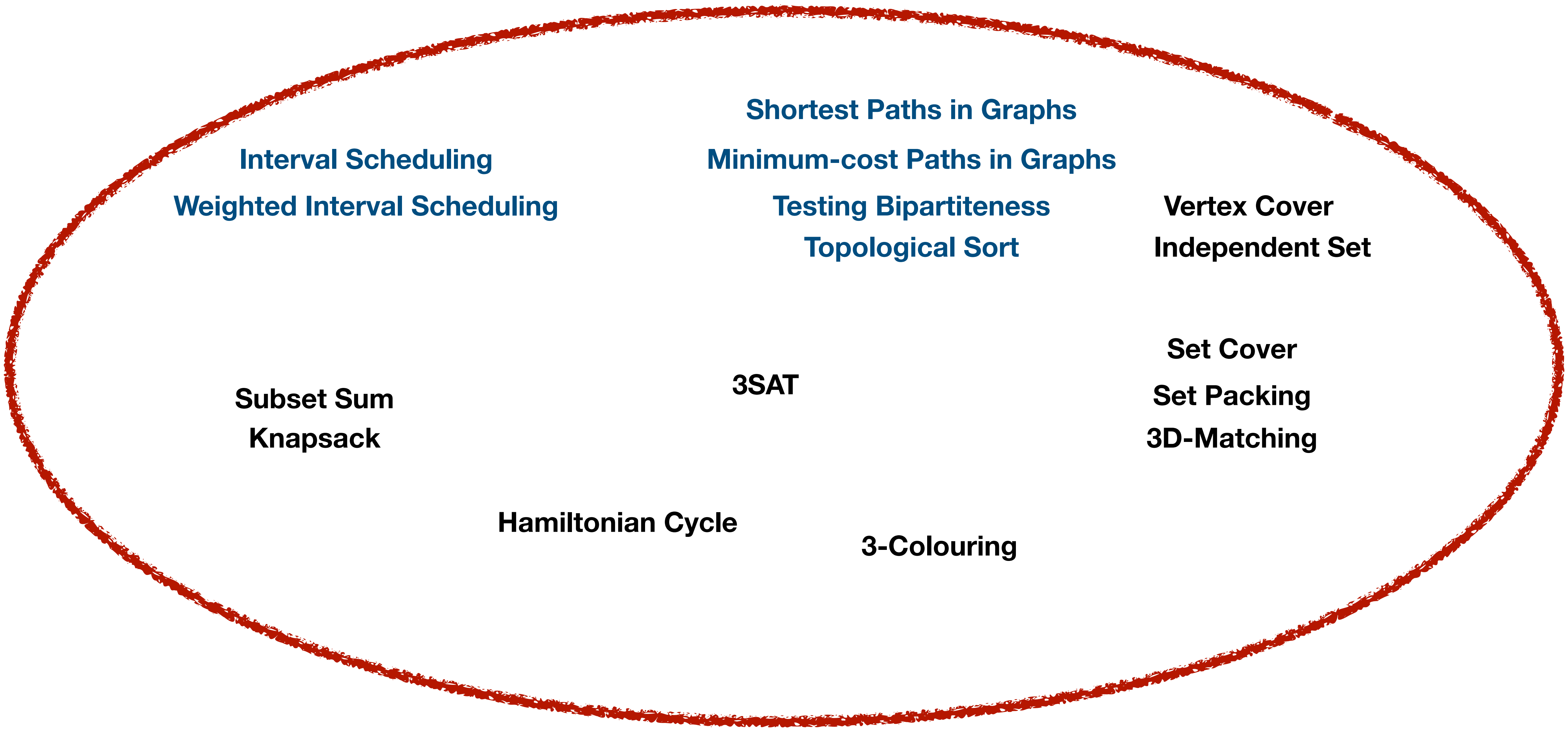
The class **NP**


Shortest Paths in Graphs
Interval Scheduling
Minimum-cost Paths in Graphs
Weighted Interval Scheduling
Testing Bipartiteness
Vertex Cover
Topological Sort
Independent Set
Set Cover
3SAT
Subset Sum
Set Packing
Knapsack
3D-Matching

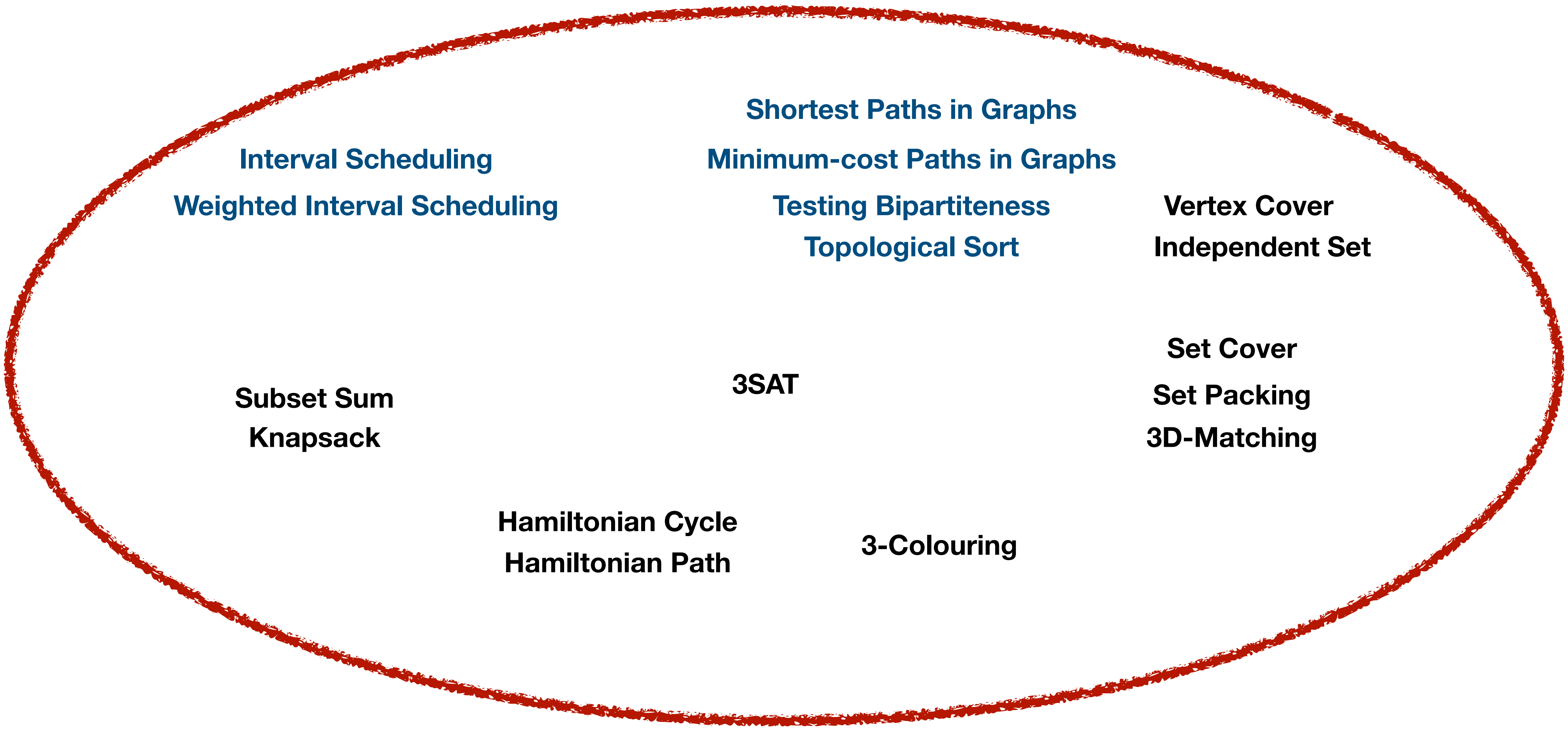
The class NP


Shortest Paths in Graphs
Interval Scheduling
Minimum-cost Paths in Graphs
Weighted Interval Scheduling
Testing Bipartiteness
Vertex Cover
Topological Sort
Independent Set
Set Cover
3SAT
Subset Sum
Set Packing
Knapsack
3D-Matching
3-Colouring

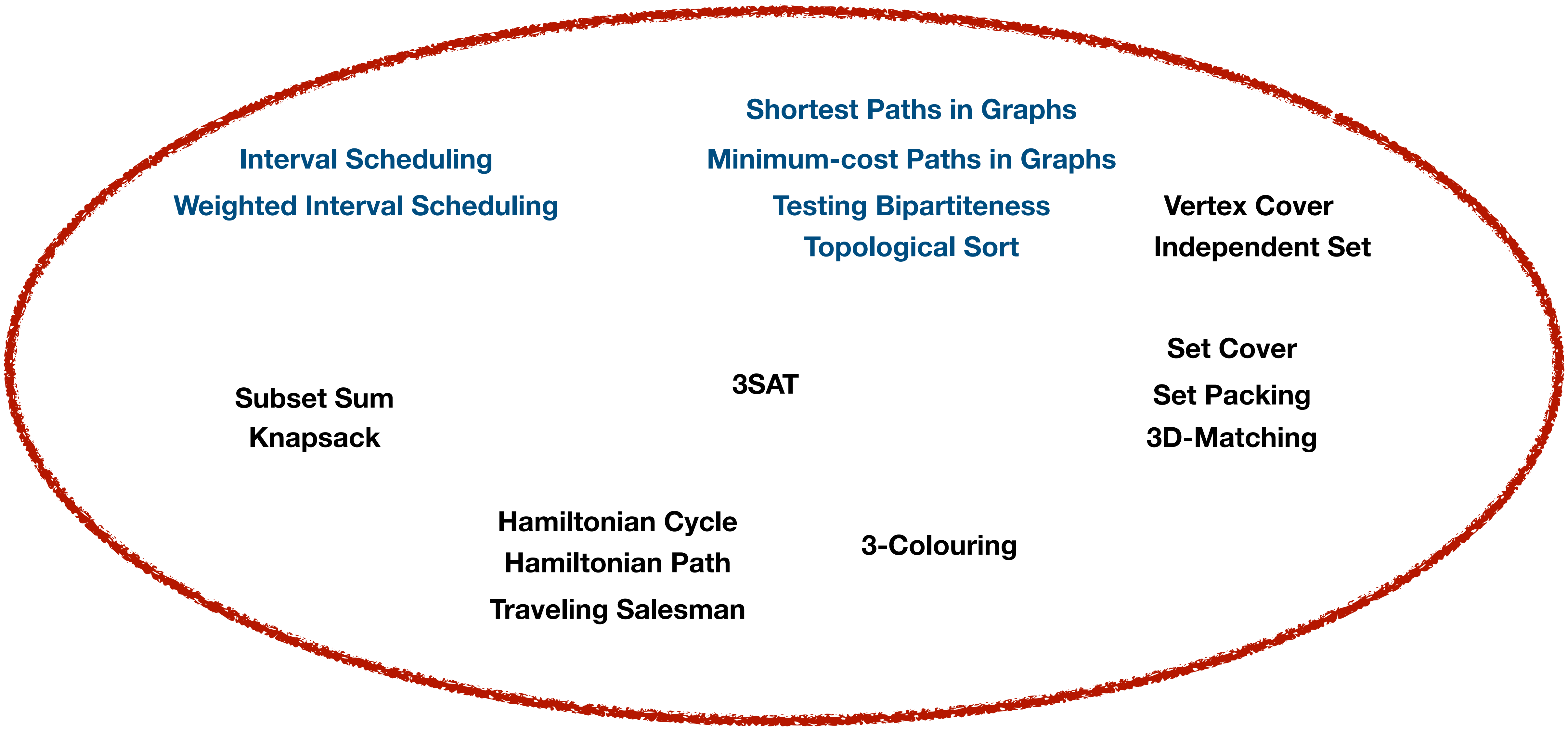
The class NP


Shortest Paths in Graphs
Interval Scheduling
Minimum-cost Paths in Graphs
Weighted Interval Scheduling
Testing Bipartiteness
Vertex Cover
Topological Sort
Independent Set
Set Cover
3SAT
Subset Sum
Set Packing
Knapsack
3D-Matching
Hamiltonian Cycle
3-Colouring

The class NP


Shortest Paths in Graphs
Minimum-cost Paths in Graphs
Testing Bipartiteness
Topological Sort
Interval Scheduling
Weighted Interval Scheduling
Vertex Cover
Independent Set
Set Cover
Set Packing
3D-Matching
3SAT
Subset Sum
Knapsack
Hamiltonian Cycle
Hamiltonian Path
3-Colouring

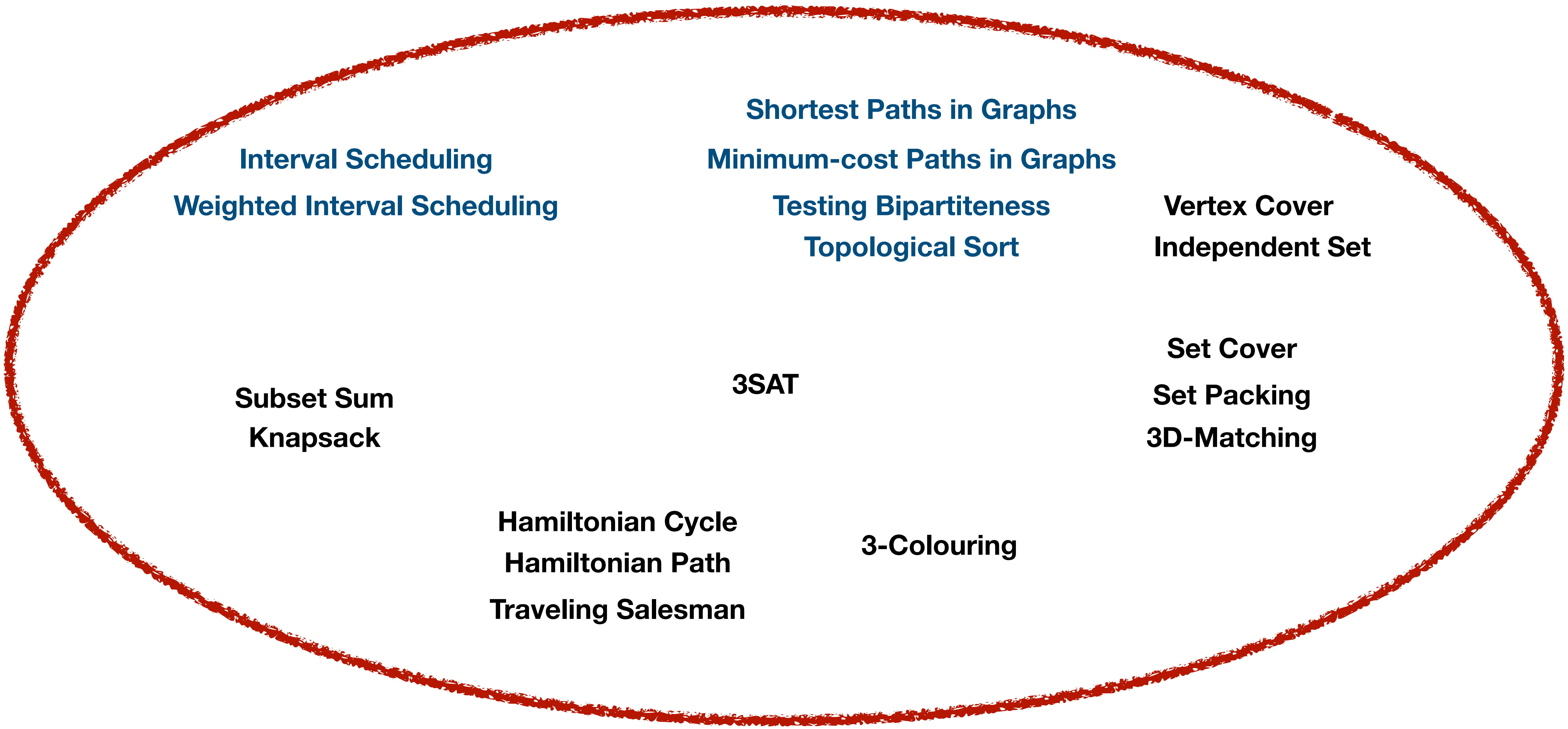
The class NP


Shortest Paths in Graphs
Interval Scheduling
Minimum-cost Paths in Graphs
Weighted Interval Scheduling
Testing Bipartiteness
Vertex Cover
Topological Sort
Independent Set
Set Cover
3SAT
Subset Sum
Set Packing
Knapsack
3D-Matching
Hamiltonian Cycle
3-Colouring
Hamiltonian Path
Traveling Salesman

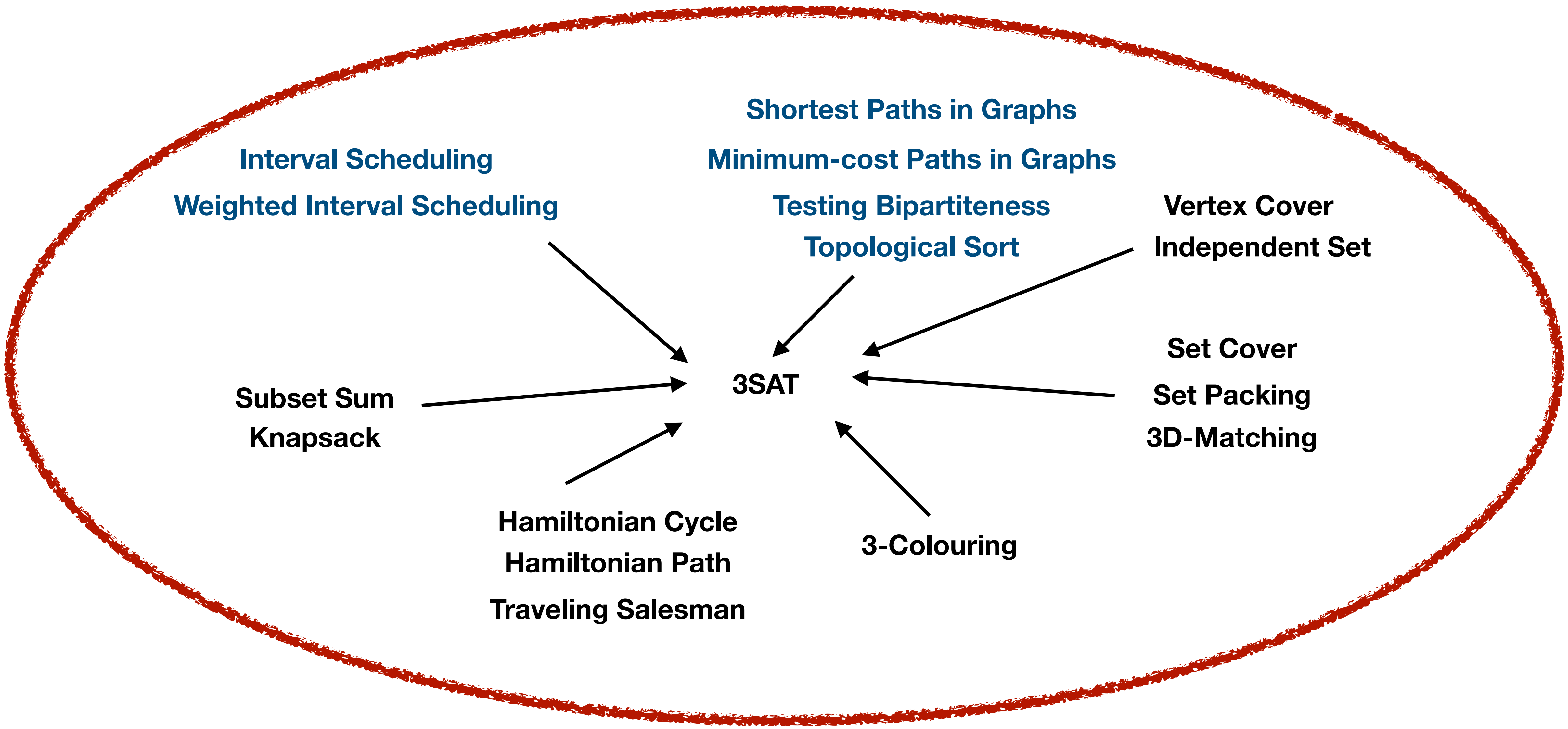
NP-completeness

- A problem B is NP-complete if
  - *It is in NP*.
    - i.e., it has a polynomial-time verifiable solution.
  - *It is NP-hard*.
    - i.e., every problem in NP can be efficiently reduced to it.

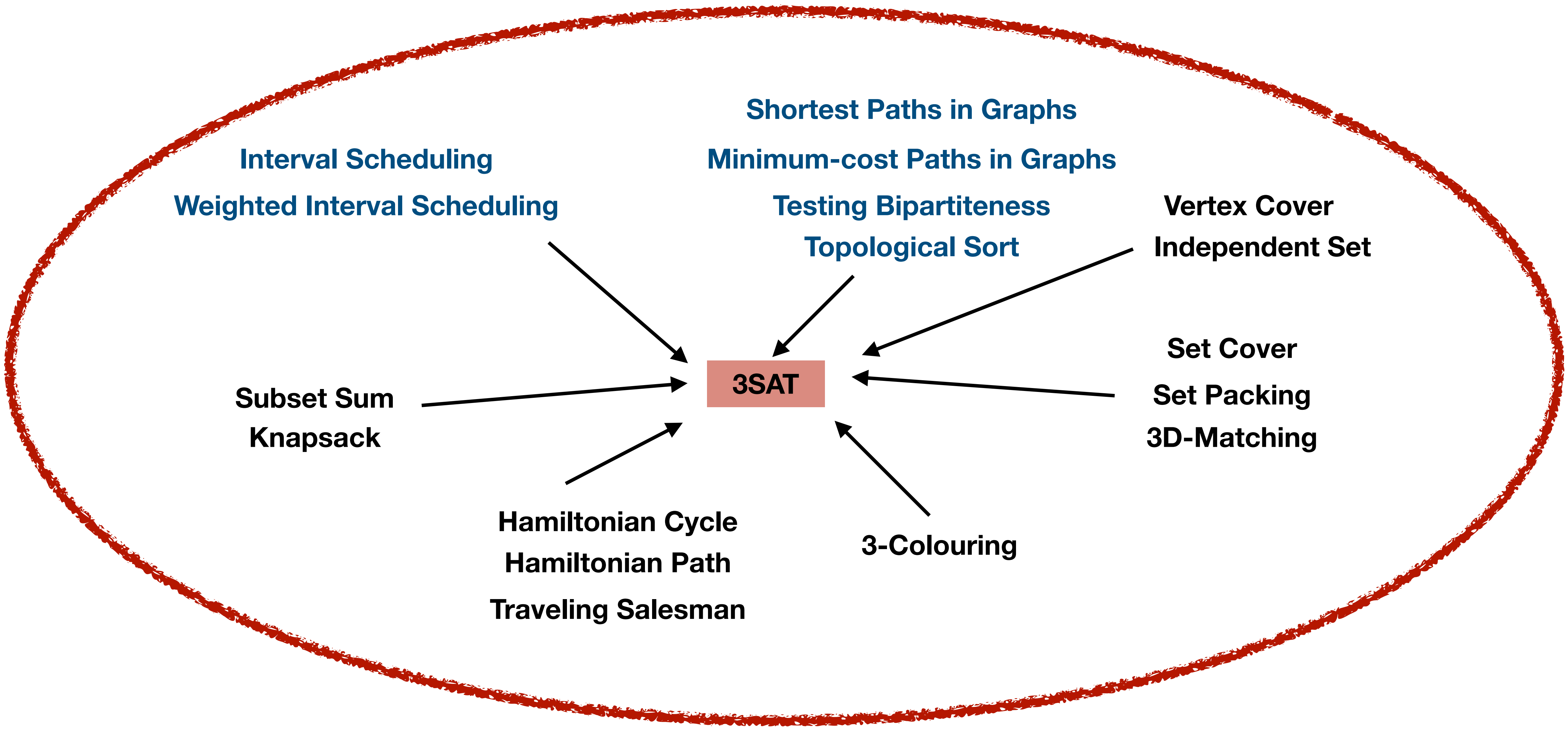
NP-completeness



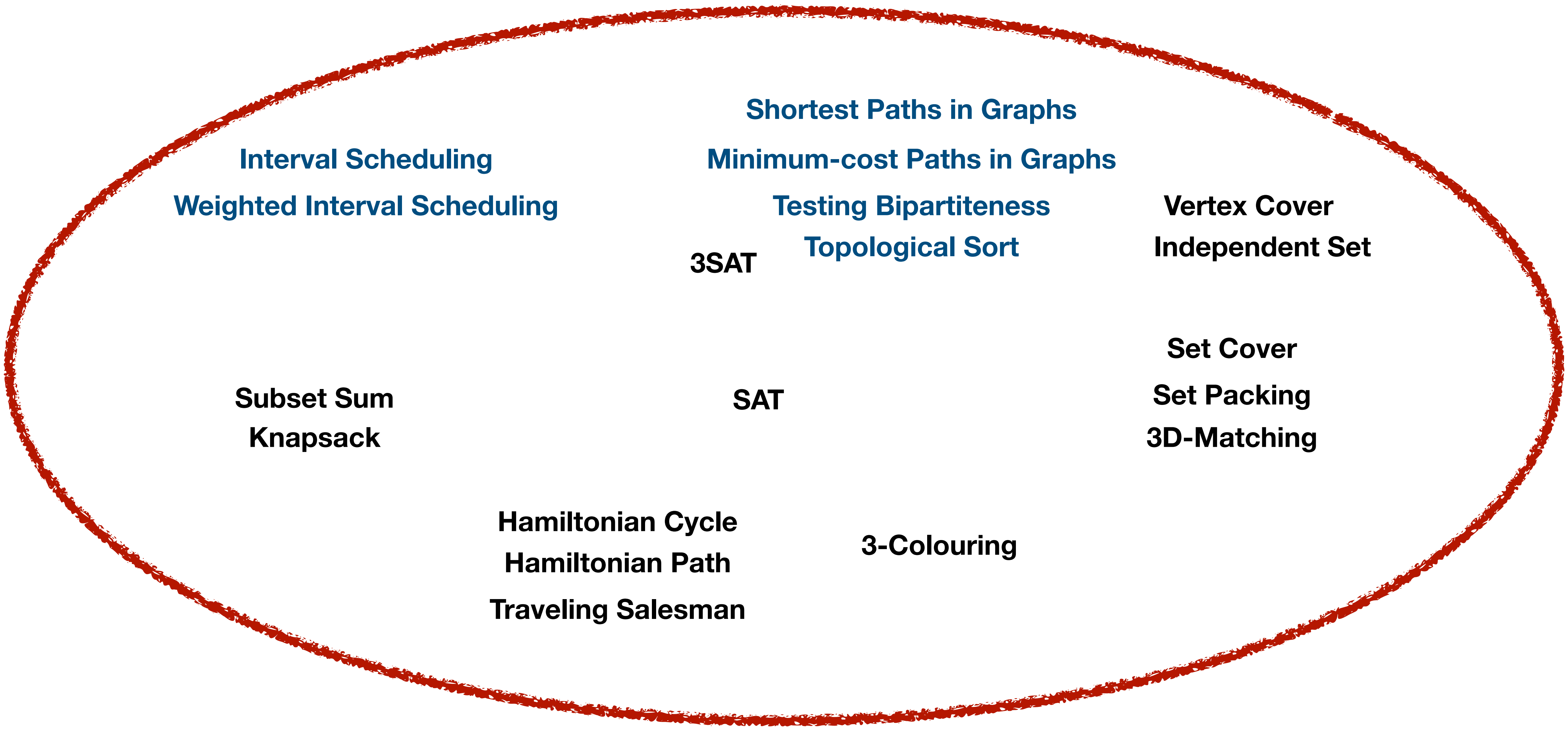
NP-completeness



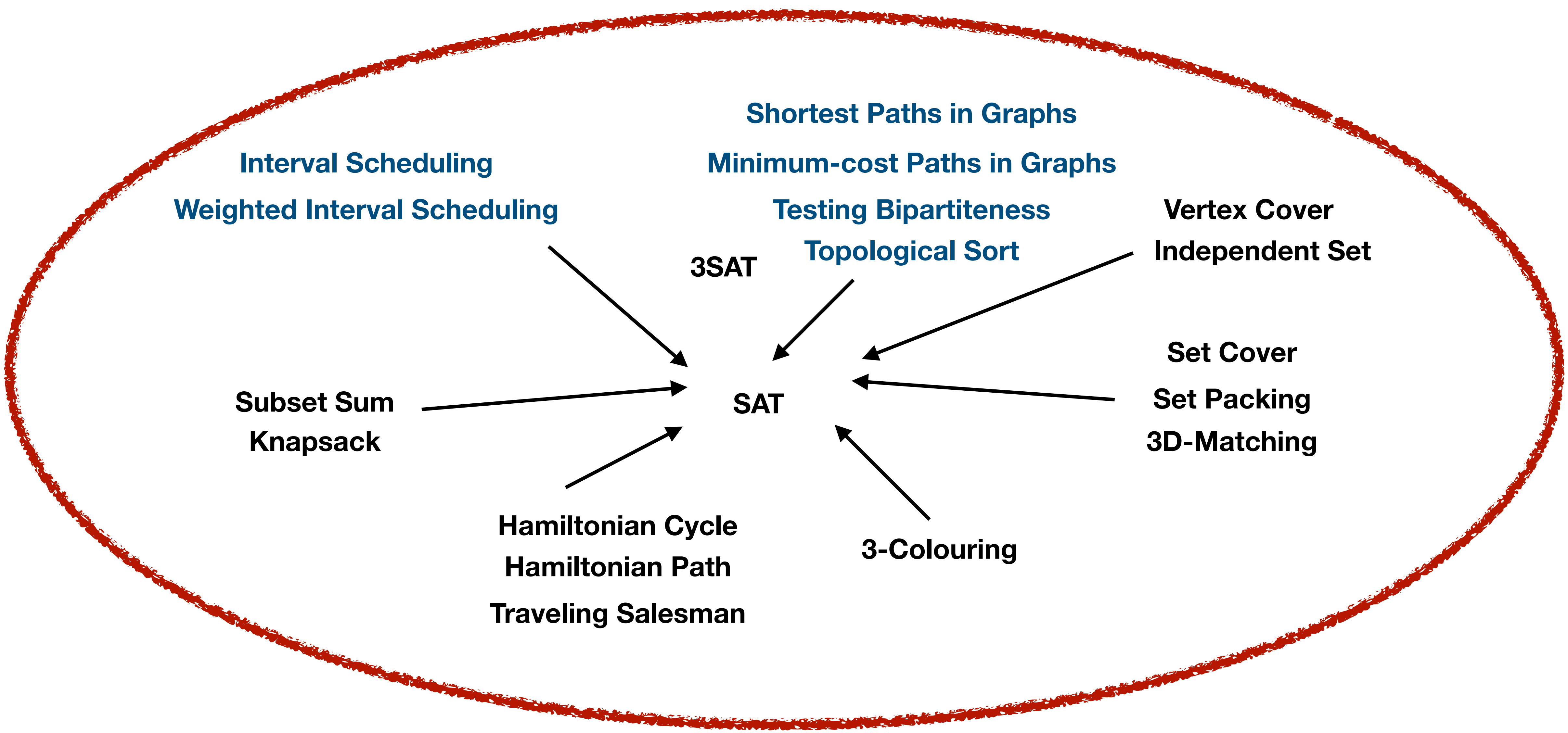
NP-completeness



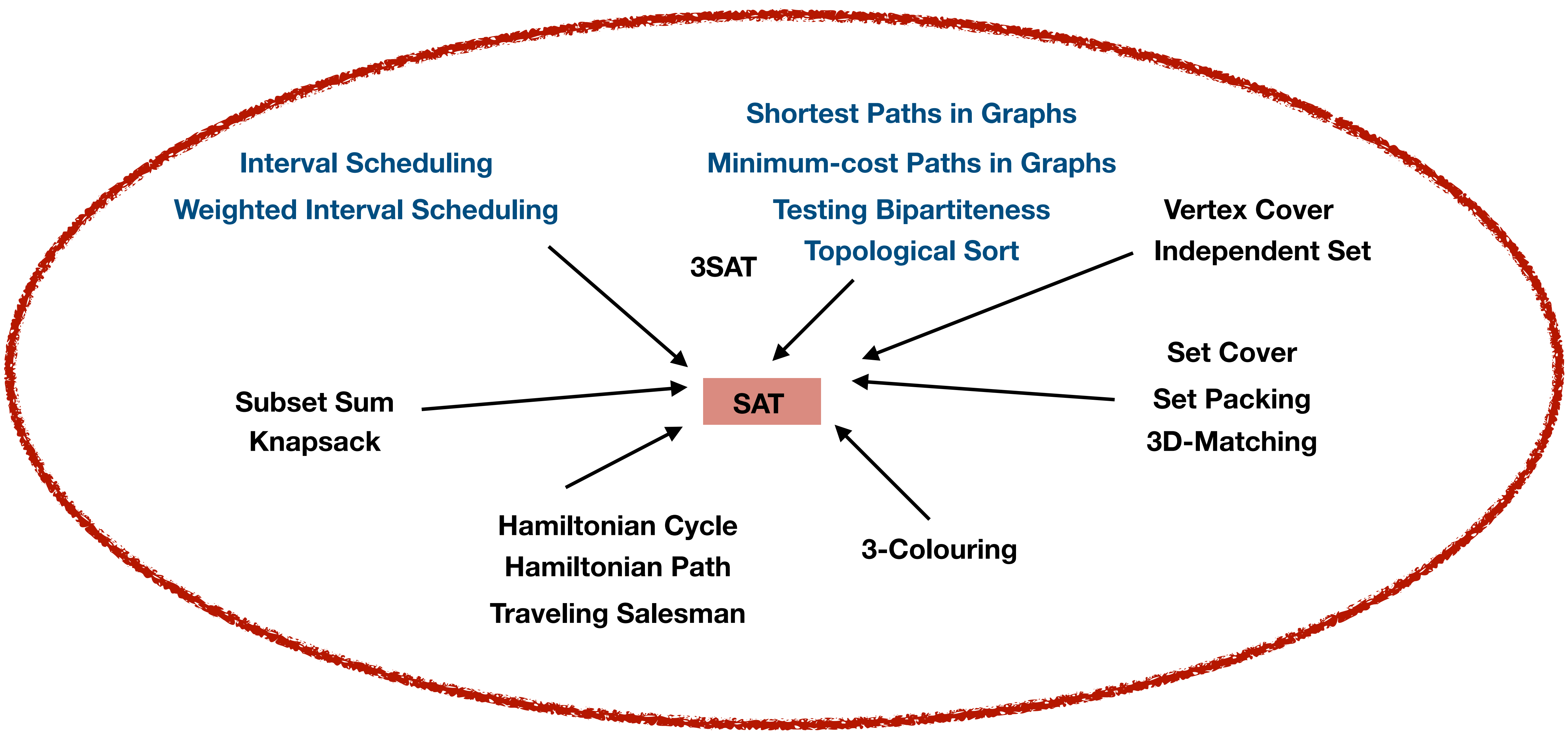
NP-completeness



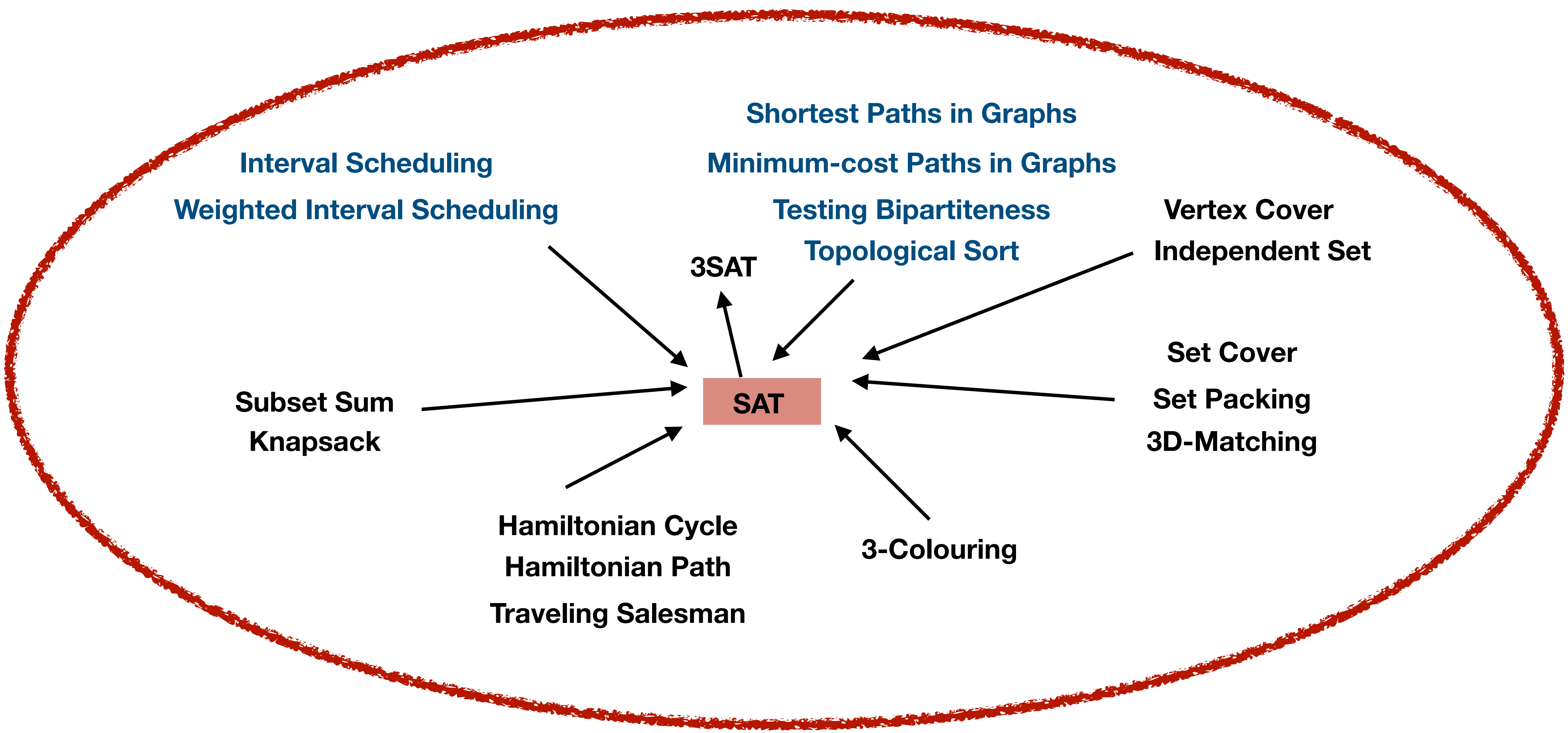
NP-completeness



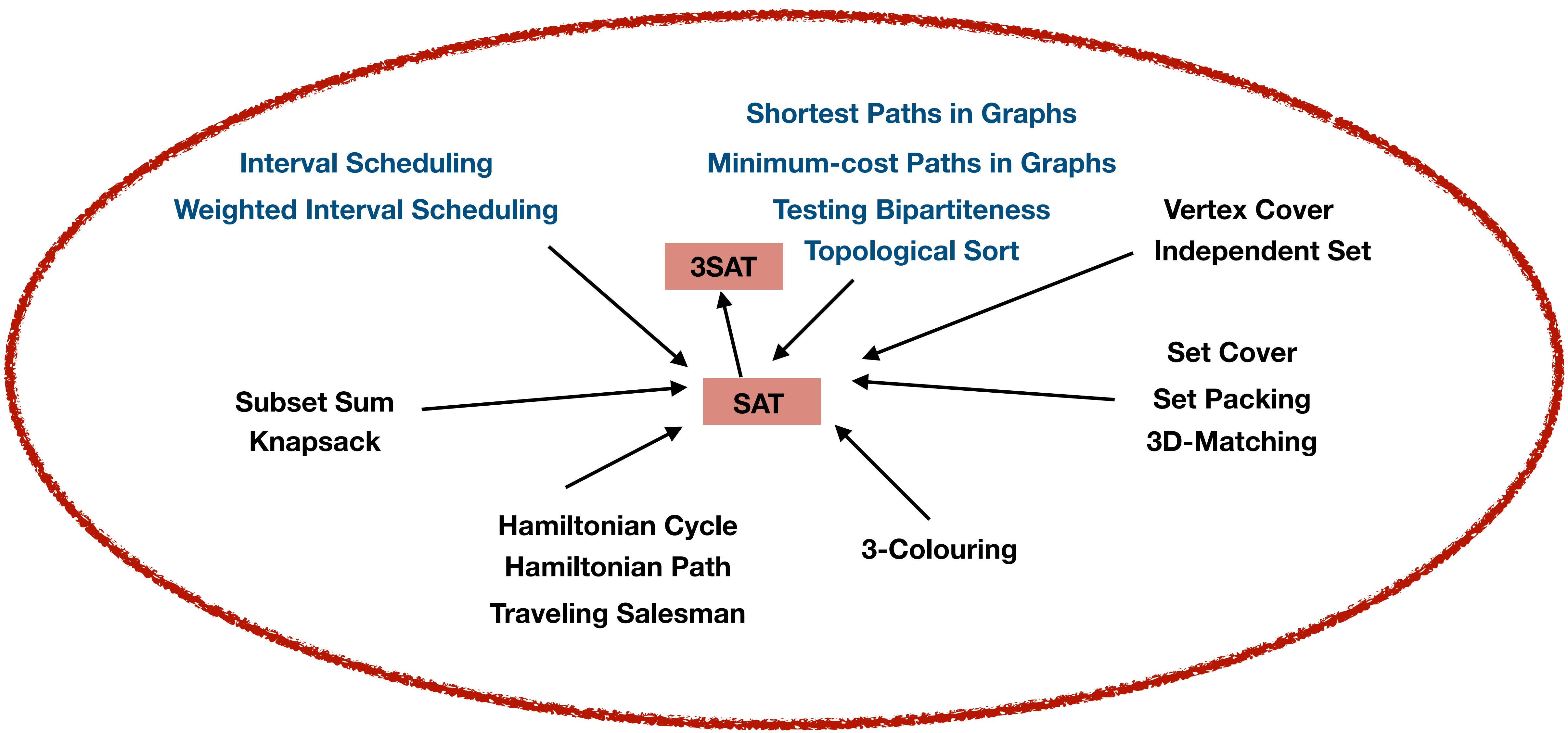
NP-completeness



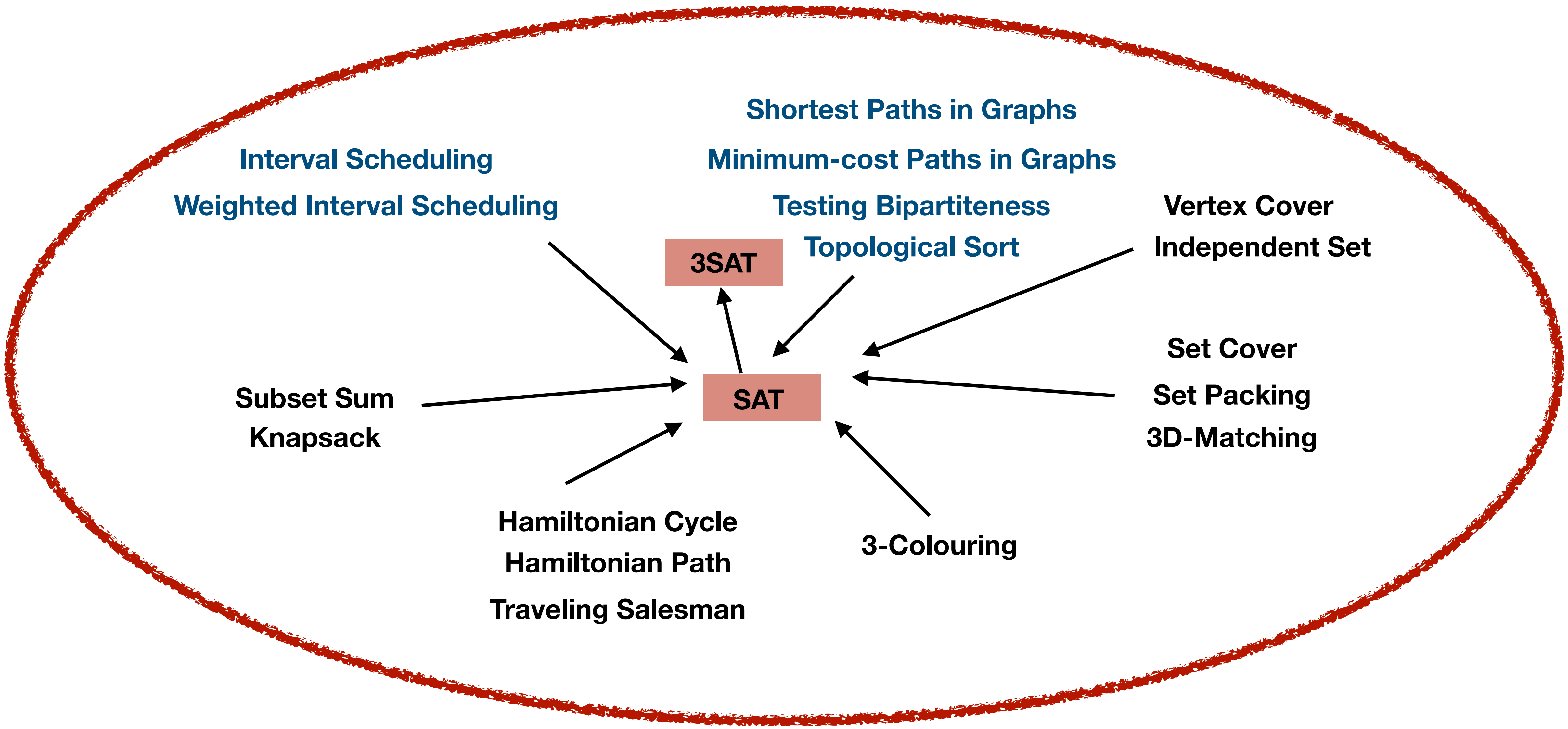
NP-completeness



NP-completeness

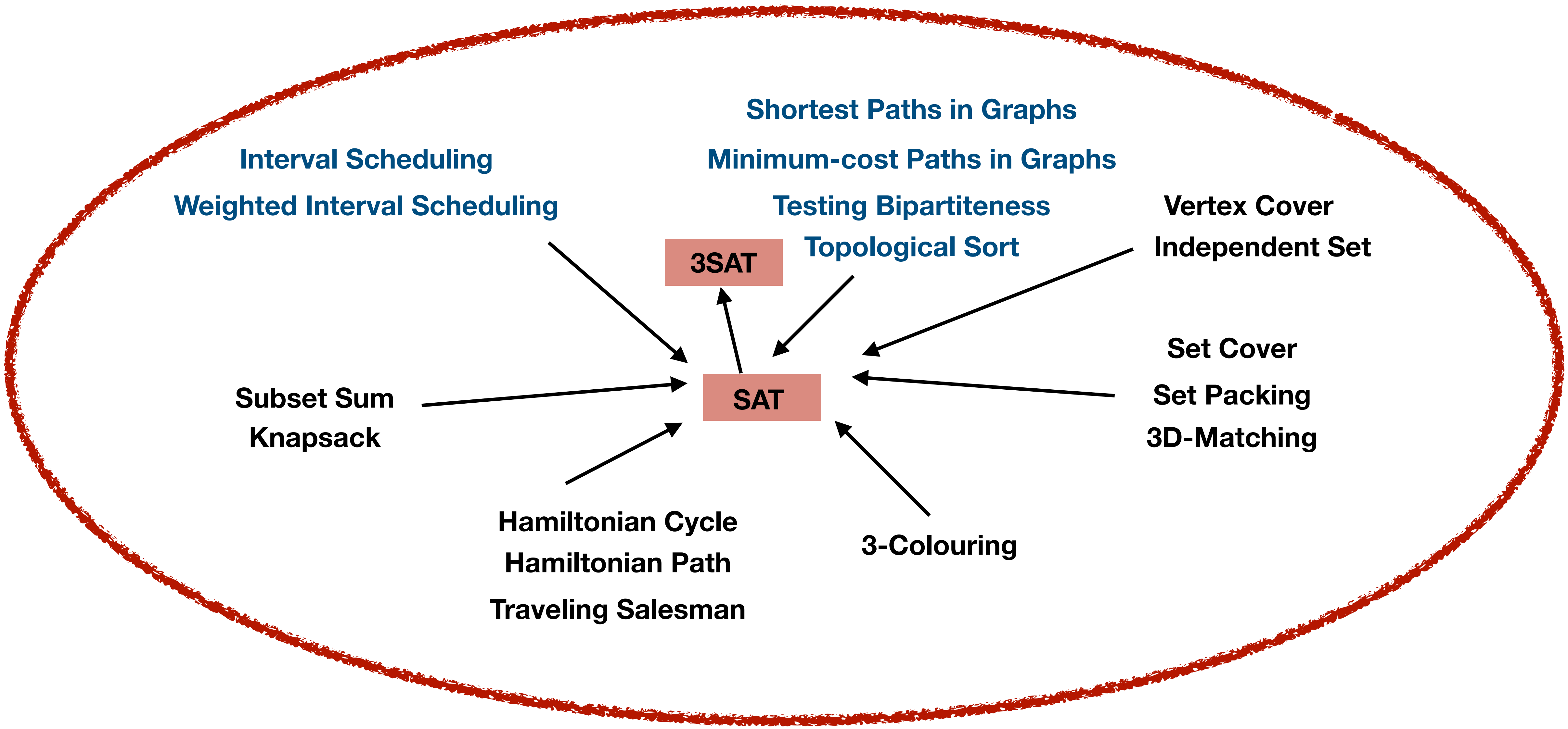


NP-completeness



The Cook-Levin Theorem (1971, 1973)

NP-completeness



The Cook-Levin Theorem (1971, 1973)

The proof uses a generic argument that if a problem has a solution which can be verified in polynomial-time, then it reduces in polynomial time to the SAT problem.

NP completeness

THE UNIVERSITY of EDINBURGH

DEGREE REGULATIONS & PROGRAMMES OF STUDY 2023/2024

Timetable information in the Course Catalogue may be subject to change.

DRPS : Course Catalogue : School of Informatics : Informatics

Undergraduate Course: Introduction to Theoretical Computer Science (INFR10059)

Course Outline			
School	School of Informatics	College	College of Science and Engineering
Credit level (Normal year taken)	SCQF Level 10 (Year 3 Undergraduate)	Availability	Available to all students
SCQF Credits	10	ECTS Credits	5
Summary	This course introduces the fundamental concepts of the theory of computer science, which include some of the greatest intellectual advances of the last century: what does `computing' mean? Are all `computers' basically the same? Can we tell whether our programs are `correct' - and what does `correct' mean, anyway? Can we solve problems in reasonable time, and can we tell whether we can? The course concentrates primarily on conceptual understanding, but adds enough detail to allow students to go on to further courses, and illustrates how the fundamental concepts are reflected throughout the discipline.		
Course description	The first section of the course asks the question, what does it mean to compute? We start with the finite automata introduced in earlier years, and then generalise to pushdown automata, and show that they have more power. Next we generalize further to very simple abstract general computers, and argue they can do everything real computers can do. We then ask, can we solve every computational question? The answer, with which Turing shocked the mathematicians of the 1930s, is "no", with a remarkably easy but beautiful argument (introduced at the end of Inf2-IADS INFR08026). We then explore some different, but always equivalent, ways of defining "a computer". We finish the section by asking how we can compare the difficulty of different problems, and introduce the idea of "reduction" as a way of compiling one problem into another. Technically, this covers register machines, undecidability, Turing machines, and reductions. The second section thinks about how hard it is to solve solvable problems, leading to one of the most important problems in all mathematics, and the foundation of internet security. We start by reprising Inf2-IADS INFR08026 analysis of algorithms, and then discuss the idea of classifying problems as `tractable' (easy) or `intractable' (hard). We find that the idea of algorithms whose running time grows polynomially in the problem size is a good mathematical definition of `tractable', though not always a practical one. After making this more precise, we ask what happens if we're allowed to just check all the possible answers in parallel - does this give us more problem-solving power? The question is made precise by the concept of NP, and we show that there are "hardest" such problems, such as the famous Travelling Salesman. Although the question is easy to ask, nobody knows how to answer it. This is P = NP - if you can solve it, you win a million dollars, and fame for as long as civilization lasts. So far, NP problems are very hard to solve in practice, so we discuss how to deal with them. We finish the section by talking about much harder problems still. Technically, this section covers P, NP, hardness and completeness, Cook's Theorem, P = NP, and the complexity hierarchy above NP. The third section takes brief look at a different way of seeing computation. Haskell needn't be seen as a programming language, it can be the computer itself. We'll show how the lambda-calculus (on which Haskell is based) can do all the computing our other models could, and how the halting problem was actually first solved (or rather unsolved) within lambda-calculus.		

Entry Requirements (not applicable to Visiting Students)			
Pre-requisites		Co-requisites	
Prohibited Combinations		Other requirements	This course is open to all Informatics students including those on joint degrees. It is also open to students in the School of Mathematics.

Information for Visiting Students	
Pre-requisites	None
High Demand Course?	Yes

Course Delivery Information	
Academic year 2023/24, Available to all students (SV1)	Quota: None
Course Start	Semester 1
Timetable	Timetable
Learning and Teaching activities (Further Info)	Total Hours: 100 (Programme Level Learning and Teaching Hours 2, Directed Learning and Independent Learning Hours 98)
Assessment (Further Info)	Written Exam 80 %, Coursework 20 %, Practical Exam 0 %
Additional Information (Assessment)	An exam provides the main assessment. In order to ensure coverage of the three major sections, the format will be three compulsory easier questions, and a choice of one of two longer questions. Assessed coursework will be issued at two points, containing mainly relatively straightforward exercises designed to reinforce basics, the first coursework being formative and the second being summative. Additional formative work in tutorial sheets will stretch those who wish. You should expect to spend approximately 15 hours on the coursework for this course.
Feedback	Formative feedback is given verbally in tutorials, and in writing for the first exercise. Summative and formative feedback is given in writing for the second exercise.

Traveling Salesman

The Cook-Levin Theorem (1971, 1973)

The proof uses a generic argument that if a problem has a solution which can be verified in polynomial-time, then it reduces in polynomial time to the SAT problem.

NP-completeness

NP-completeness

- What does the NP-completeness of SAT mean?

NP-completeness

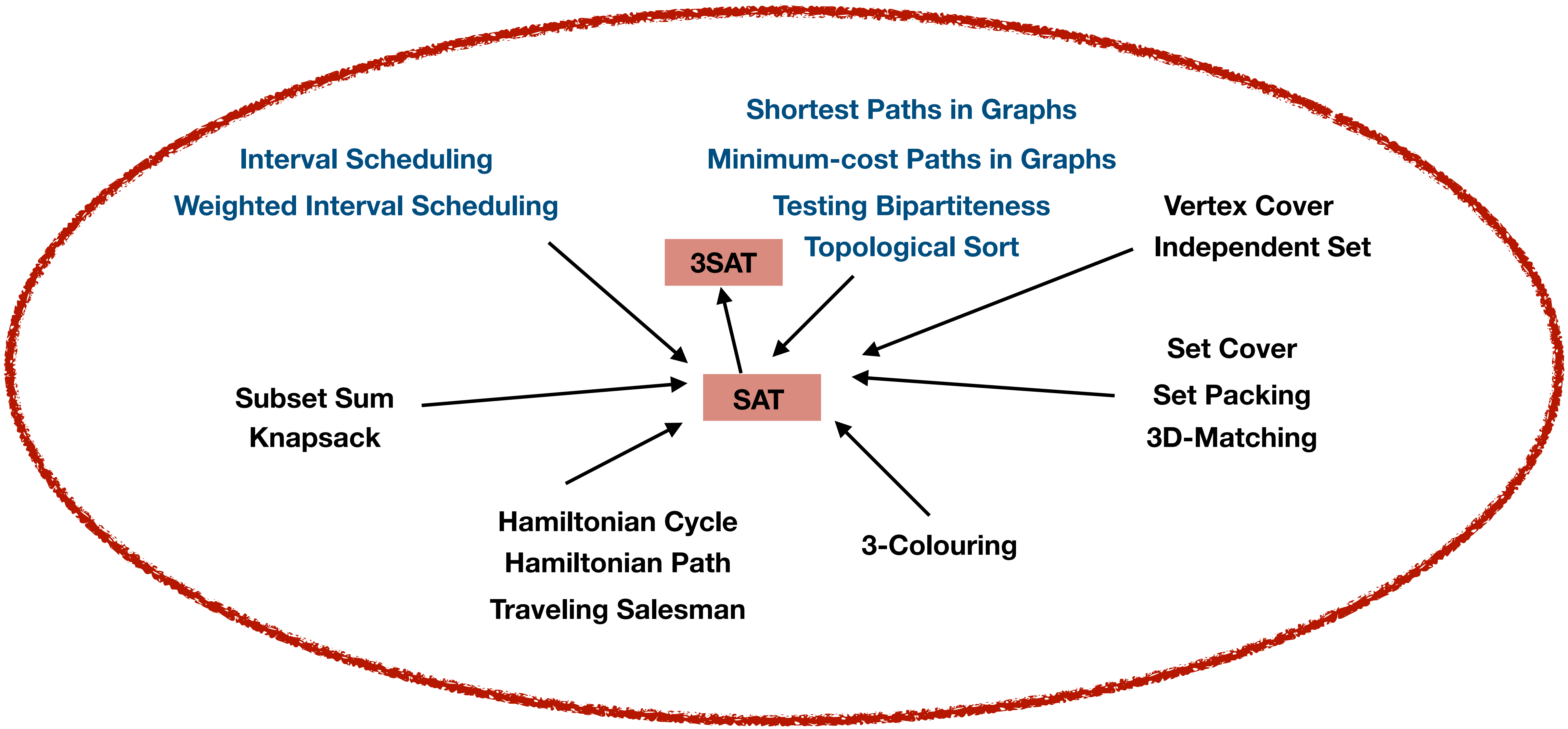
- What does the NP-completeness of SAT mean?
  - It means that it is *at least as hard to solve* as any other problem in NP.

NP-completeness

- What does the NP-completeness of SAT mean?
  - It means that it is *at least as hard to solve* as any other problem in NP.
  - In particular, if we had a polynomial-time algorithm for solving SAT, *we could solve any other problem in NP*, via the reduction (the arrow).

Wooclap!

NP-completeness



NP-completeness

- What does the NP-completeness of SAT mean?
  - It means that it is *at least as hard to solve* as any other problem in NP.
  - In particular, if we had a polynomial-time algorithm for solving SAT, *we could solve any other problem in NP*, via the reduction (the arrow).
  - At this stage, that doesn't necessarily say much.

NP-completeness

NP-completeness

- Some time passes, and we tried and tried to find a polynomial-time algorithm for SAT (or 3SAT) and we are still looking for one…

NP-completeness

- Some time passes, and we tried and tried to find a polynomial-time algorithm for SAT (or 3SAT) and we are still looking for one…
- This seems to suggest that SAT might be in some sense *harder to solve* than e.g., Interval Scheduling or Testing Bipartiteness.

NP-completeness

- Some time passes, and we tried and tried to find a polynomial-time algorithm for SAT (or 3SAT) and we are still looking for one…

- This seems to suggest that SAT might be in some sense *harder to solve* than e.g., Interval Scheduling or Testing Bipartiteness.

  - We know of course that it is at least as hard to solve, by virtue of being NP-complete.

NP-completeness

- Some time passes, and we tried and tried to find a polynomial-time algorithm for SAT (or 3SAT) and we are still looking for one…
- This seems to suggest that SAT might be in some sense *harder to solve* than e.g., Interval Scheduling or Testing Bipartiteness.
    - We know of course that it is at least as hard to solve, by virtue of being NP-complete.
    - But this seems to suggest that some problems in NP are harder than others.

NP-completeness

NP-completeness

- After a while, we gave up on SAT and decided to try to solve our new favourite problem, Vertex Cover, in polynomial time.

NP-completeness

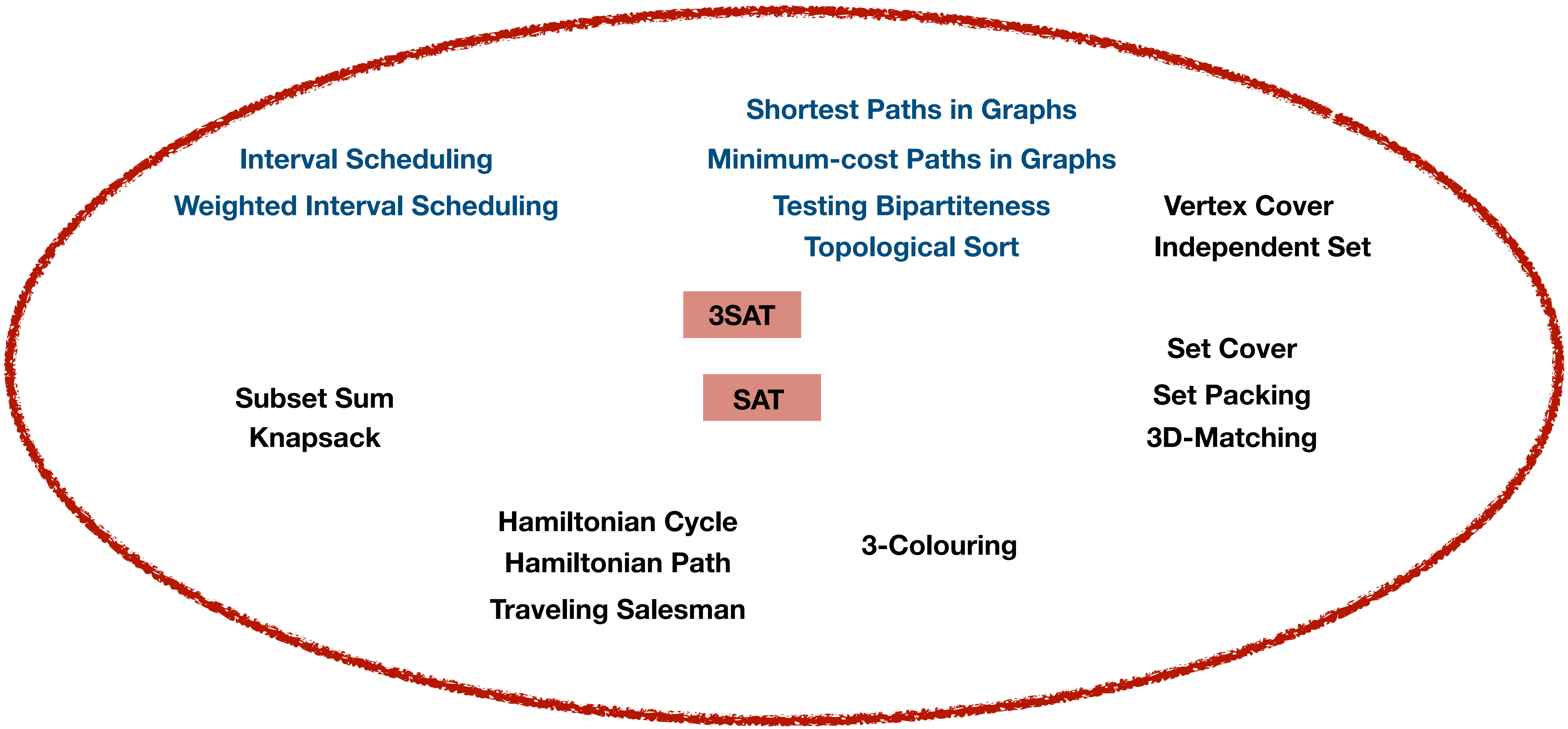
- After a while, we gave up on SAT and decided to try to solve our new favourite problem, Vertex Cover, in polynomial time.

- We tried hard and we failed… We are still looking for a polynomial-time algorithm.

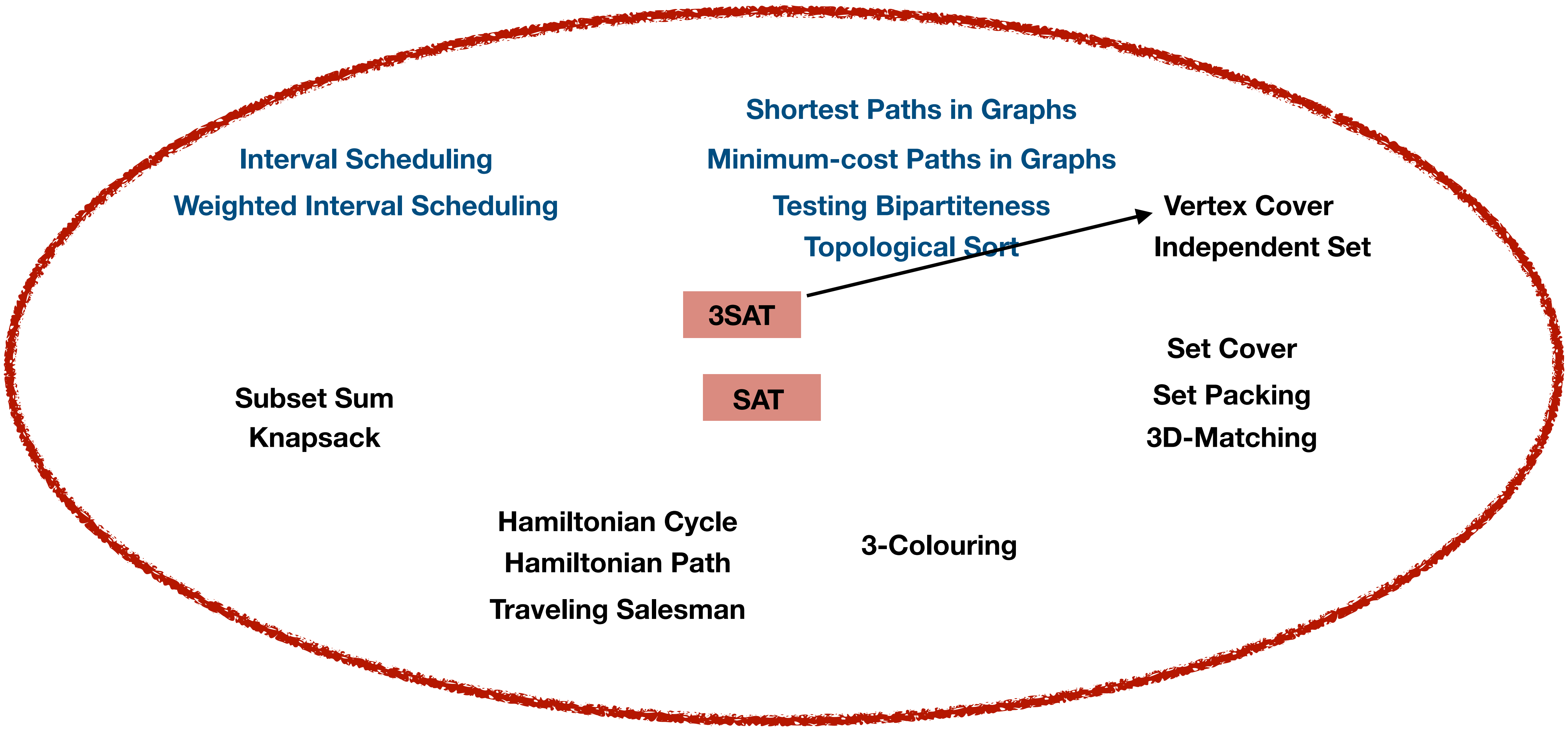
NP-completeness

- After a while, we gave up on SAT and decided to try to solve our new favourite problem, Vertex Cover, in polynomial time.
- We tried hard and we failed… We are still looking for a polynomial-time algorithm.
- Hmm, maybe Vertex Cover is also harder to solve than, say, Interval Scheduling or Testing Bipartiteness…

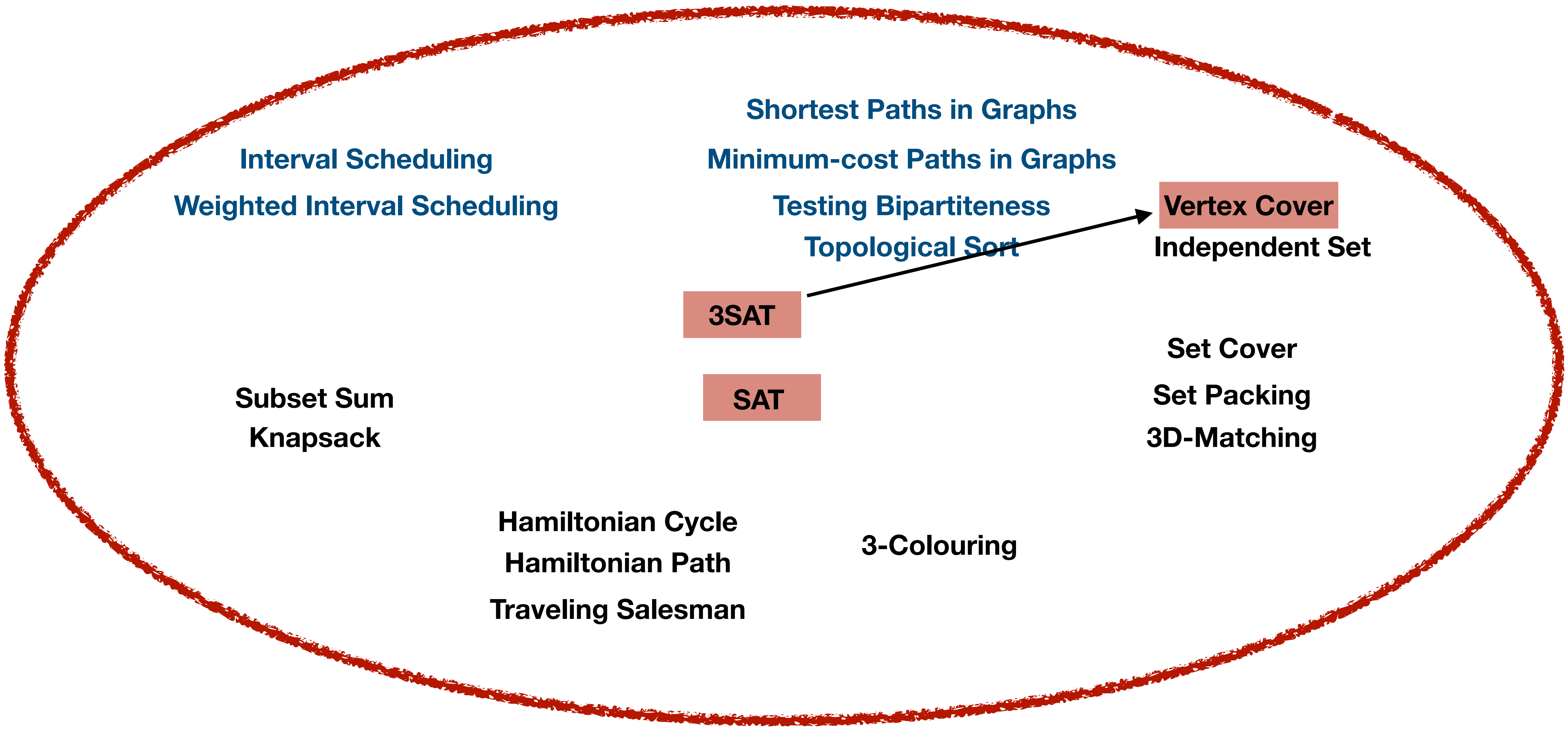
NP-completeness



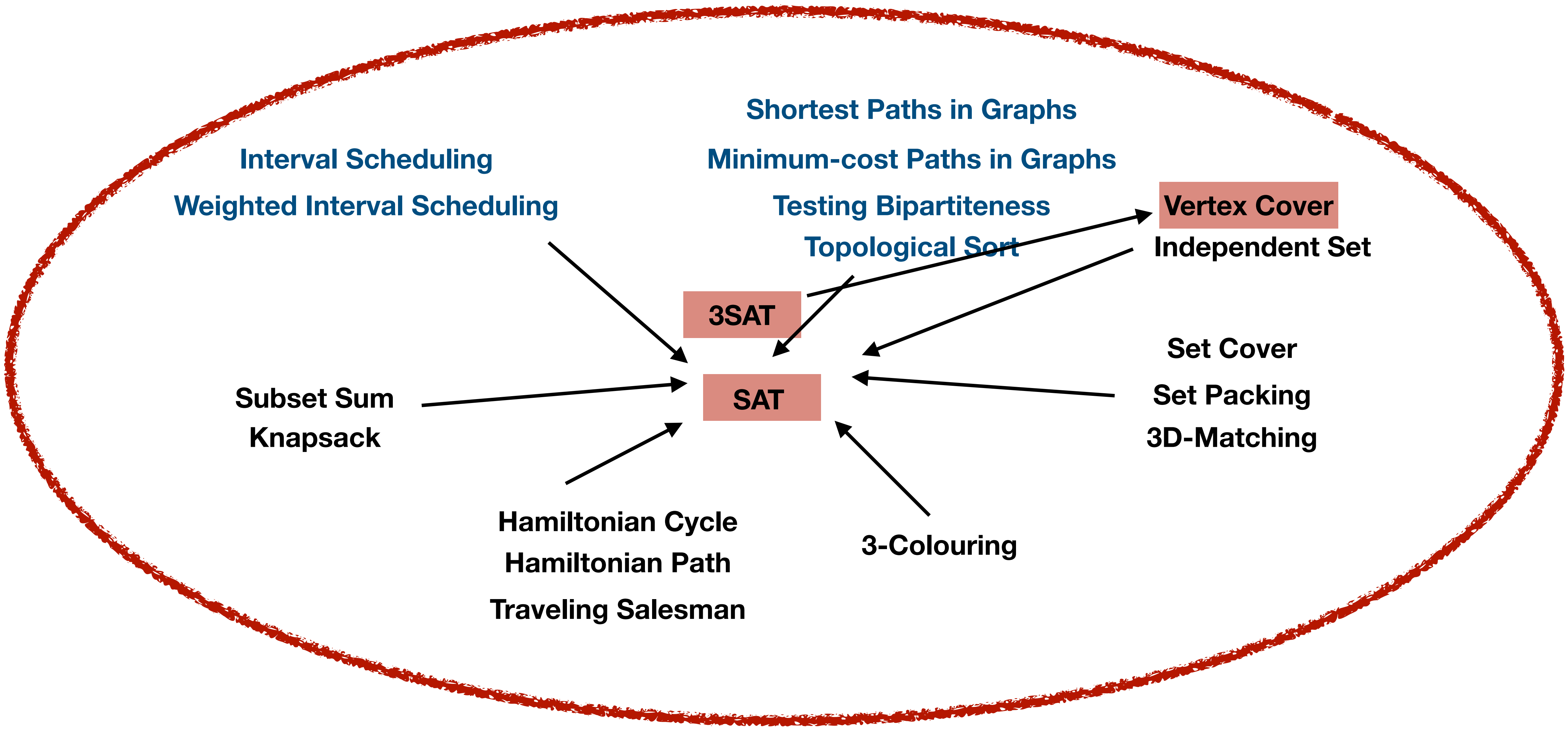
NP-completeness



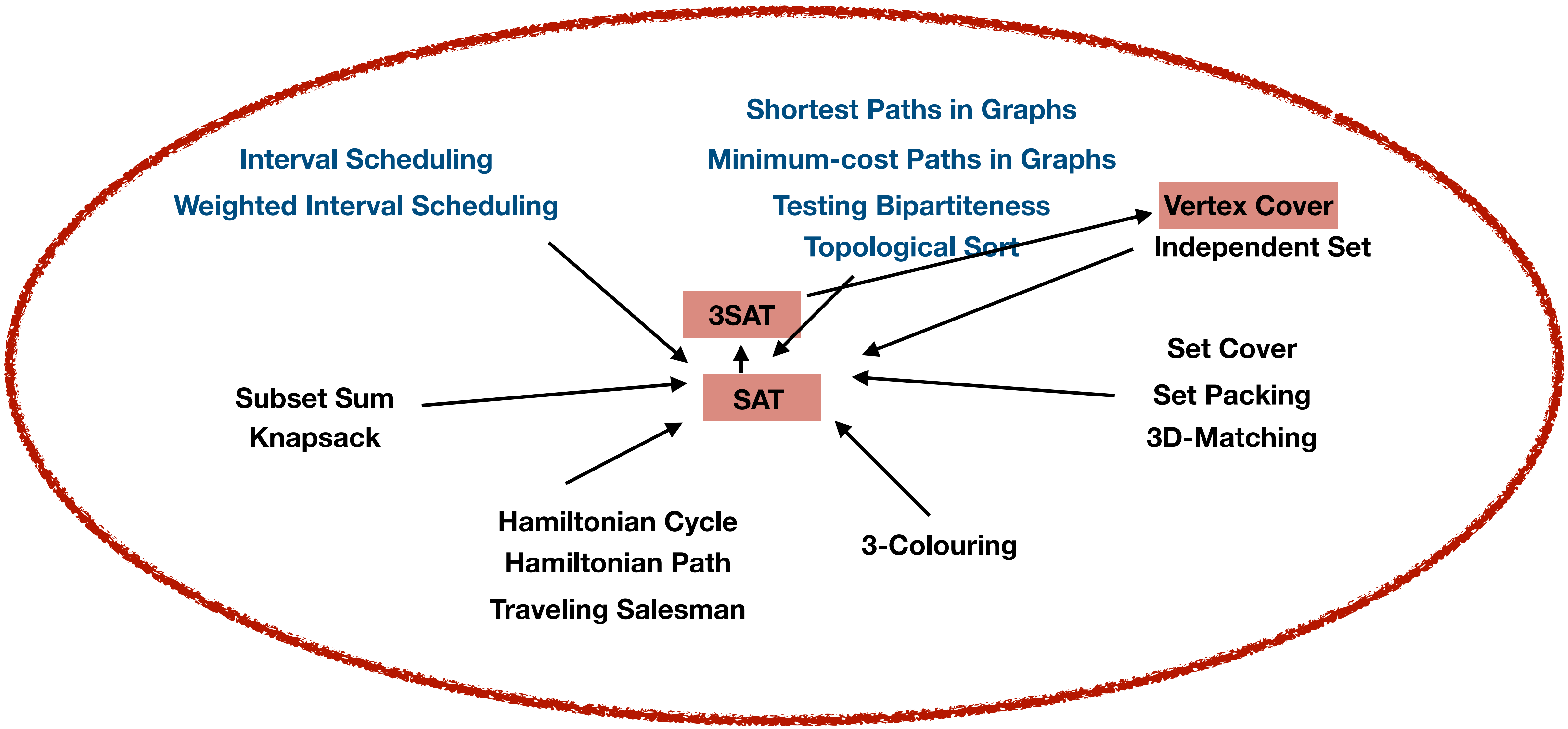
NP-completeness



NP-completeness



NP-completeness



NP-completeness

NP-completeness

- Vertex Cover is NP-complete.

NP-completeness

- Vertex Cover is NP-complete.
- This means that it is at least as hard as any problem in NP, including SAT.

NP-completeness

- Vertex Cover is NP-complete.
- This means that it is at least as hard as any problem in NP, including SAT.
- But we really tried to solve SAT in polynomial-time… No wonder we failed to solve Vertex Cover too!

NP-completeness

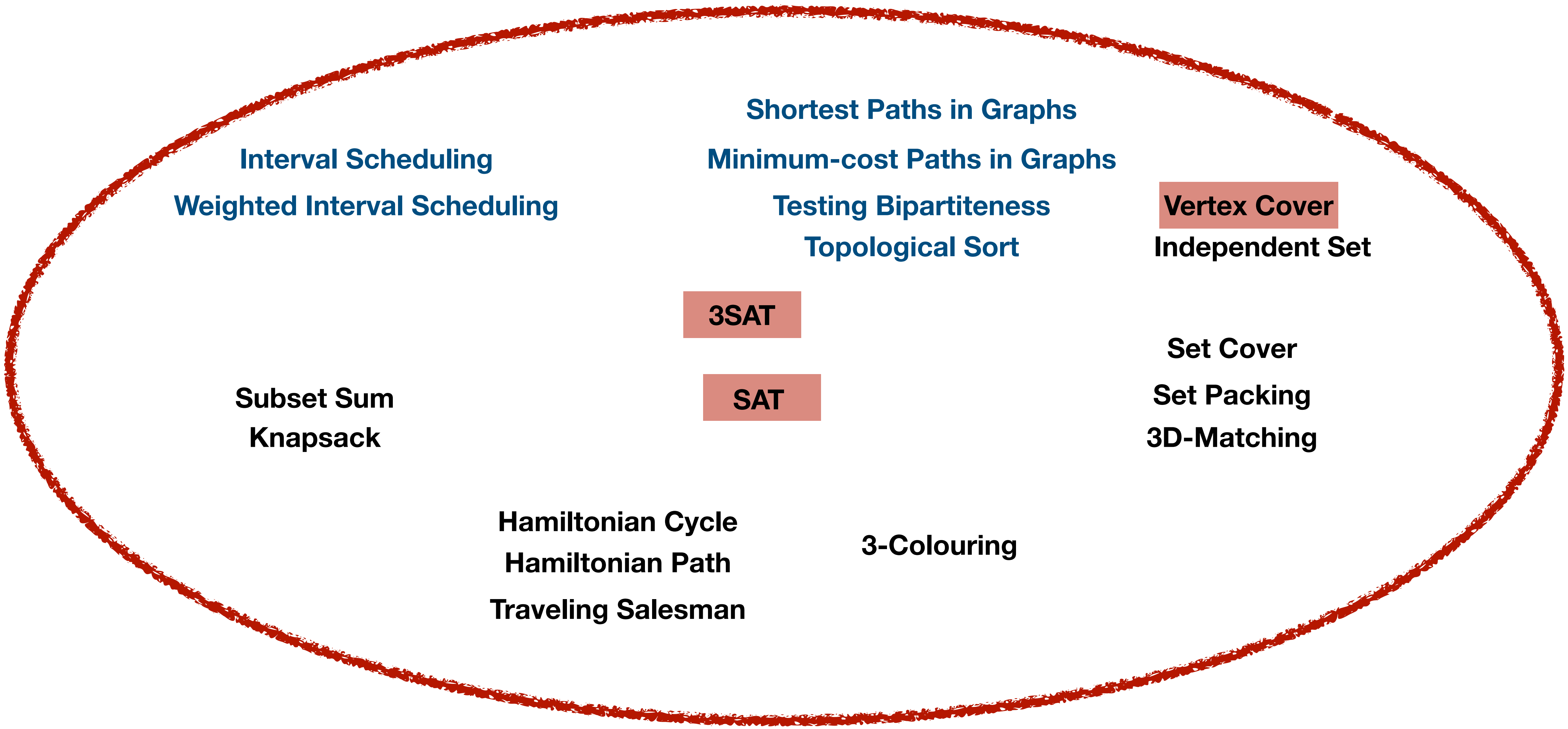
NP-completeness

- Ok, let's try to solve Independent Set in polynomial time then.

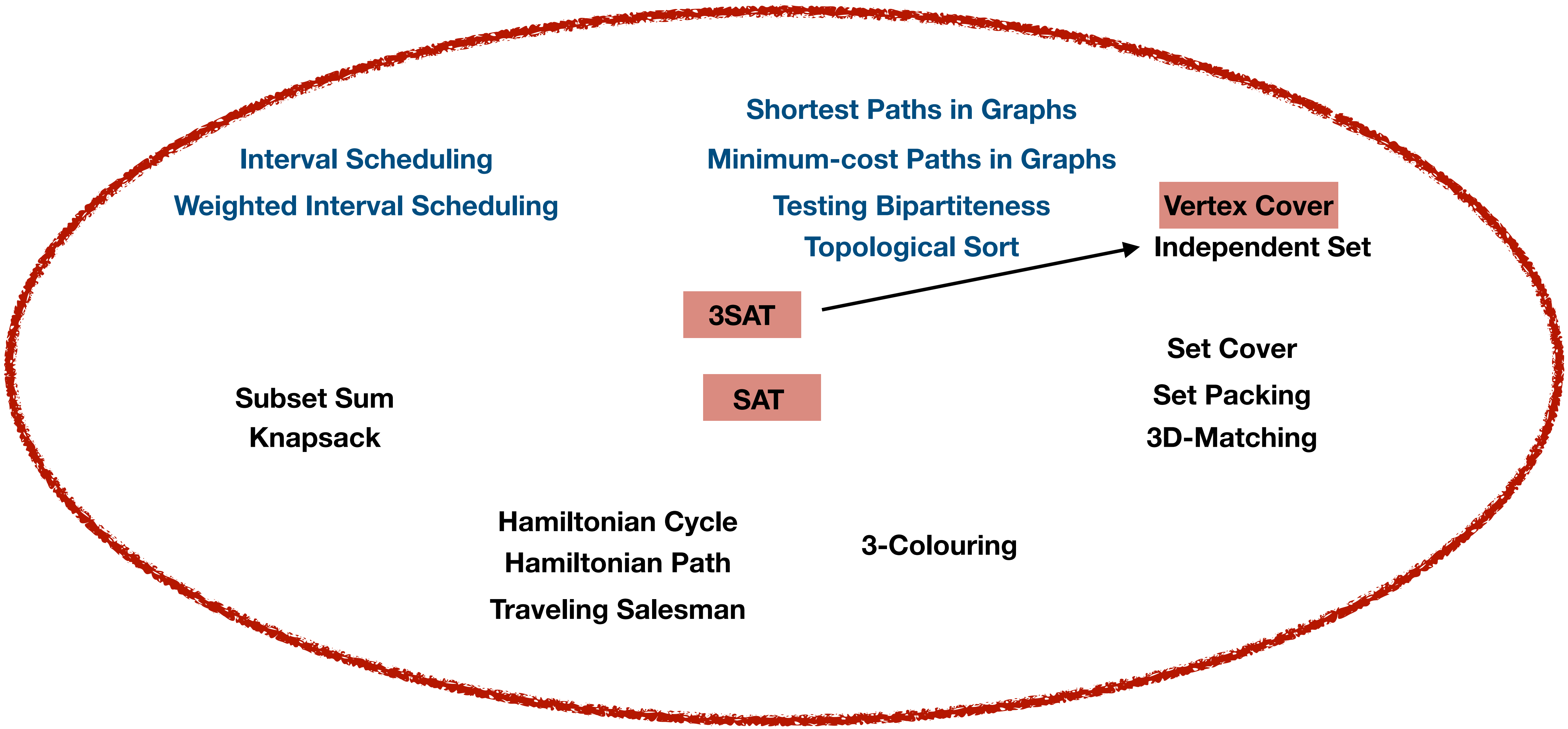
NP-completeness

- Ok, let's try to solve Independent Set in polynomial time then.
- Arghh, we can't solve that either!

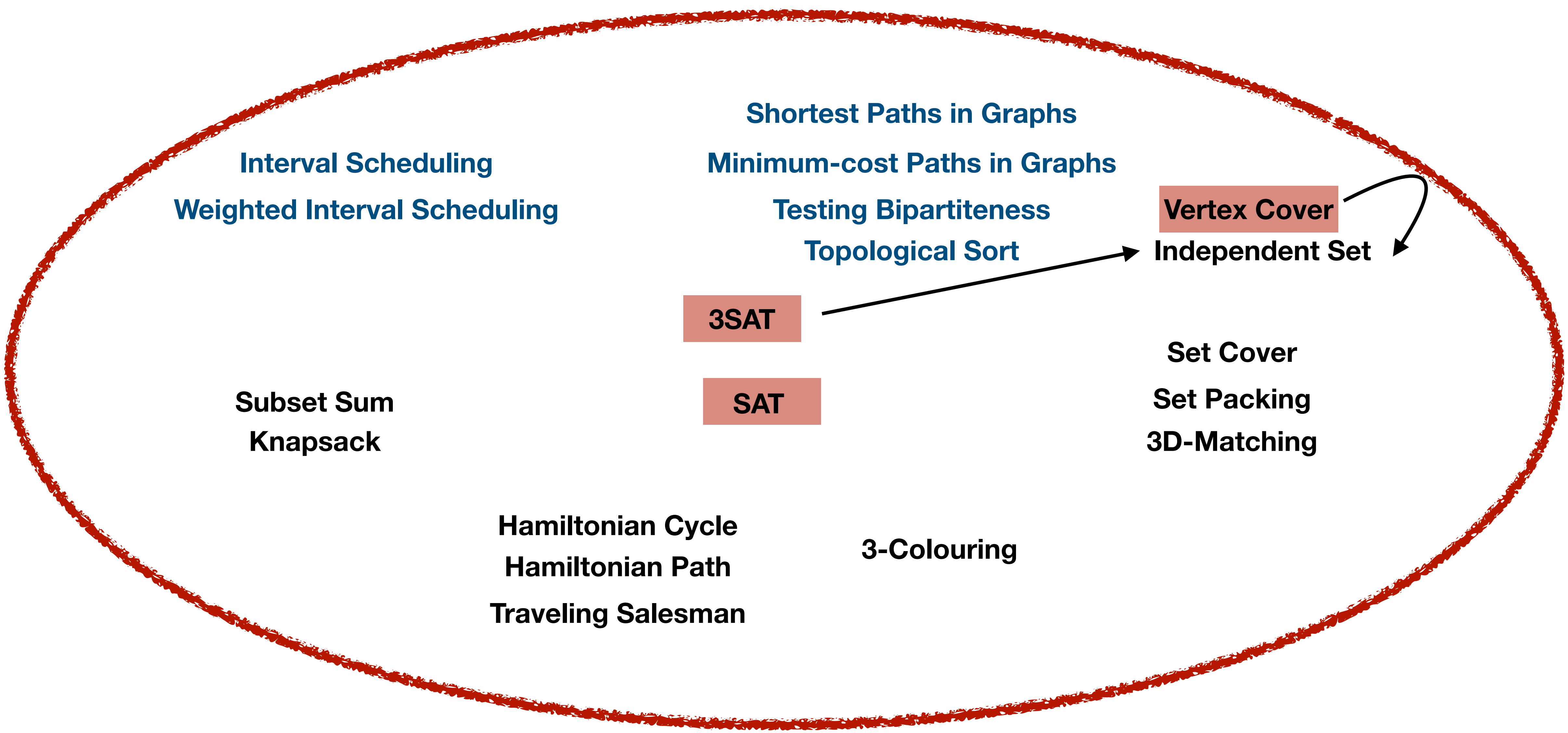
NP-completeness



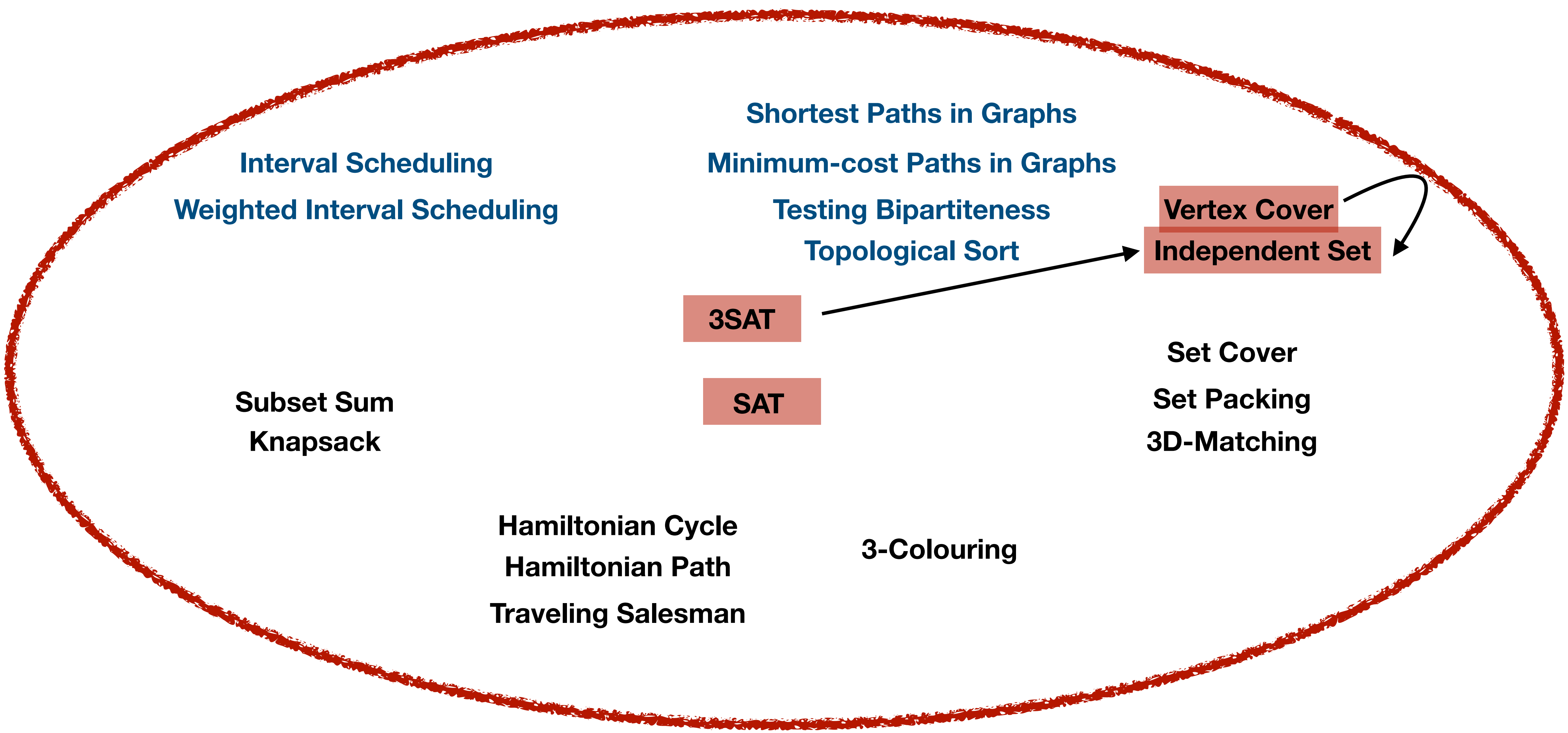
NP-completeness



NP-completeness



NP-completeness



NP-completeness

NP-completeness

- Independent Set is NP-complete.

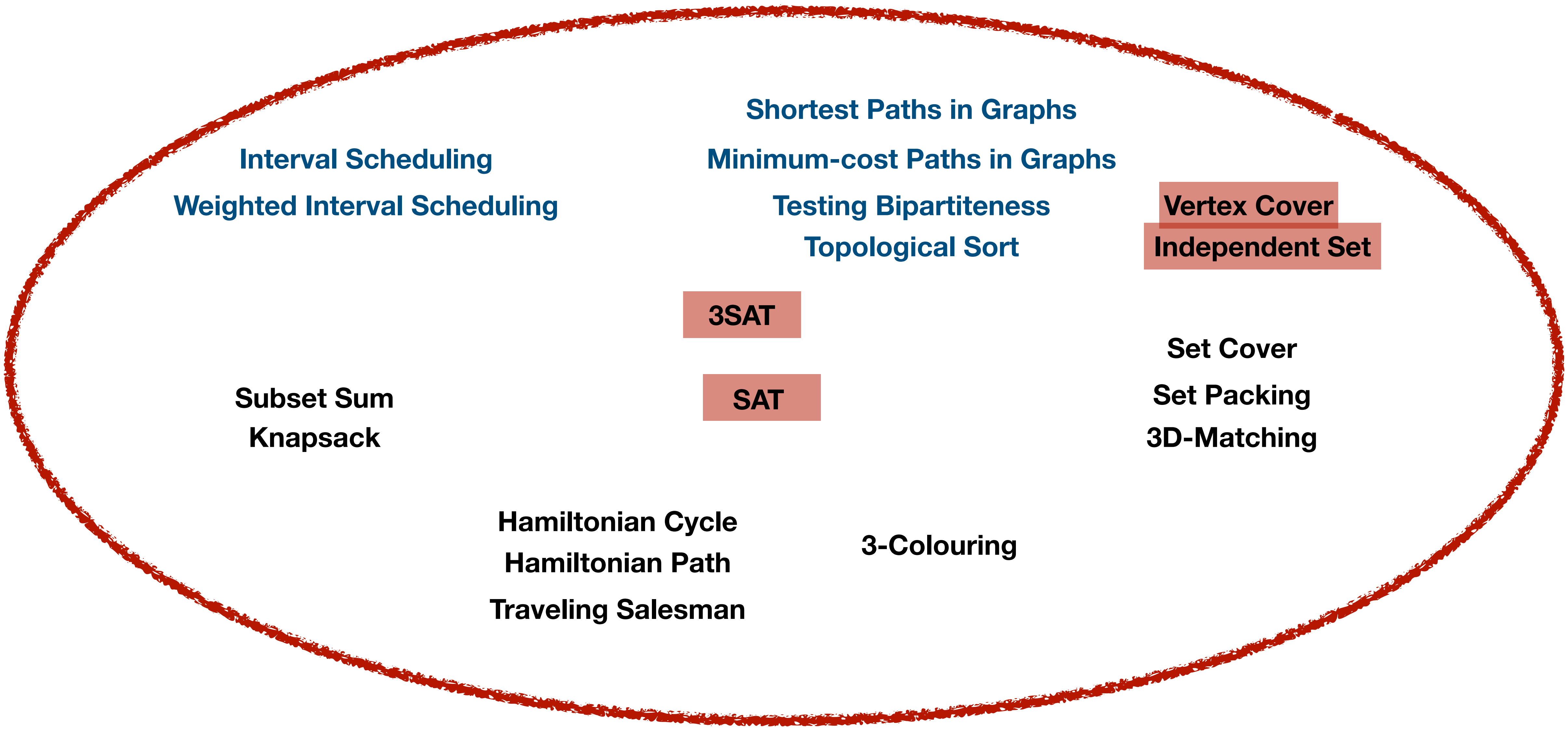
NP-completeness

- Independent Set is NP-complete.
- This means that it is at least as hard as any problem in NP, including SAT and Vertex Cover.

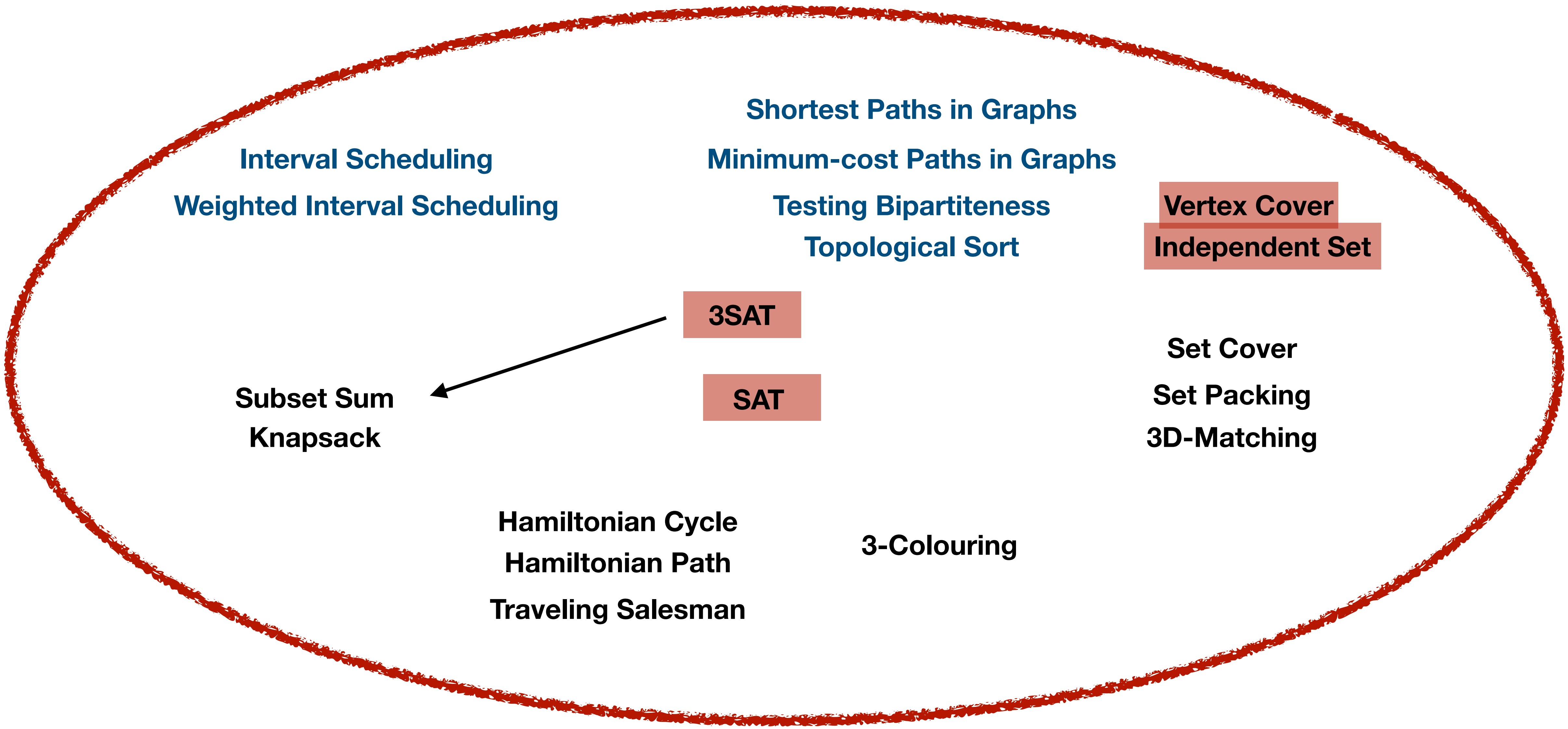
NP-completeness

- Independent Set is NP-complete.
- This means that it is at least as hard as any problem in NP, including SAT and Vertex Cover.
- But we really tried to solve SAT and Vertex Cover in polynomial-time… No wonder we failed to solve Independent Set too.

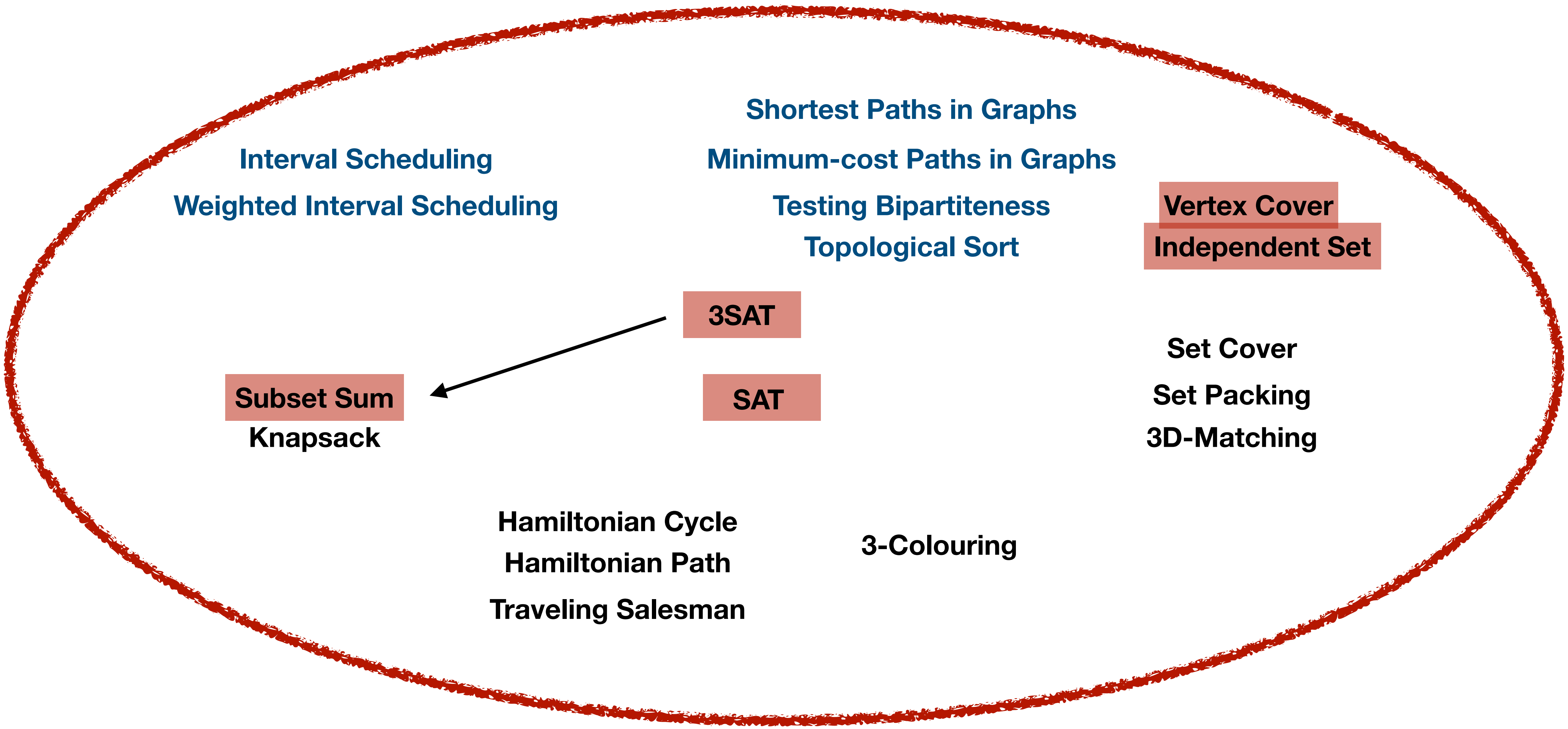
NP-completeness



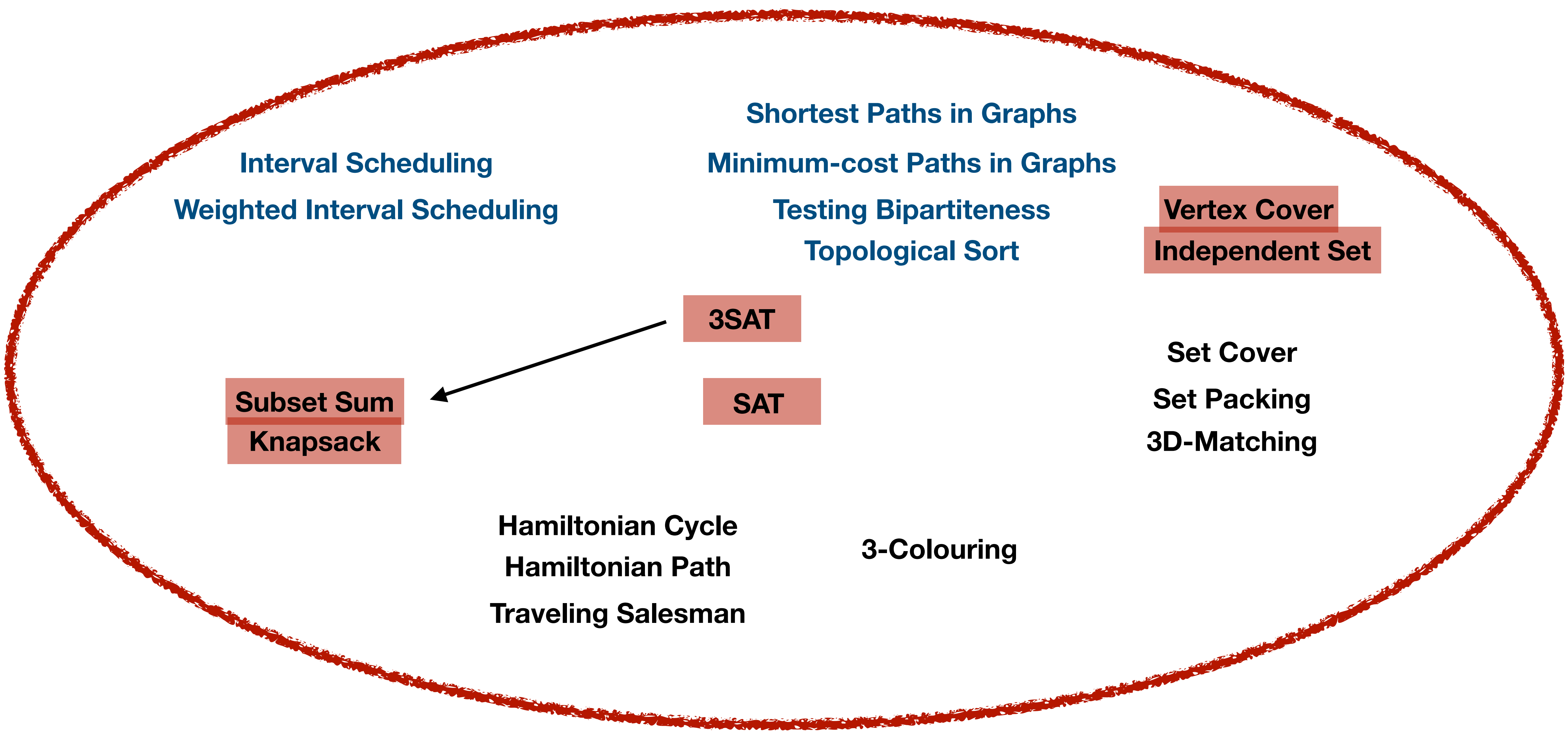
NP-completeness



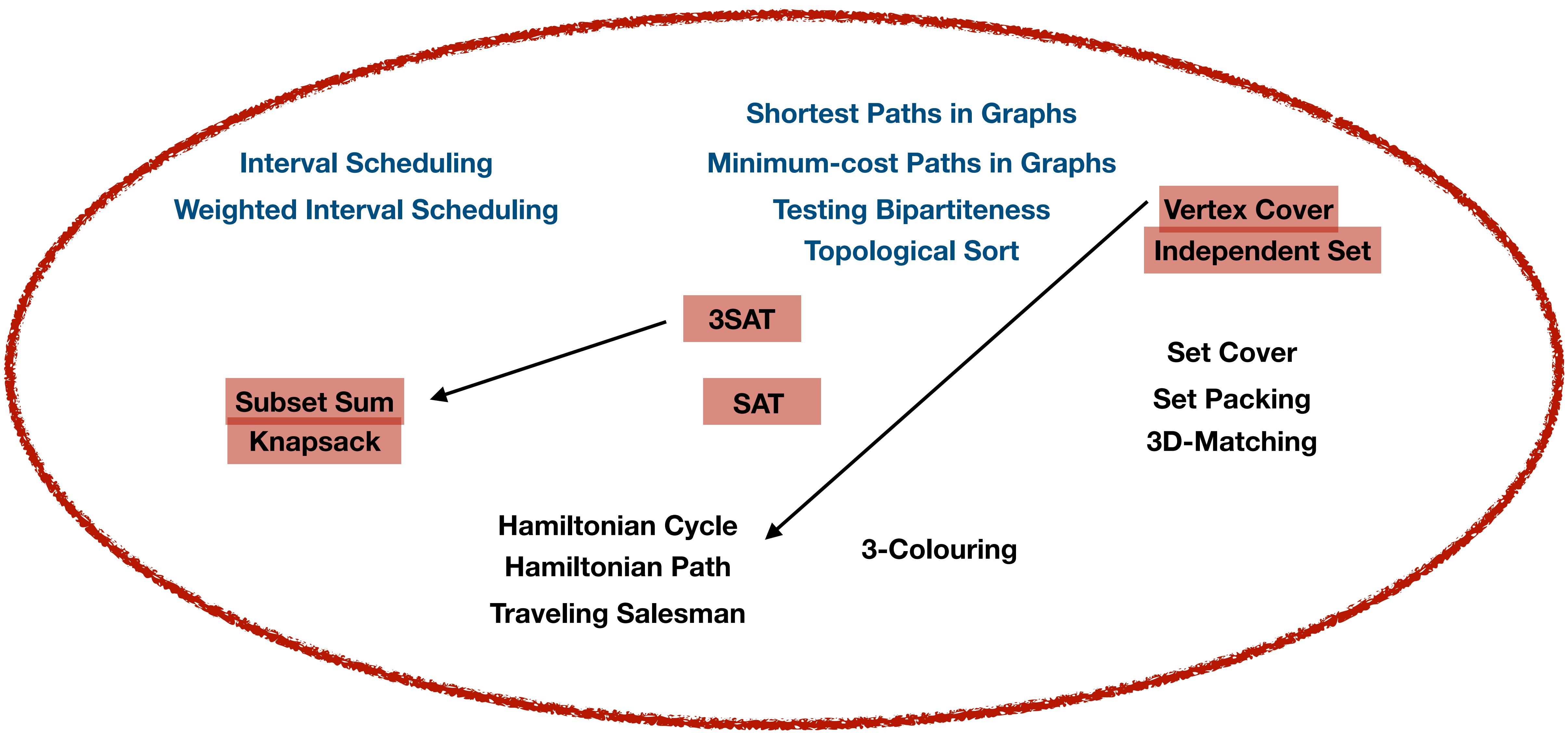
NP-completeness



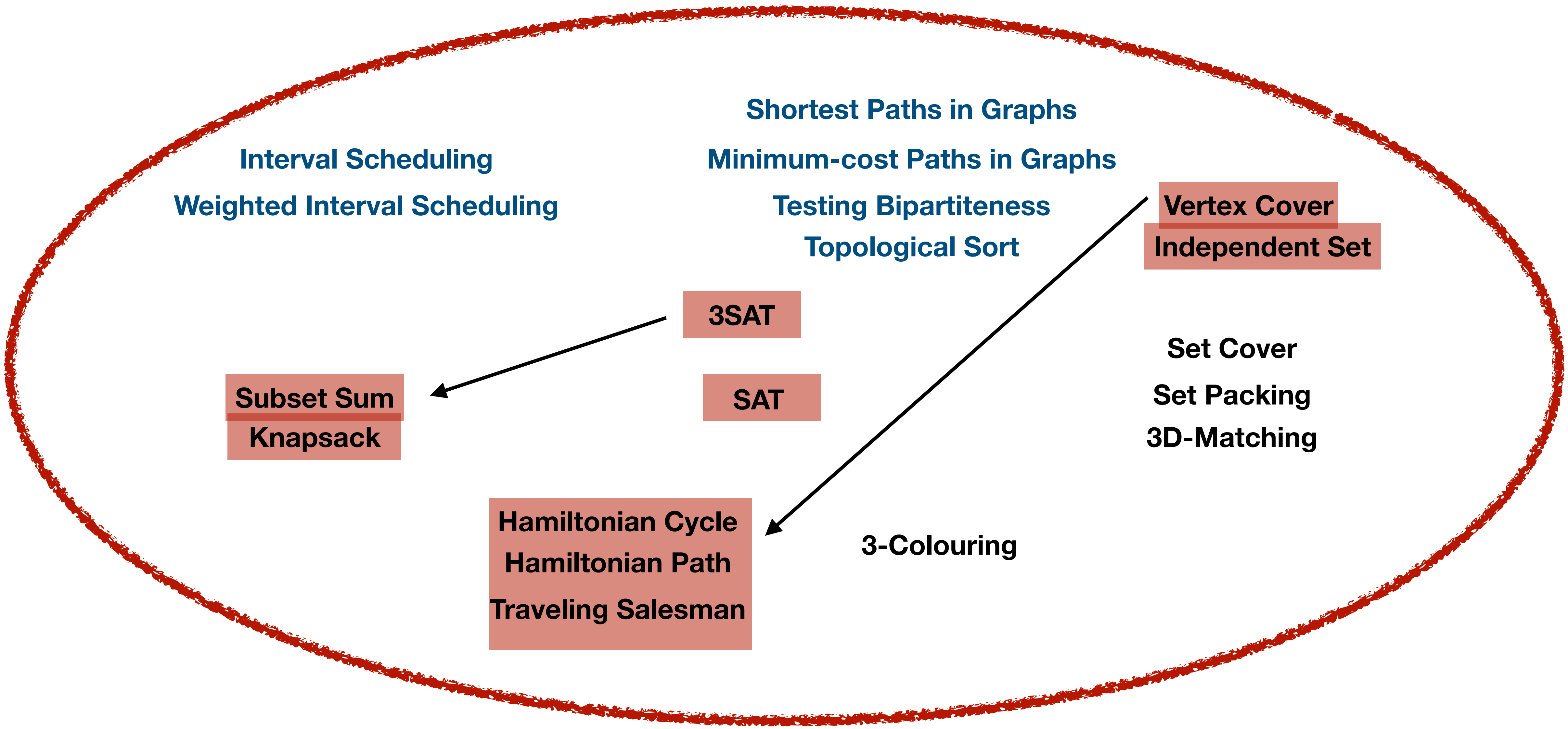
NP-completeness



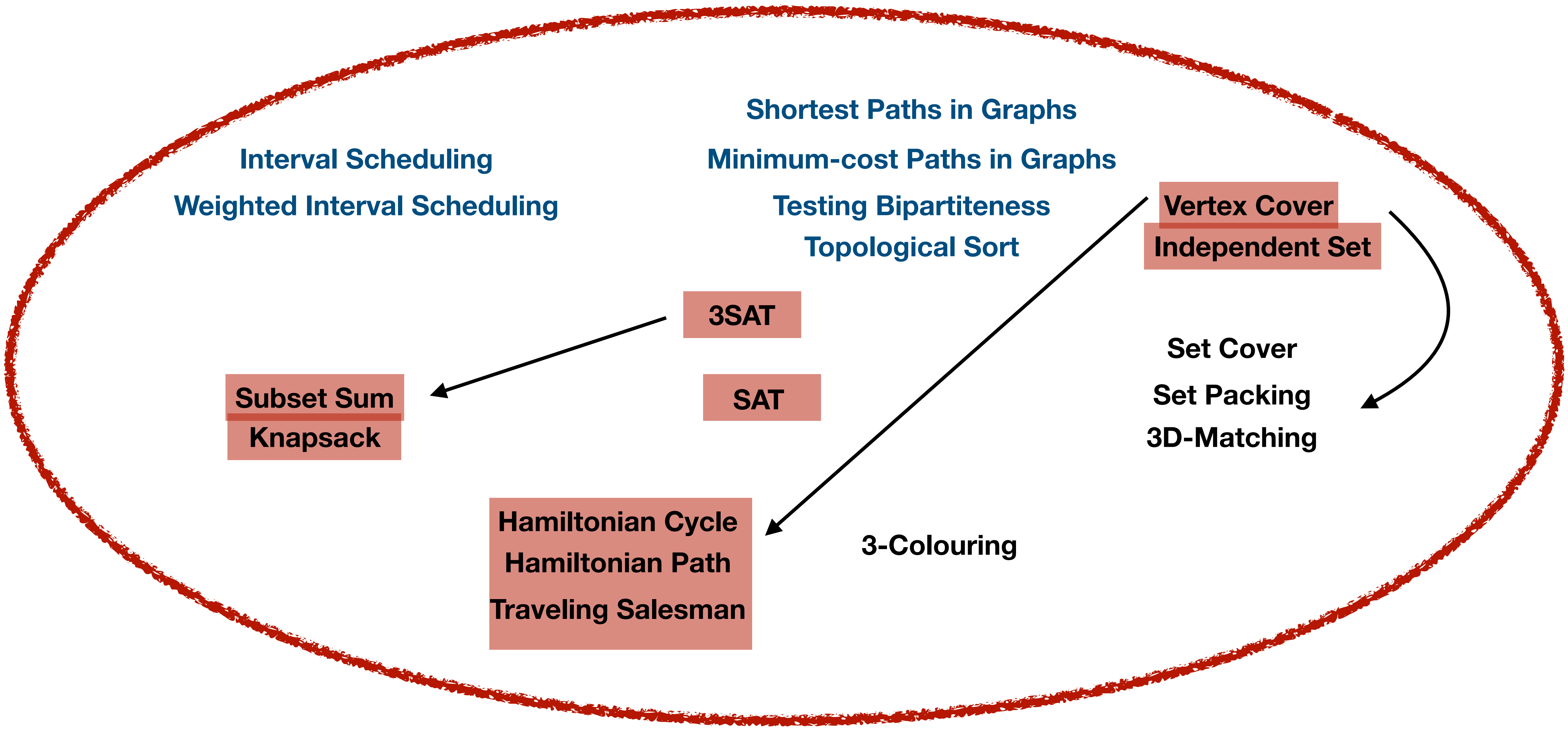
NP-completeness



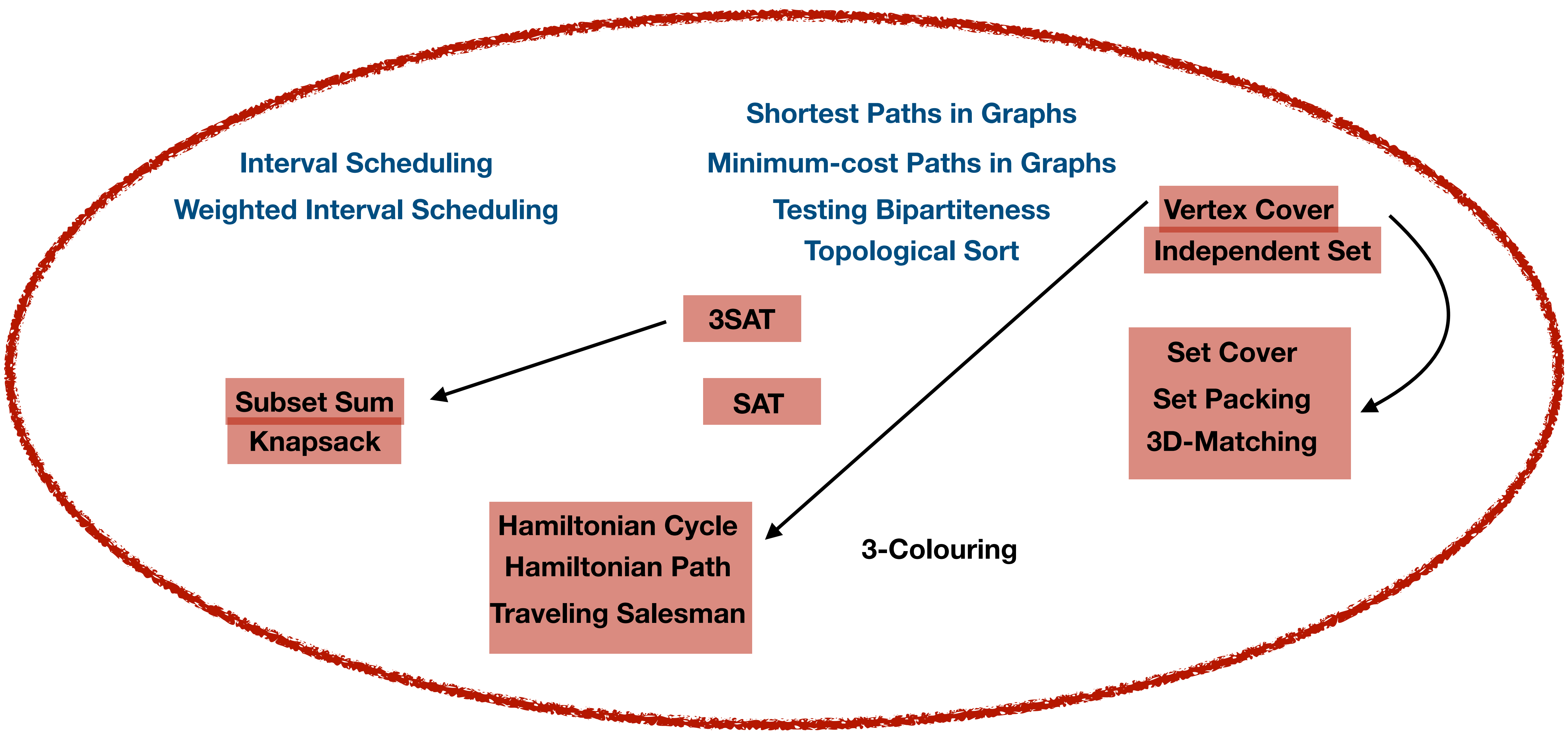
NP-completeness



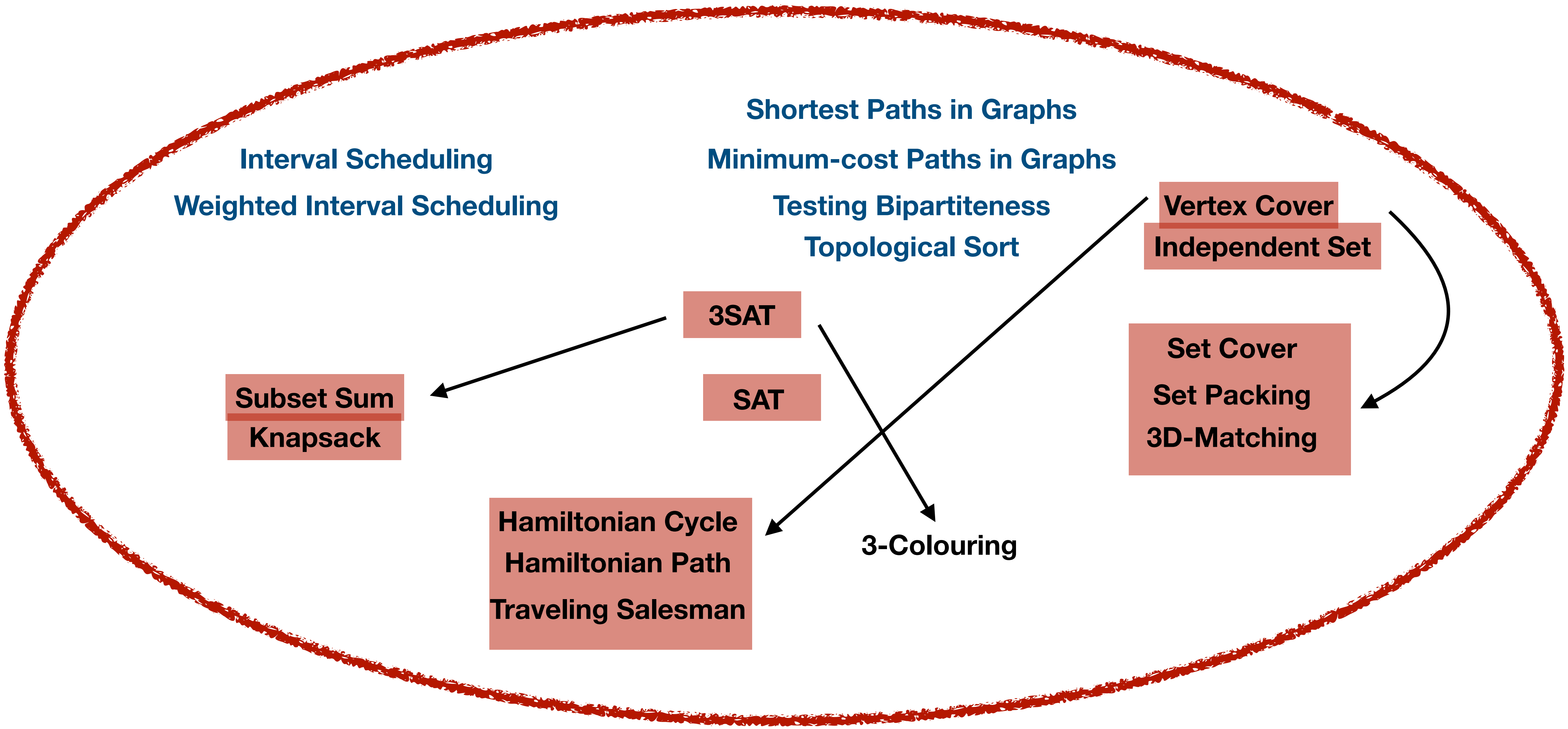
NP-completeness



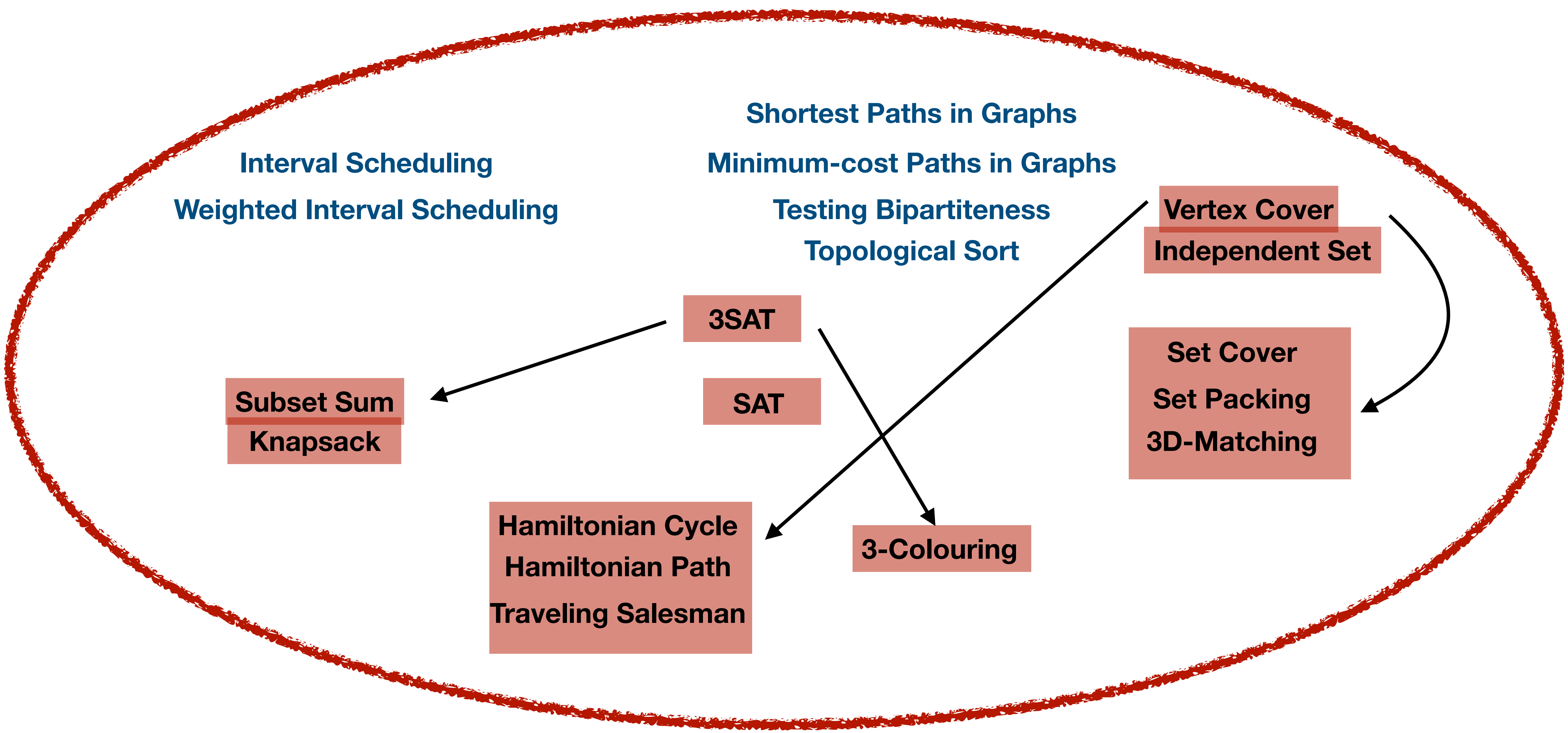
NP-completeness



NP-completeness



NP-completeness



NP-completeness

NP-completeness

- 3SAT, Vertex Cover, Independent Set, Subset Sum, Knapsack, Hamiltonian Path, Hamiltonian Cycle, Traveling Salesman, 3-Colouring, Set Cover, Set Packing, 3D-Matching are all NP-complete.

NP-completeness

- 3SAT, Vertex Cover, Independent Set, Subset Sum, Knapsack, Hamiltonian Path, Hamiltonian Cycle, Traveling Salesman, 3-Colouring, Set Cover, Set Packing, 3D-Matching are all NP-complete.

- Actually, this is only a very small subset of NP-complete problems.

NP-completeness

- 3SAT, Vertex Cover, Independent Set, Subset Sum, Knapsack, Hamiltonian Path, Hamiltonian Cycle, Traveling Salesman, 3-Colouring, Set Cover, Set Packing, 3D-Matching are all NP-complete.

- Actually, this is only a very small subset of NP-complete problems.

    - Hundreds of other meaningful problems are NP-complete.

NP-completeness

- 3SAT, Vertex Cover, Independent Set, Subset Sum, Knapsack, Hamiltonian Path, Hamiltonian Cycle, Traveling Salesman, 3-Colouring, Set Cover, Set Packing, 3D-Matching are all NP-complete.

- Actually, this is only a very small subset of NP-complete problems.

  - Hundreds of other meaningful problems are NP-complete.

- We don't know how to solve any one of those in polynomial-time.

The effect of NP-hardness

The effect of NP-hardness

- Imagine that you have a new favourite problem P.

The effect of NP-hardness

- Imagine that you have a new favourite problem P.
- You try to design a polynomial-time algorithm for it but you find it hard to do so.

The effect of NP-hardness

- Imagine that you have a new favourite problem P.
- You try to design a polynomial-time algorithm for it but you find it hard to do so.
- Then you discover that it can be reduced to one of all of these NP-complete problems.

The effect of NP-hardness

- Imagine that you have a new favourite problem P.
- You try to design a polynomial-time algorithm for it but you find it hard to do so.
- Then you discover that it can be reduced to one of all of these NP-complete problems.
- This means that if you succeeded in your quest, you would solve all of these problems in polynomial-time.

The effect of NP-hardness

- Imagine that you have a new favourite problem P.
- You try to design a polynomial-time algorithm for it but you find it hard to do so.
- Then you discover that it can be reduced to one of all of these NP-complete problems.
- This means that if you succeeded in your quest, you would solve all of these problems in polynomial-time.
- That would mean that you are smarter than generations of researchers and pretty much anyone else that has studied computer science ever.

The effect of NP-hardness

- Imagine that you have a new favourite problem P.
- You try to design a polynomial-time algorithm for it but you find it hard to do so.
- Then you discover that it can be reduced to one of all of these NP-complete problems.
- This means that if you succeeded in your quest, you would solve all of these problems in polynomial-time.
- That would mean that you are smarter than generations of researchers and pretty much anyone else that has studied computer science ever.
- I don't know about you, but I would probably be convinced that I am not going to come up with a polynomial-time algorithm!

Wooclap!

Reduction strategies

Reduction strategies

- For now, we'll tell you what to reduce from.

Reduction strategies

- For now, we'll tell you what to reduce from.
- And the reduction will be relatively simple.

Reduction strategies

- For now, we'll tell you what to reduce from.
- And the reduction will be relatively simple.
- In general, the idea is to find a problem that looks similar to the one we are trying to prove NP-hardness for.

Reduction strategies

- For now, we'll tell you what to reduce from.
- And the reduction will be relatively simple.
- In general, the idea is to find a problem that looks similar to the one we are trying to prove NP-hardness for.
- Try to think of reductions you have seen in the past.

Reduction strategies

- For now, we'll tell you what to reduce from.
- And the reduction will be relatively simple.
- In general, the idea is to find a problem that looks similar to the one we are trying to prove NP-hardness for.
- Try to think of reductions you have seen in the past.
  - This takes time!

NP-completeness, a taxonomy

Packing problems

Independent Set
Set Packing

Covering problems

Vertex Cover
Set Cover

Partitioning problems

3D-Matching
Graph Colouring

Sequencing problems

Hamiltonian Cycle
Hamiltonian Path
Traveling Salesman

Numerical problems

Subset Sum
Knapsack

Constraint Satisfaction problems

3 SAT

NP-completeness

NP-completeness

- So when a problem is NP-complete, this means:

NP-completeness

- So when a problem is NP-complete, this means:
  - That it is in NP, and it is at least as hard to solve as any other problem in NP.

NP-completeness

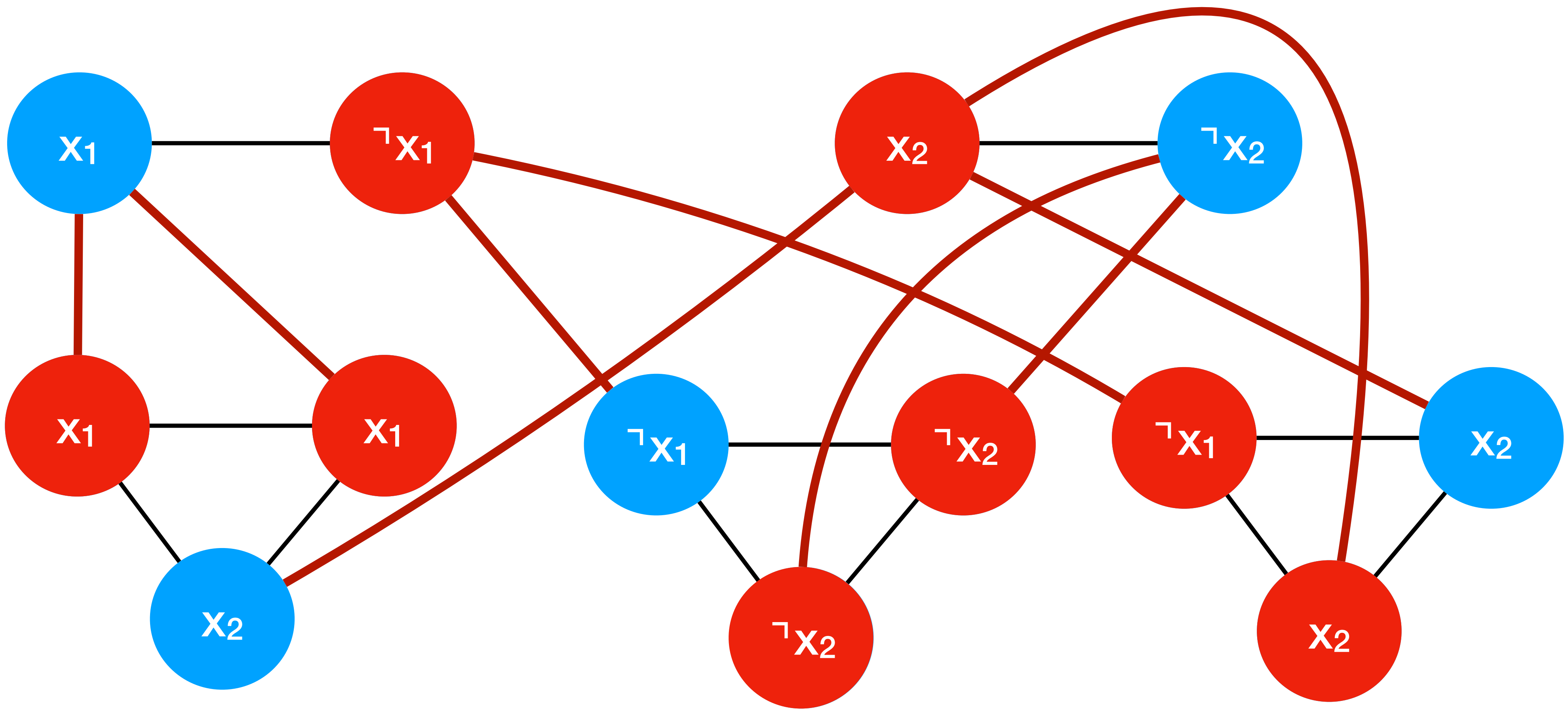
- So when a problem is NP-complete, this means:
  - That it is in NP, and it is at least as hard to solve as any other problem in NP.
  - That it is unlikely that we solve it in polynomial time, as that would imply that we solve all the NP-complete problems.

NP-completeness

- So when a problem is NP-complete, this means:
  - That it is in NP, and it is at least as hard to solve as any other problem in NP.
  - That it is unlikely that we solve it in polynomial time, as that would imply that we solve all the NP-complete problems.
  - That it is not solvable in polynomial time assuming $P \neq NP$.

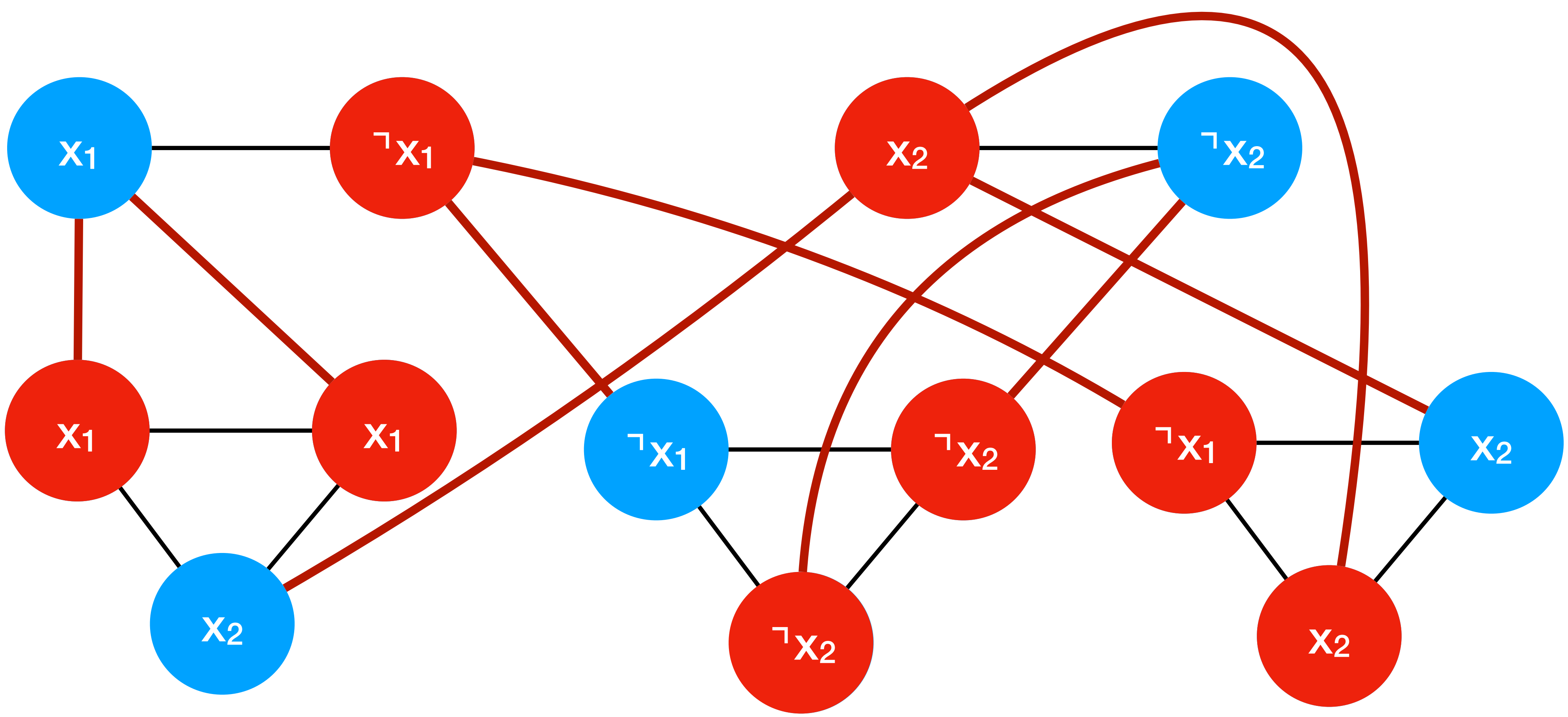
Wooclap!

NP-hardness is a worst-case impossibility



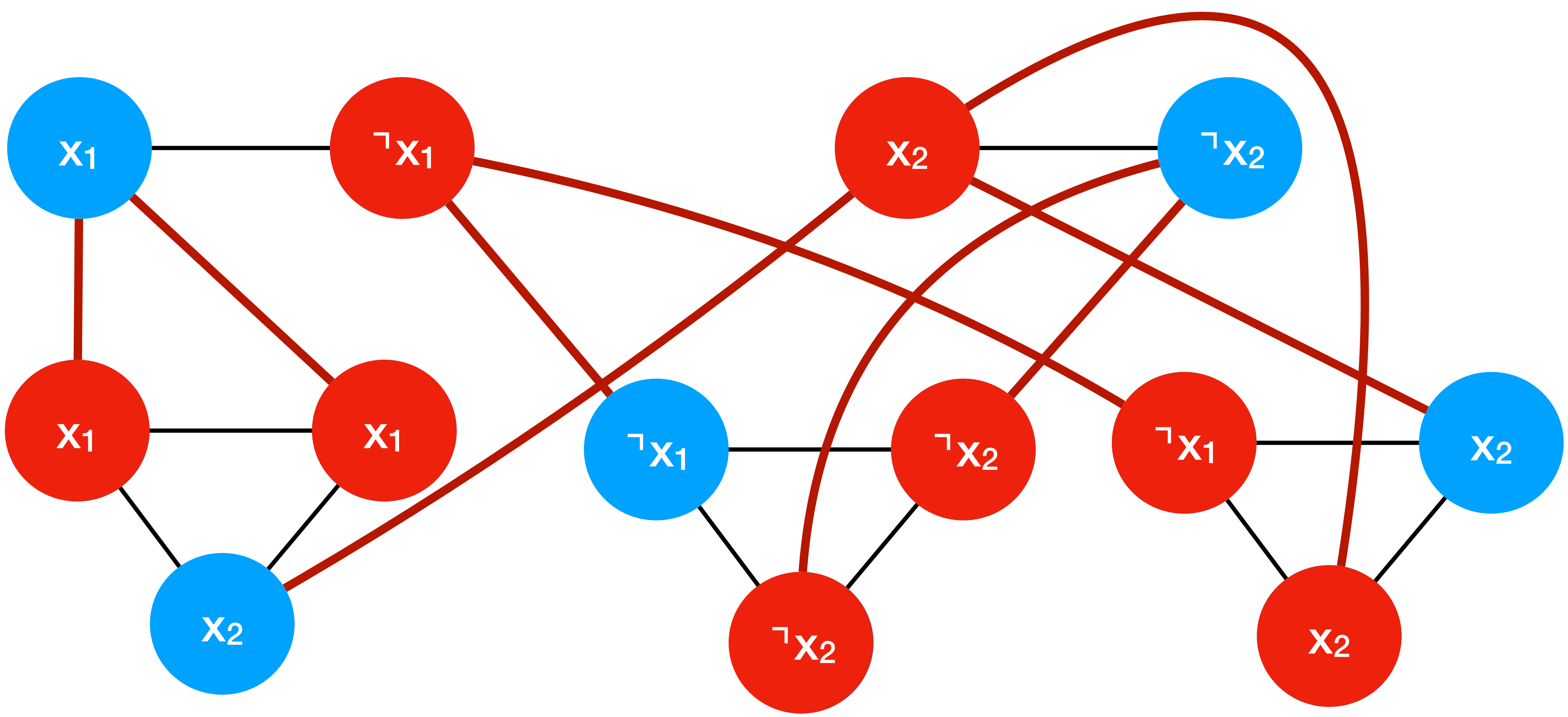
NP-hardness is a worst-case impossibility

- Let's recall the NP-hardness proof for Vertex Cover.



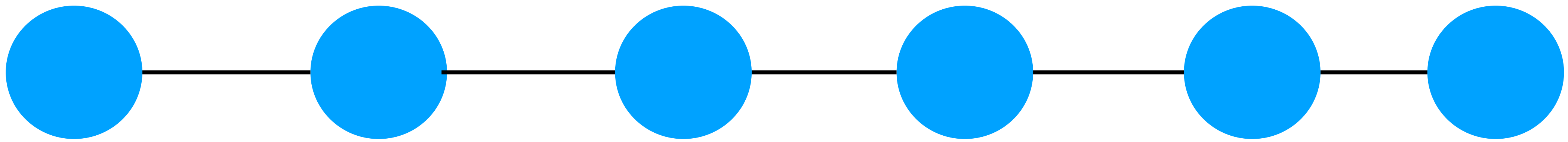
NP-hardness is a worst-case impossibility

- Let's recall the NP-hardness proof for Vertex Cover.



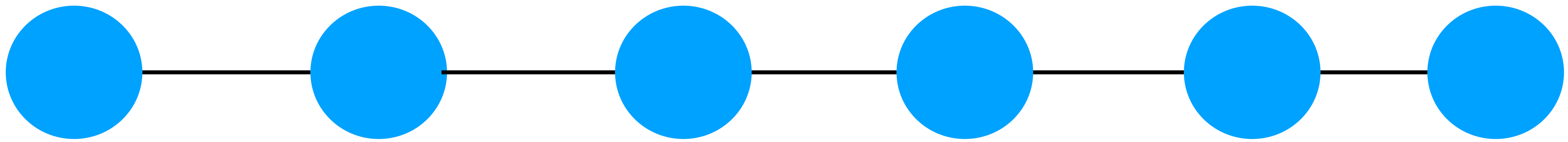
- If I could decide Vertex Cover on this graph, I could decide 3SAT.

NP-hardness is a worst-case impossibility



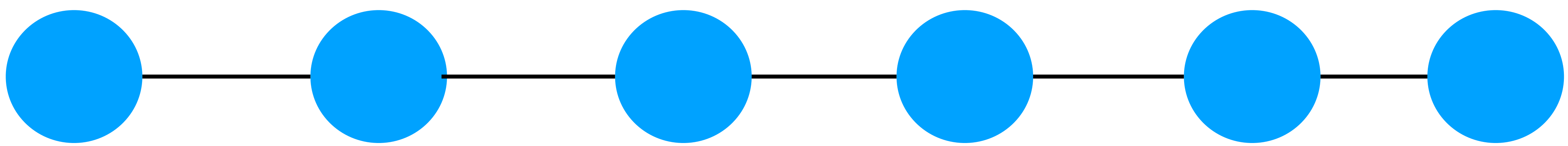
NP-hardness is a worst-case impossibility

- What about this graph? Can I decide Vertex Cover on this graph?



NP-hardness is a worst-case impossibility

- What about this graph? Can I decide Vertex Cover on this graph?



- “Choose one leave one” finds a minimum vertex cover.

NP-hardness is a worst-case impossibility

NP-hardness is a worst-case impossibility

- An NP-hardness result does not mean that (unless P=NP) we cannot solve the problem in polynomial time *for any* instance.

NP-hardness is a worst-case impossibility

- An NP-hardness result does not mean that (unless P=NP) we cannot solve the problem in polynomial time *for any* instance.

- It means that we cannot solve it in polynomial time *for every* instance.

NP-hardness is a worst-case impossibility

- An NP-hardness result does not mean that (unless P=NP) we cannot solve the problem in polynomial time *for any* instance.
- It means that we cannot solve it in polynomial time *for every* instance.
- For all we know, every other instance besides those used in the reduction could be easy to solve.

NP-hardness is a worst-case impossibility

- For all we know, every other instance besides those used in the reduction could be easy to solve.

NP-hardness is a worst-case impossibility

- For all we know, every other instance besides those used in the reduction could be easy to solve.
- Usually not the case! In practice usually we don’t have good ways of solving NP-hard problems.

NP-hardness is a worst-case impossibility

- For all we know, every other instance besides those used in the reduction could be easy to solve.

- Usually not the case! In practice usually we don't have good ways of solving NP-hard problems.

- Still, sometimes we can provably design polynomial algorithms on certain *input structures*.

NP-hardness is a worst-case impossibility

- For all we know, every other instance besides those used in the reduction could be easy to solve.
- Usually not the case! In practice usually we don’t have good ways of solving NP-hard problems.
- Still, sometimes we can provably design polynomial algorithms on certain *input structures*.
- For example, a minimum Vertex Cover on *trees* can be found in polynomial time using Dynamic Programming.

Wooclap!

NP-hardness ≠ Exponential Time

NP-hardness $\neq$ Exponential Time

- This is true even if $P \neq NP$.

NP-hardness $\neq$ Exponential Time

- This is true even if $P \neq NP$.
- Roughgarden: *Acceptable Inaccuracy #3.*

NP-hardness $\neq$ Exponential Time

- This is true even if $P \neq NP$.
- Roughgarden: *Acceptable Inaccuracy #3.*
- Subexponential time: $n^{O(\lg n)}$ or $2^{O(\sqrt{n})}$.

NP-hardness $\neq$ Exponential Time

- This is true even if $P \neq NP$.
- Roughgarden: *Acceptable Inaccuracy #3.*
- Subexponential time: $n^{O(\lg n)}$ or $2^{O(\sqrt{n})}$.
- There are NP-complete problems that can be solved in subexponential time.

NP-hardness $\neq$ Exponential Time

- This is true even if $P \neq NP$.

- Roughgarden: *Acceptable Inaccuracy #3.*

- Subexponential time: $n^{O(\lg n)}$ or $2^{O(\sqrt{n})}$.

- There are NP-complete problems that can be solved in subexponential time.

- Exponential Time Hypothesis (ETH): SAT requires exponential time to be solved.

NP-hardness $\neq$ Exponential Time

- This is true even if $P \neq NP$.
- Roughgarden: *Acceptable Inaccuracy #3.*
- Subexponential time: $n^{O(\lg n)}$ or $2^{O(\sqrt{n})}$.
- There are NP-complete problems that can be solved in subexponential time.
- Exponential Time Hypothesis (ETH): SAT requires exponential time to be solved.
  - ETH $\Rightarrow P \neq NP$

NP-hardness vs NP-completeness

NP-hardness vs NP-completeness

- Every NP-complete problem is NP-hard.

NP-hardness vs NP-completeness

- Every NP-complete problem is NP-hard.
- Is every NP-hard problem NP-complete?

NP-hardness vs NP-completeness

- Every NP-complete problem is NP-hard.
- Is every NP-hard problem NP-complete?
- Are there problems that are NP-hard but not in NP?

Totally Quantified Boolean Formula (TQBF)

Totally Quantified Boolean Formula (TQBF)

- A CNF formula with m clauses and k literals, and a set of quantifiers.

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \phi(x_1, x_2, \ldots, x_n)$$

where $Q_i \in \{\forall, \exists\}$

Totally Quantified Boolean Formula (TQBF)

- A CNF formula with m clauses and k literals, and a set of quantifiers.

  $$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \phi(x_1, x_2, \ldots, x_n)$$

  where $Q_i \in \{\forall, \exists\}$

- For example, we may have
  $\exists x_1 \forall x_2 \exists x_3, \ldots, \forall x_n \phi(x_1, x_2, x_3, \ldots, x_n)$

Totally Quantified Boolean Formula (TQBF)

- A CNF formula with m clauses and k literals, and a set of quantifiers.

  $Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \phi(x_1, x_2, \ldots, x_n)$

  where $Q_i \in \{\forall, \exists\}$

- For example, we may have
  $\exists x_1 \forall x_2 \exists x_3, \ldots, \forall x_n \phi(x_1, x_2, x_3, \ldots, x_n)$

- We read “Does there exists x_1 such that for every x_2 there exists x_3 such that … such that for every x_n, the formula ϕ is satisfiable?”

Or maybe a game of chess

Or maybe a game of chess

- Does white have a winning strategy?

Or maybe a game of chess

- Does white have a winning strategy?

- Does there exist a move for white, such that for every move of black, there exists a move for white, such that for every move of black, … , such that for every move of black, white wins?

Totally Quantified Boolean Formula (TQBF)

- A CNF formula with m clauses and k literals, and a set of quantifiers.

  $Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \phi(x_1, x_2, \ldots, x_n)$

  where $Q_i \in \{\forall, \exists\}$

- For example, we may have
  $\exists x_1 \forall x_2 \exists x_3, \ldots, \forall x_n \phi(x_1, x_2, x_3, \ldots, x_n)$

Totally Quantified Boolean Formula (TQBF)

- A CNF formula with m clauses and k literals, and a set of quantifiers.

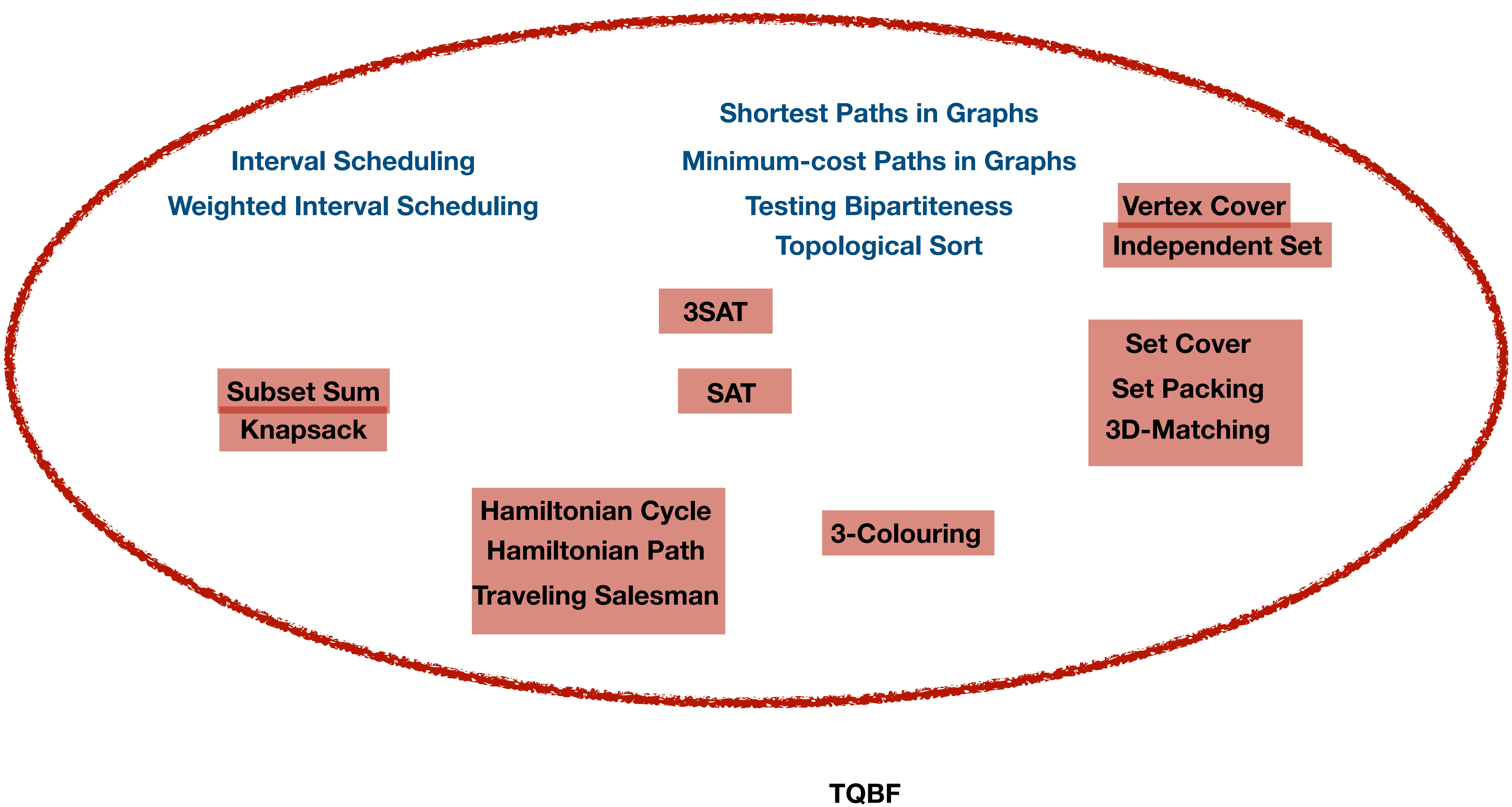
  $$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \phi(x_1, x_2, \ldots, x_n)$$

  where $Q_i \in \{\forall, \exists\}$

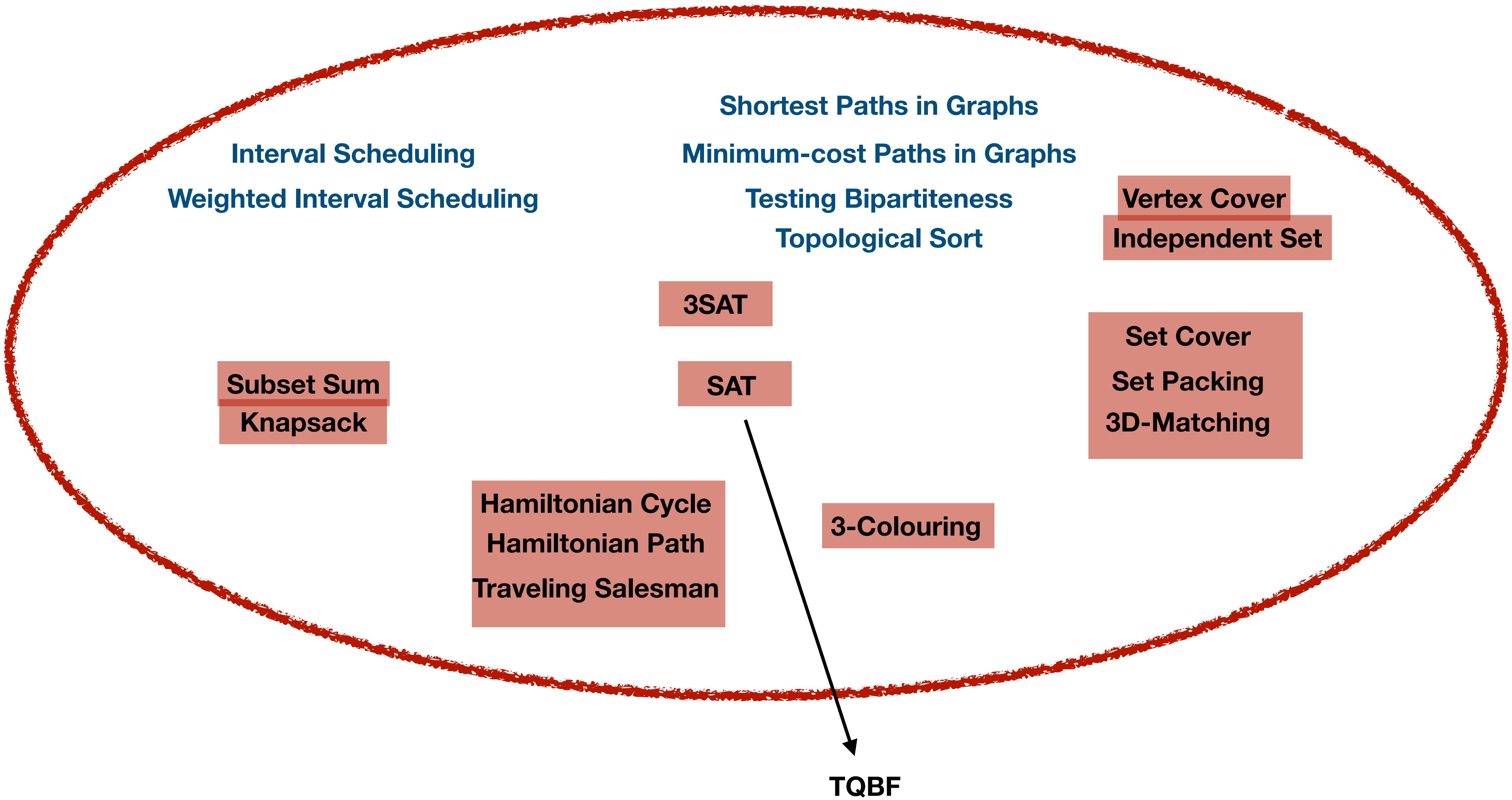
- For example, we may have
  $\exists x_1 \forall x_2 \exists x_3, \ldots, \forall x_n \phi(x_1, x_2, x_3, \ldots, x_n)$

- TQBF is NP-hard. Why?

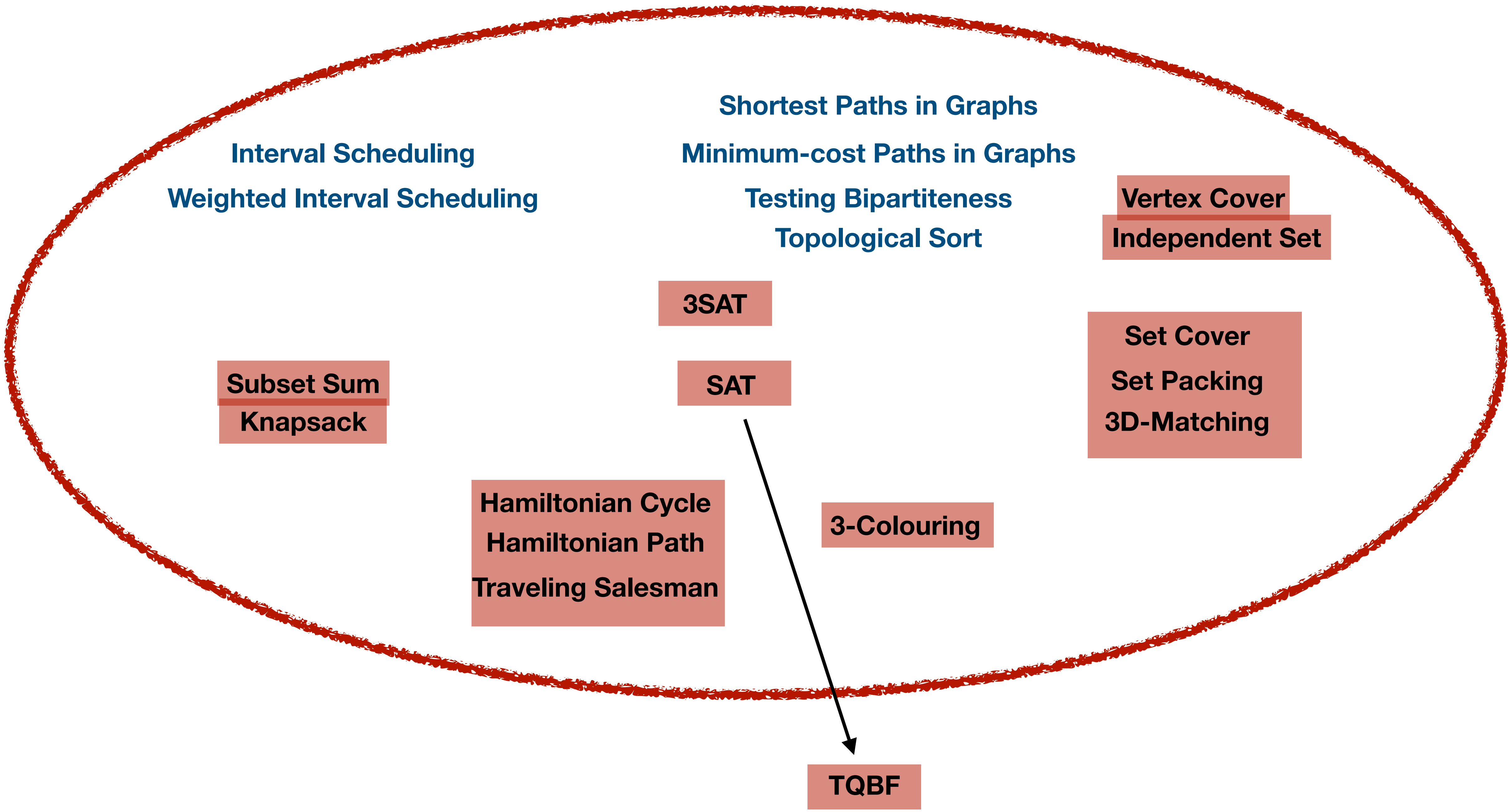
NP-completeness



NP-completeness



NP-completeness



Totally Quantified Boolean Formula (TQBF)

- A CNF formula with m clauses and k literals, and a set of quantifiers.

  $Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \phi(x_1, x_2, \ldots, x_n)$

  where $Q_i \in \{\forall, \exists\}$

- For example, we may have $\exists x_1 \forall x_2 \exists x_3, \ldots, \forall x_n \phi(x_1, x_2, x_3, \ldots, x_n)$

- We read "Does there exists x_1 such that for every x_2 there exists x_3 such that … such that for every x_n, the formula ϕ is satisfiable?"

Totally Quantified Boolean Formula (TQBF)

- A CNF formula with m clauses and k literals, and a set of quantifiers.

  $Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \phi(x_1, x_2, \ldots, x_n)$

  where $Q_i \in \{\forall, \exists\}$

- For example, we may have $\exists x_1 \forall x_2 \exists x_3, \ldots, \forall x_n \phi(x_1, x_2, x_3, \ldots, x_n)$

- We read "Does there exists x_1 such that for every x_2 there exists x_3 such that … such that for every x_n, the formula ϕ is satisfiable?"

- For a given x_1 we have to check the values of x_2 for all possible values of the remaining x_4, x_6, etc. Does not seem to be doable in polynomial time.

An interesting but important note

An interesting but important note

- We cannot categorically say that TQBF is not efficiently verifiable (i.e., that it is not in NP).

An interesting but important note

- We cannot categorically say that TQBF is not efficiently verifiable (i.e., that it is not in NP).

- In the same way as we cannot categorically say that SAT is not efficiently solvable (i.e., that it is not in P).

An interesting but important note

- We cannot categorically say that TQBF is not efficiently verifiable (i.e., that it is not in NP).
- In the same way as we cannot categorically say that SAT is not efficiently solvable (i.e., that it is not in P).
- But we have reasons to believe that SAT is not in P.

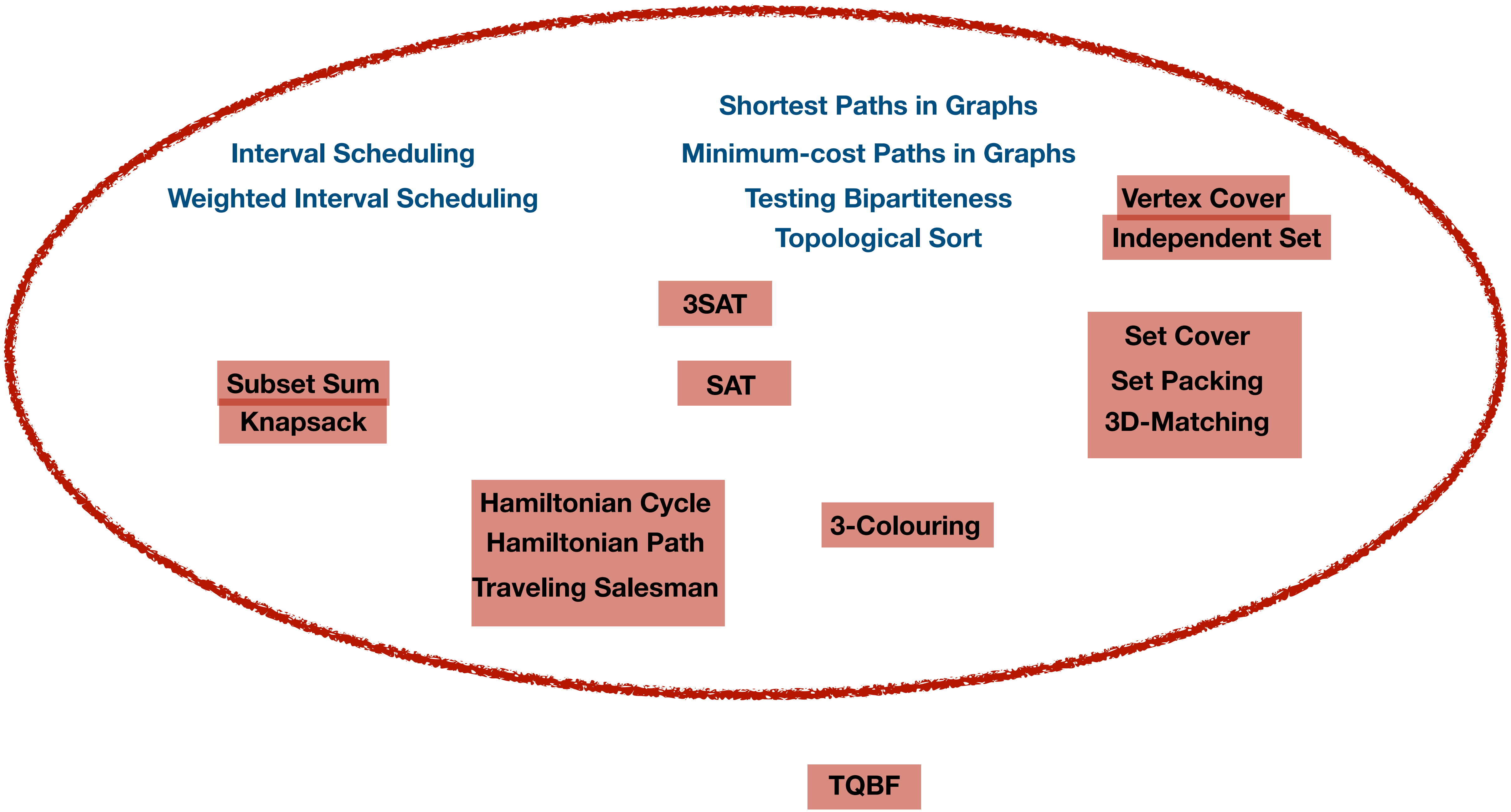
An interesting but important note

- We cannot categorically say that TQBF is not efficiently verifiable (i.e., that it is not in NP).
- In the same way as we cannot categorically say that SAT is not efficiently solvable (i.e., that it is not in P).
- But we have reasons to believe that SAT is not in P.
  - Because then we would be able to solve all NP-complete problems.

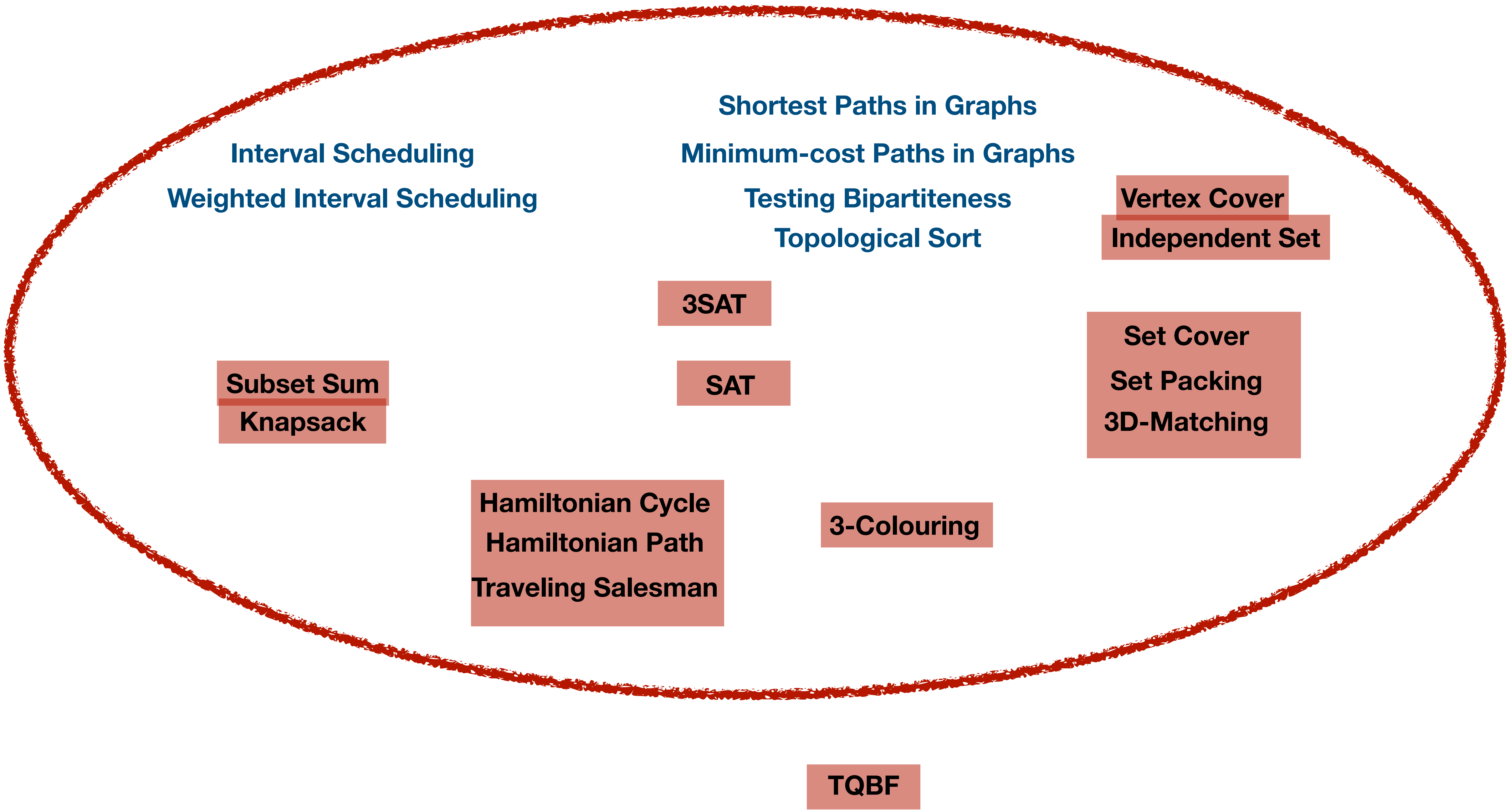
An interesting but important note

- We cannot categorically say that TQBF is not efficiently verifiable (i.e., that it is not in NP).
- In the same way as we cannot categorically say that SAT is not efficiently solvable (i.e., that it is not in P).
- But we have reasons to believe that SAT is not in P.
  - Because then we would be able to solve all NP-complete problems.
- Similarly we have reasons to believe that TQBF is not in NP.

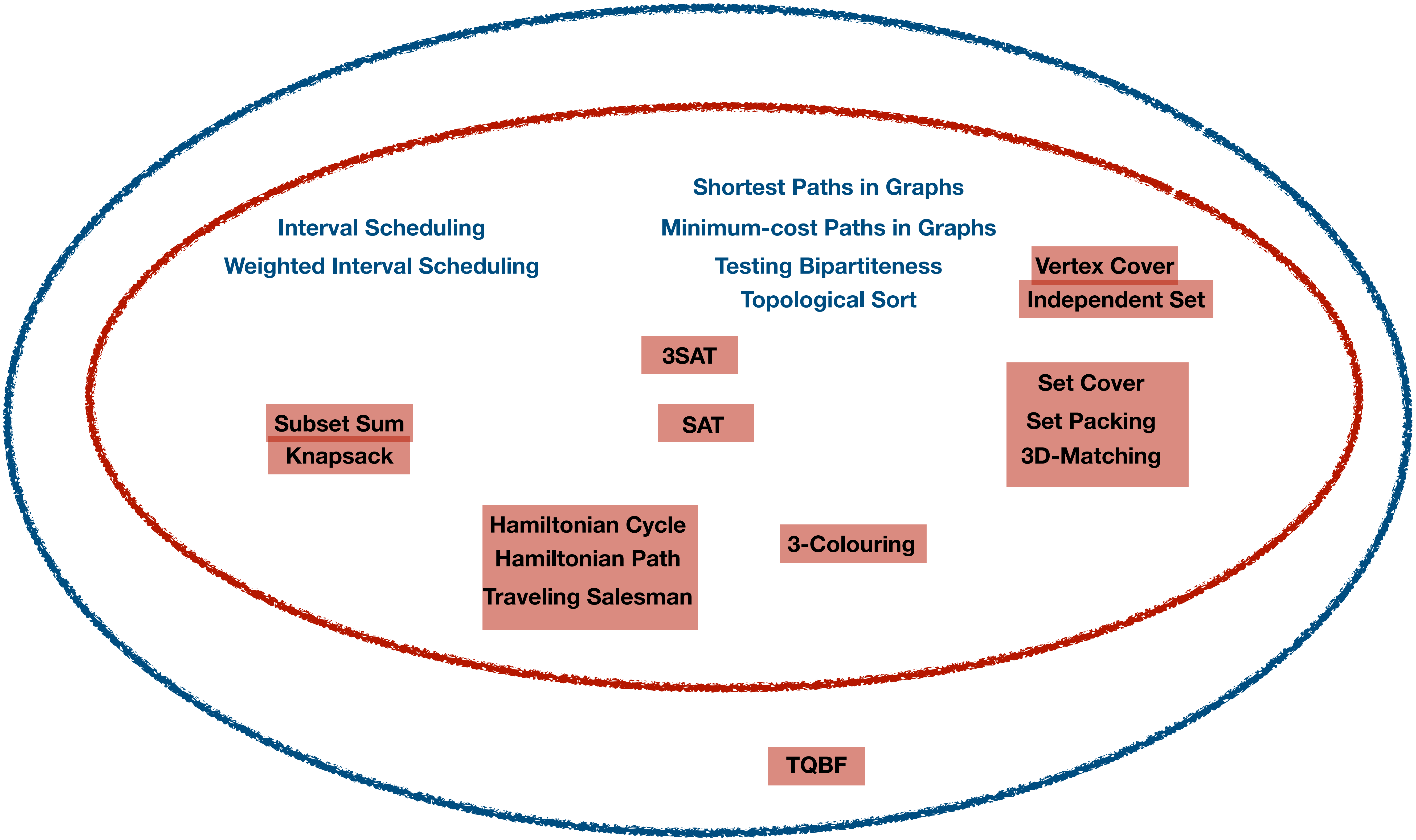
NP


Shortest Paths in Graphs
Interval Scheduling
Minimum-cost Paths in Graphs
Weighted Interval Scheduling
Testing Bipartiteness
Vertex Cover
Topological Sort
Independent Set
3SAT
Set Cover
Subset Sum
SAT
Set Packing
Knapsack
3D-Matching
Hamiltonian Cycle
Hamiltonian Path
Traveling Salesman
3-Colouring
TQBF

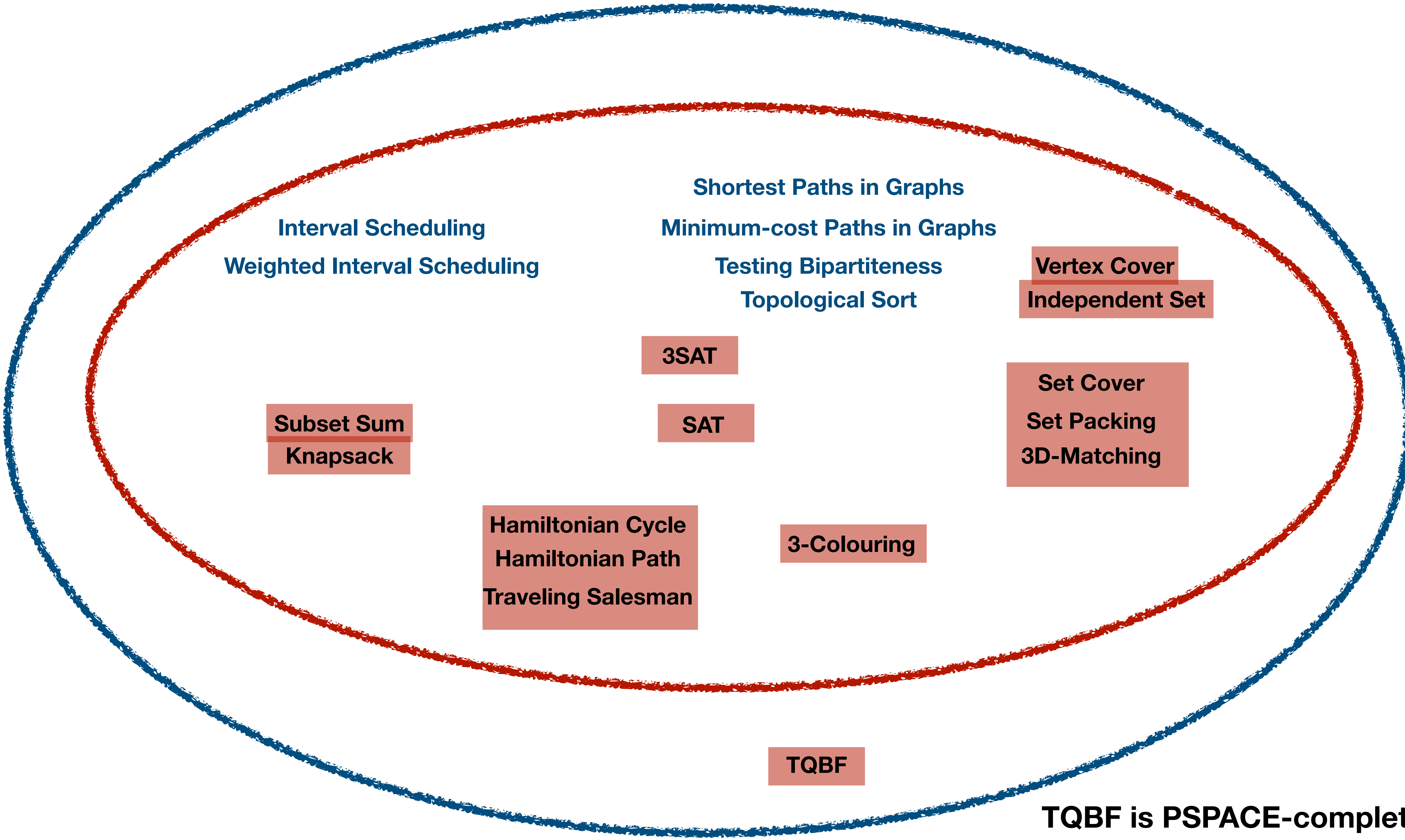
PSPACE


Shortest Paths in Graphs
Interval Scheduling
Minimum-cost Paths in Graphs
Weighted Interval Scheduling
Testing Bipartiteness
Vertex Cover
Topological Sort
Independent Set
3SAT
Set Cover
Subset Sum
SAT
Set Packing
Knapsack
3D-Matching
Hamiltonian Cycle
3-Colouring
Hamiltonian Path
Traveling Salesman
TQBF

PSPACE


Shortest Paths in Graphs
Interval Scheduling
Minimum-cost Paths in Graphs
Weighted Interval Scheduling
Testing Bipartiteness
Vertex Cover
Topological Sort
Independent Set
3SAT
Set Cover
Subset Sum
SAT
Set Packing
Knapsack
3D-Matching
Hamiltonian Cycle
Hamiltonian Path
Traveling Salesman
3-Colouring
TQBF

PSPACE



TQBF is PSPACE-complete

Another NP-hard problem that is not in NP

Another NP-hard problem that is not in NP

- Informally: Given the description of an arbitrary computer program and an input to the program, determine if the program will terminate or not.

Another NP-hard problem that is not in NP

- Informally: Given the description of an arbitrary computer program and an input to the program, determine if the program will terminate or not.

- This is the Halting Problem, which is NP-hard but it is undecidable.

Another NP-hard problem that is not in NP

- Informally: Given the description of an arbitrary computer program and an input to the program, determine if the program will terminate or not.

- This is the Halting Problem, which is NP-hard but it is undecidable.

    - i.e., it cannot be solved in any amount of time on any computer.

Another NP-hard problem that is not in NP

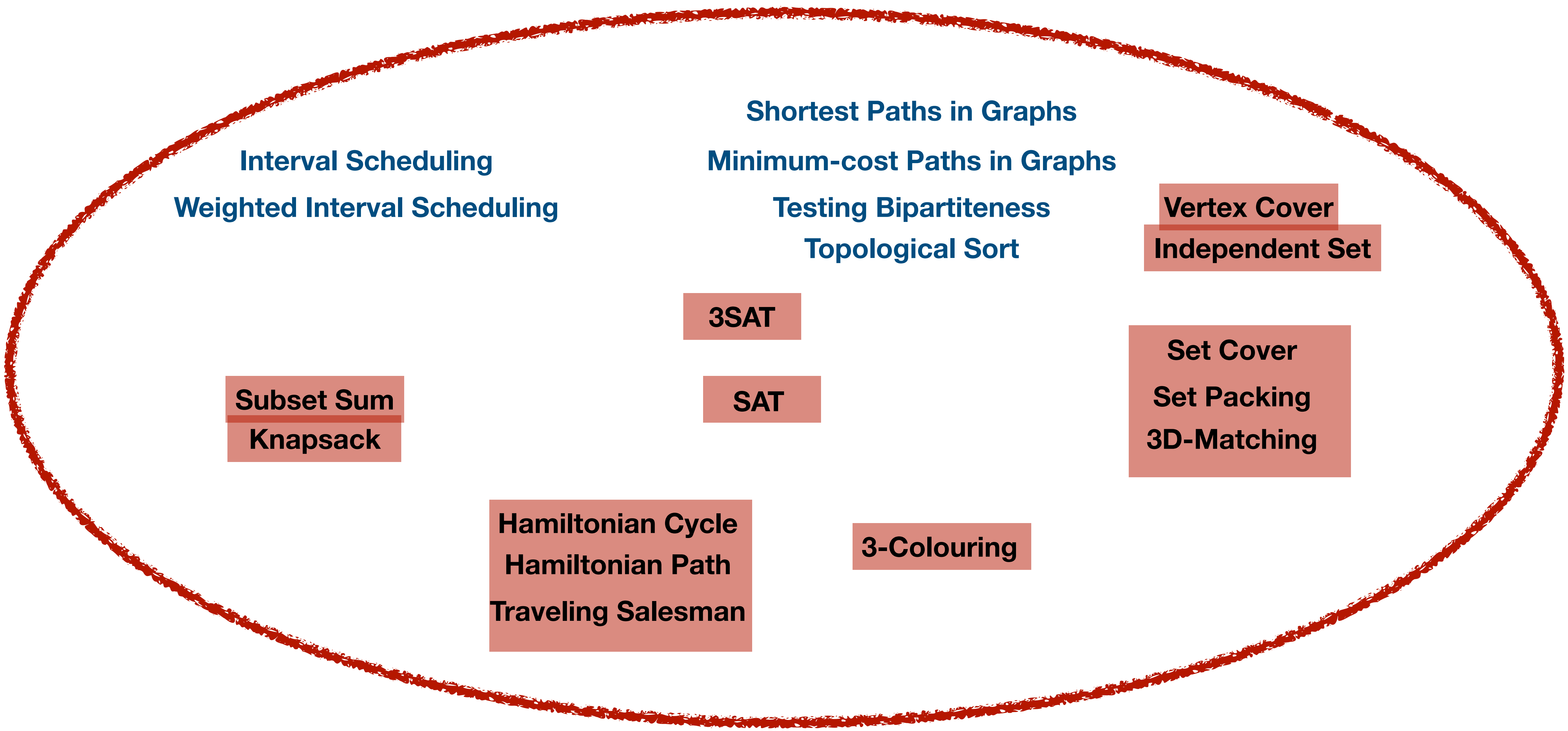
- Informally: Given the description of an arbitrary computer program and an input to the program, determine if the program will terminate or not.

- This is the Halting Problem, which is NP-hard but it is undecidable.

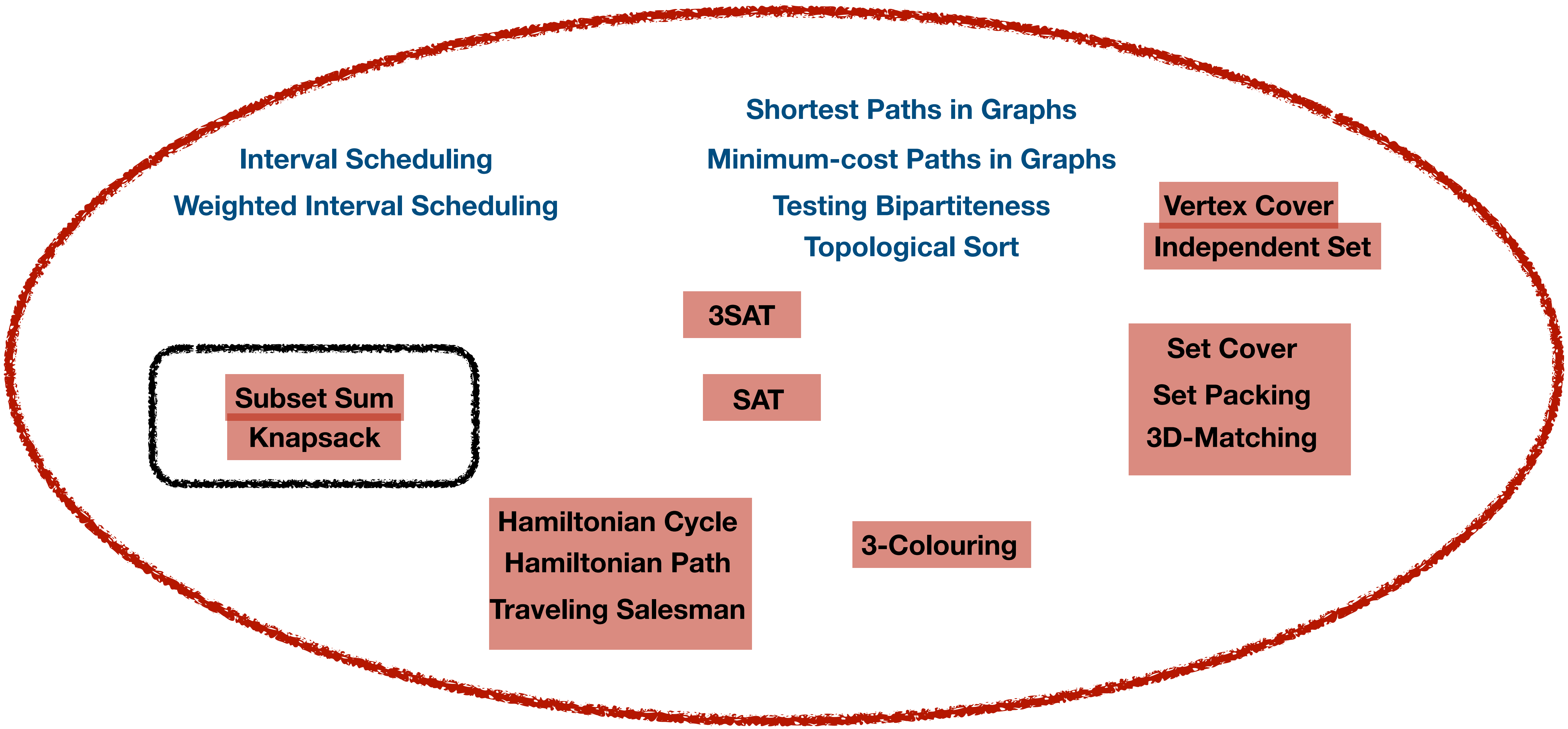
  - i.e., it cannot be solved in any amount of time on any computer.

  - More about that later!

Are all NP-complete problems equally hard?

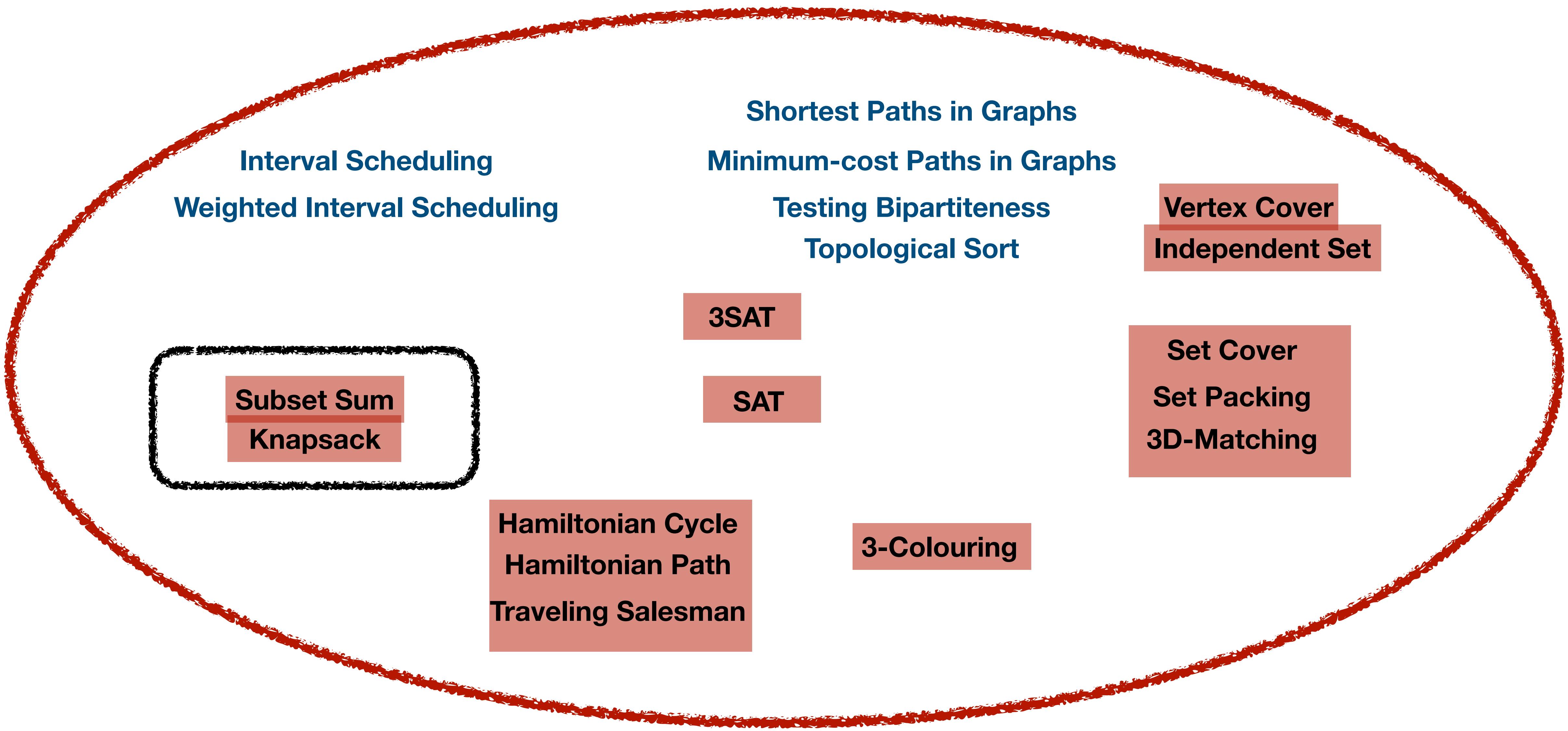


Are all NP-complete problems equally hard?



We solved those in pseudopolynomial time.

Are all NP-complete problems equally hard?



We solved those in pseudopolynomial time.

Could they be "easier" than SAT in some sense?

Strong vs Weak NP-hardness

Strong vs Weak NP-hardness

- A problem P is strongly NP-hard if it remains NP-hard even when the numerical parameters in the input are given in unary representation.

Strong vs Weak NP-hardness

- A problem P is strongly NP-hard if it remains NP-hard even when the numerical parameters in the input are given in unary representation.

- Otherwise, it is weakly NP-hard.

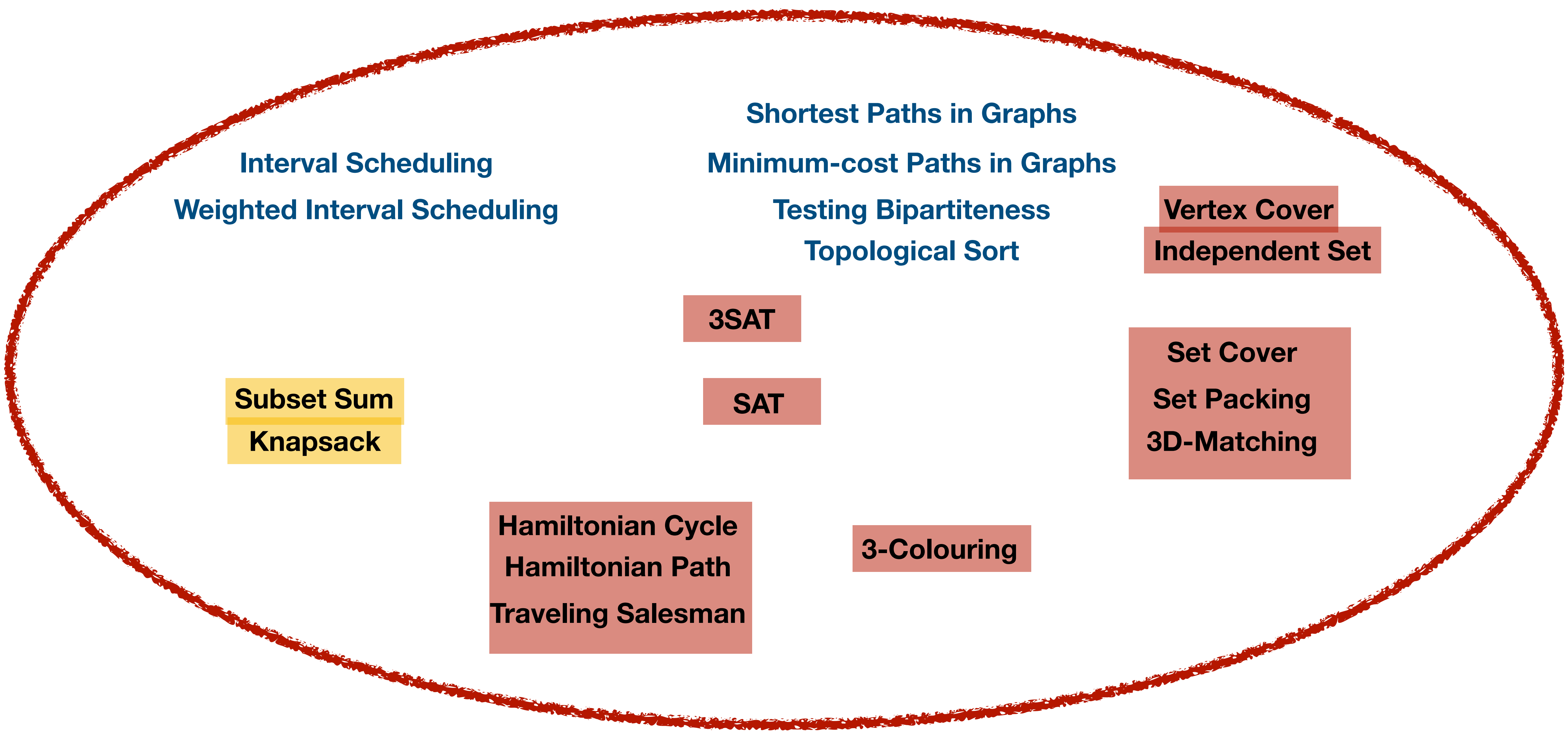
Strong vs Weak NP-hardness

- A problem P is strongly NP-hard if it remains NP-hard even when the numerical parameters in the input are given in unary representation.

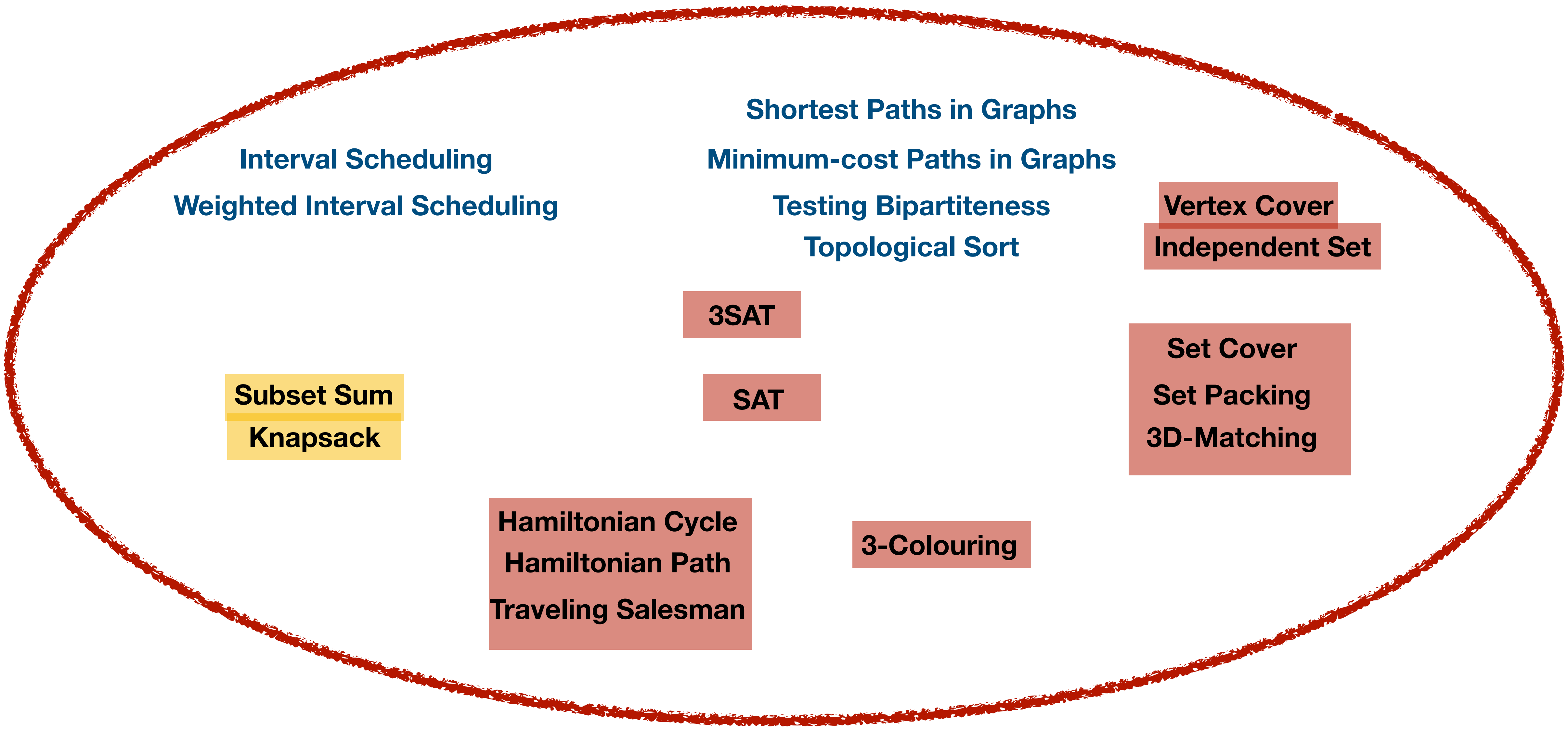
- Otherwise, it is weakly NP-hard.

    - Weakly NP-hard problems admit pseudopolynomial algorithms.

Are all NP-complete problems equally hard?

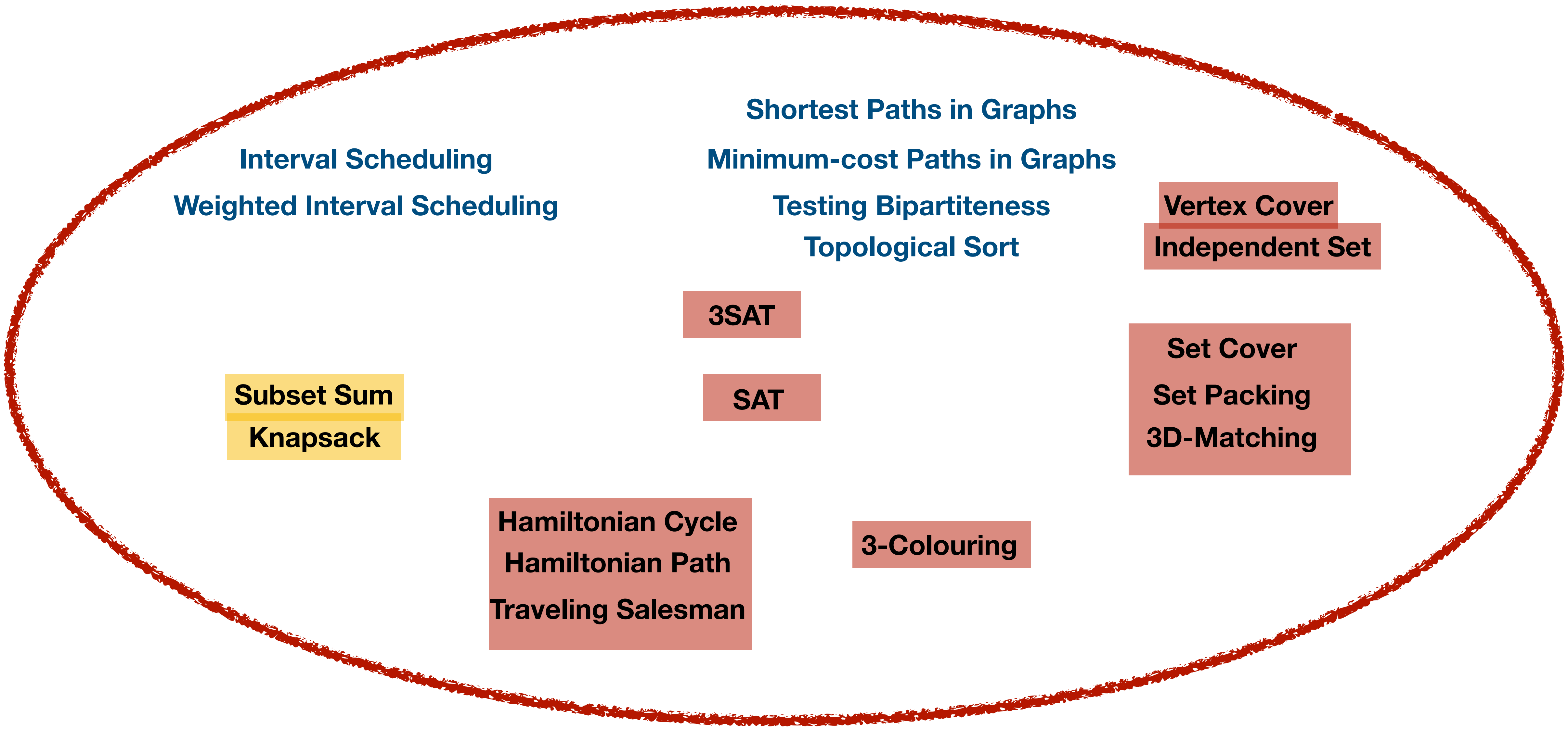


Are all NP-complete problems equally hard?



Another way to compare: Approximate Solutions

Are all NP-complete problems equally hard?



Another way to compare: Approximate Solutions

More about that over the next two lectures!