Journée graphes – October 1st, 2015 – Rouen

Graph matching for biological databases and social network data

Pasquale Foggia <pfoggia@unisa.it>

DIEM
University of Salerno – Italy

Machine Intelligence lab for Video, Image and Audio processing

Outline

- Motivation of this work
- The VF2 algorithm
- Improvements to the VF2 algorithms
- Experimentation on biological graphs
- Experimentation on social network graphs
- Conclusions

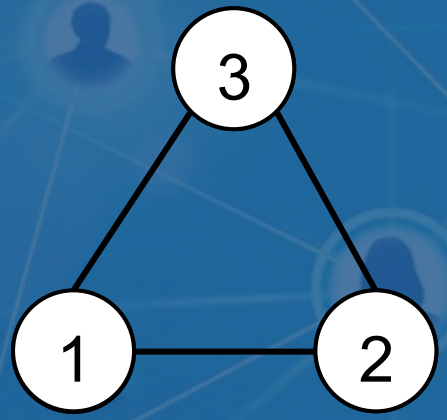
Motivation of this work

- VF2 is a general-purpose graph isomorphism and subgraph isomorphism algorithm [Cordella et al. 2004]
- At the time of its publication, one of the fastest algorithms
- But, on graphs from biological databases, several newer algorithm are able to outperform it

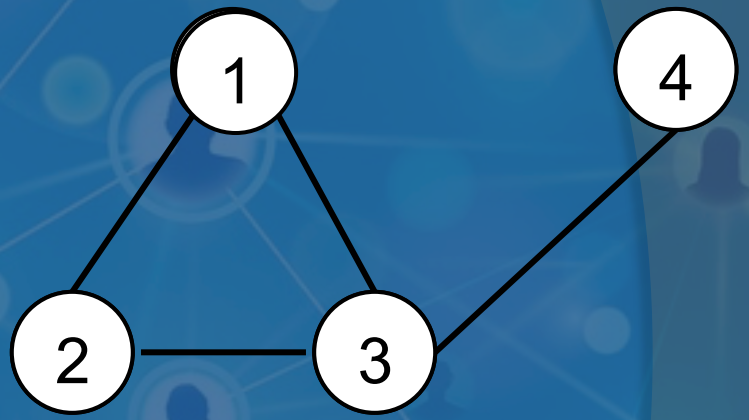
Motivation of this work

- We have devised an improved version of the algorithm, first presented at GbR2015, aimed at biological databases
- We are also testing this improved algorithm on graph models that are typically found in Social Network Analysis

(Node induced) Subgraph isomorphism

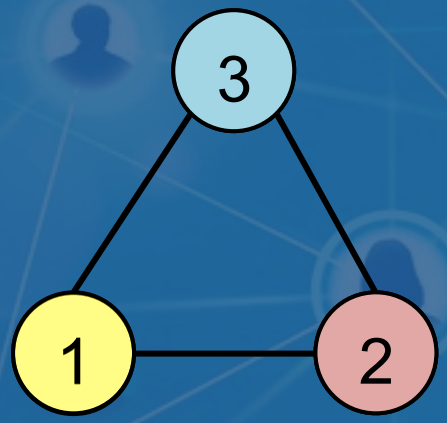


Pattern graph

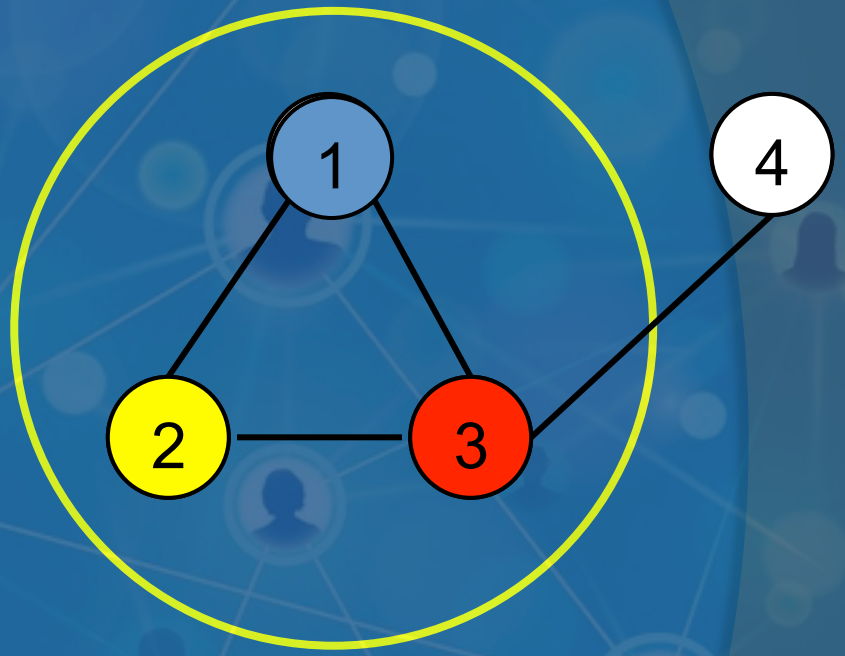


Target graph

(Node induced) Subgraph isomorphism

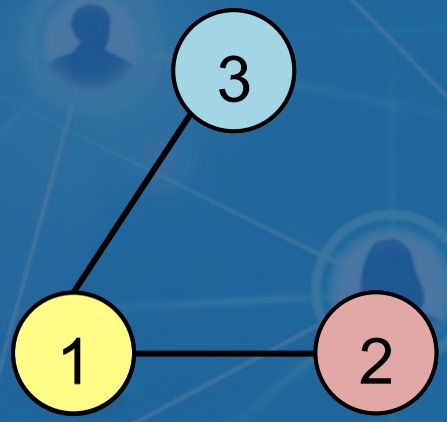


Pattern graph

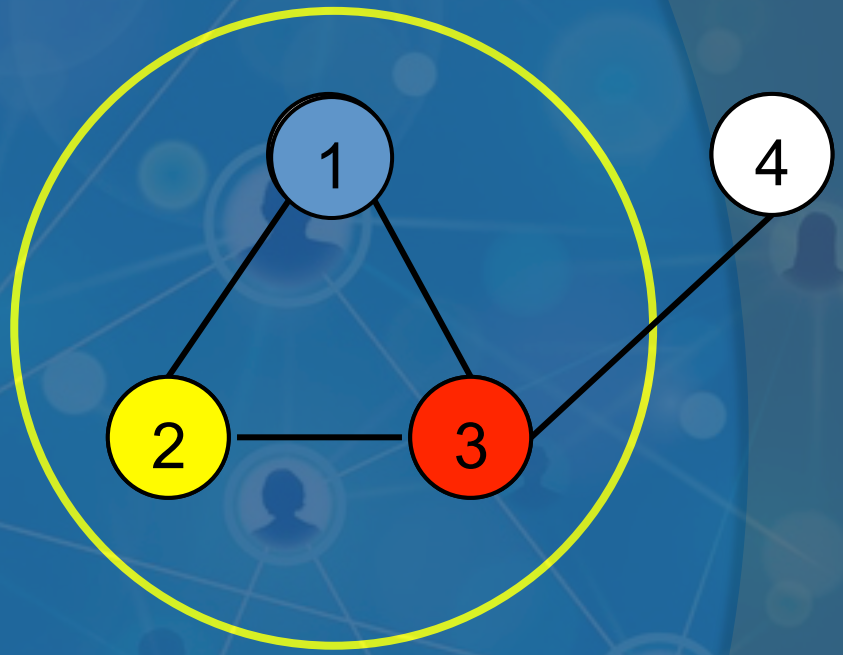


Target graph

(Edge induced) Subgraph isomorphism (aka Monomorphism)



Pattern graph



Target graph

(the target graph may contain extra edges between mapped nodes)

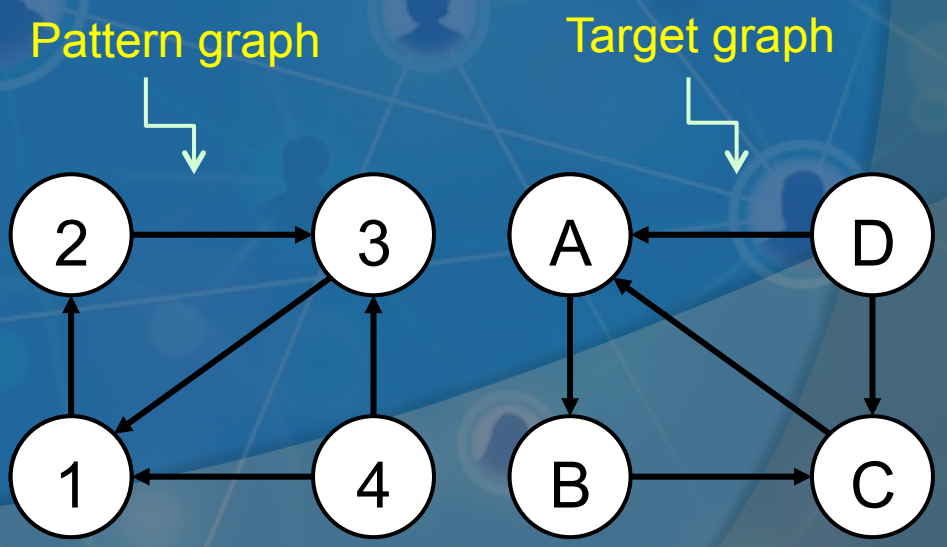
The VF2 algorithm

- The problem is formulated in terms of State Space Representation, where each state represents a partial mapping solution.
- A generic matching algorithm based on SSR can be represented as the search over a state graph, where the edges are the addition of a pair of nodes to an existing partial solution
- If only *consistent* partial solutions are generated, then each state covering the whole pattern is a valid solution to the problem

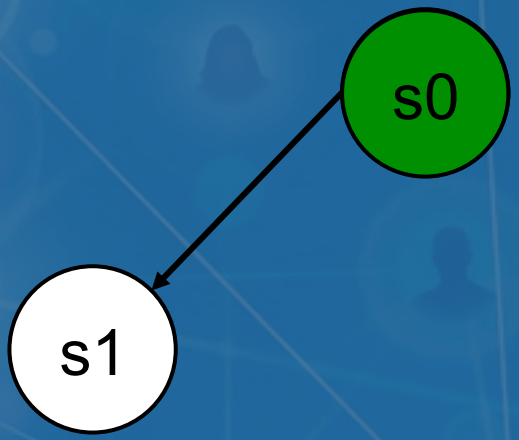
State Space Representation

M(s0) = {}

Partial mapping of the initial state

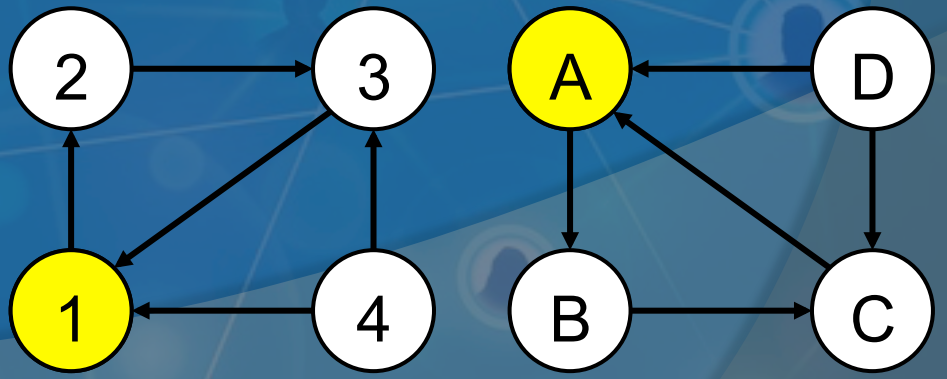


State Space Representation

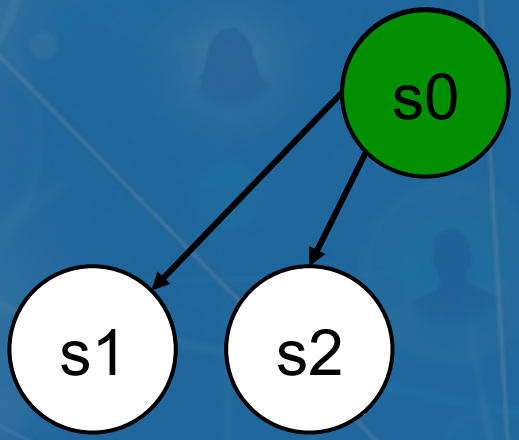


M(s0) = {}

M(s1) = {(1,A)}



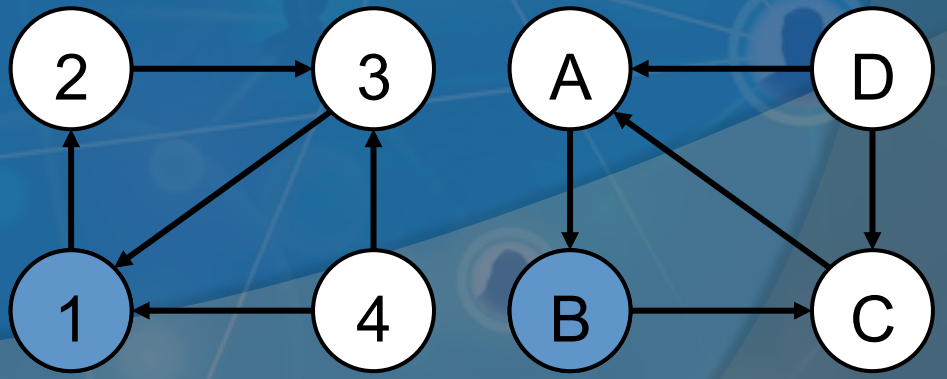
State Space Representation



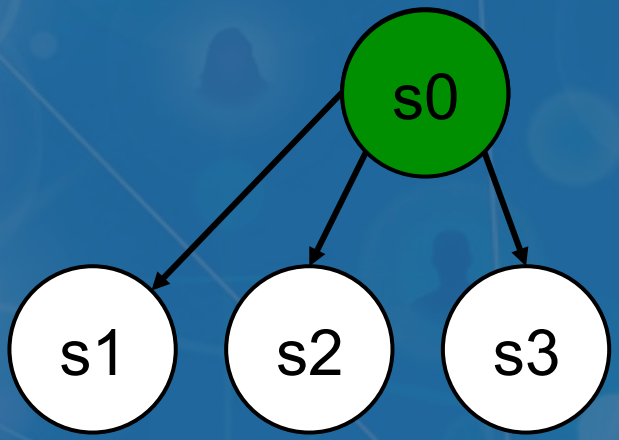
M(s0) = {0}

M(s1) = {(1,A)}

M(s2) = {(1,B)}



State Space Representation

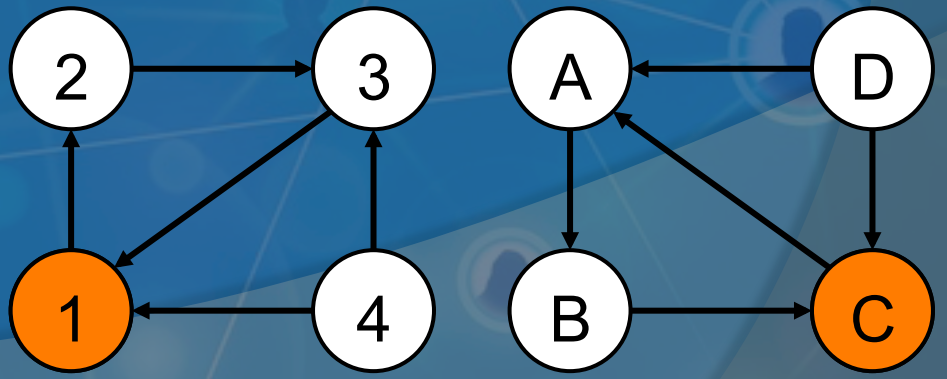


M(s0) = {0}

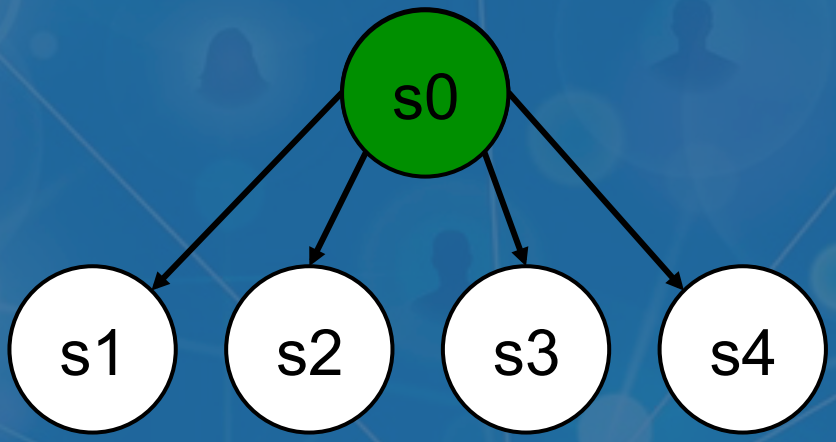
M(s1) = {(1,A)}

M(s2) = {(1,B)}

M(s3) = {(1,C)}



State Space Representation



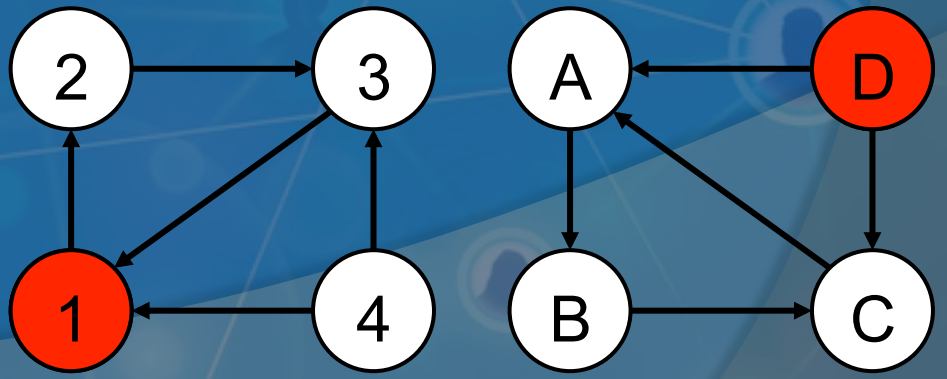
M(s0) = {0}

M(s1) = {(1,A)}

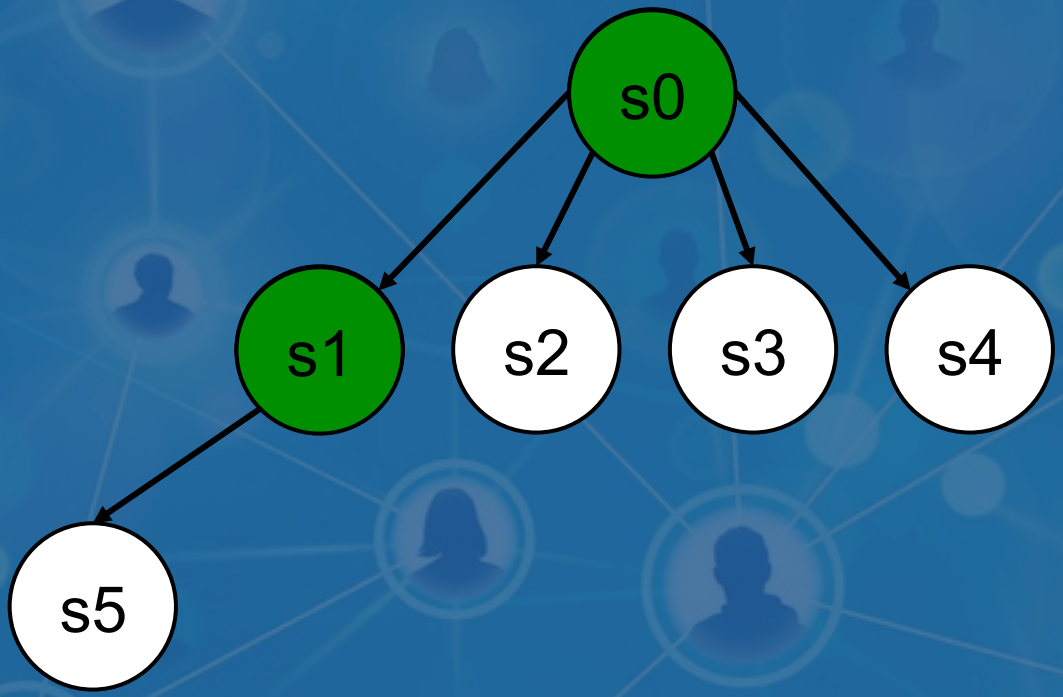
M(s2) = {(1,B)}

M(s3) = {(1,C)}

M(s4) = {(1,D)}



State Space Representation



M(s0) = {0}

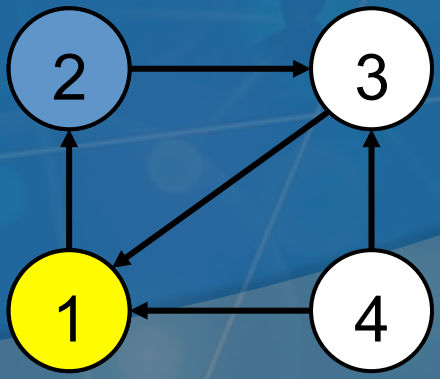
M(s1) = {(1,A)}

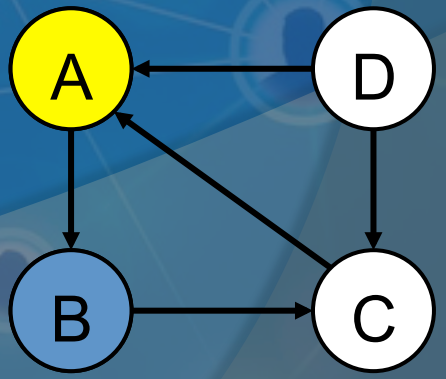
M(s2) = {(1,B)}

M(s3) = {(1,C)}

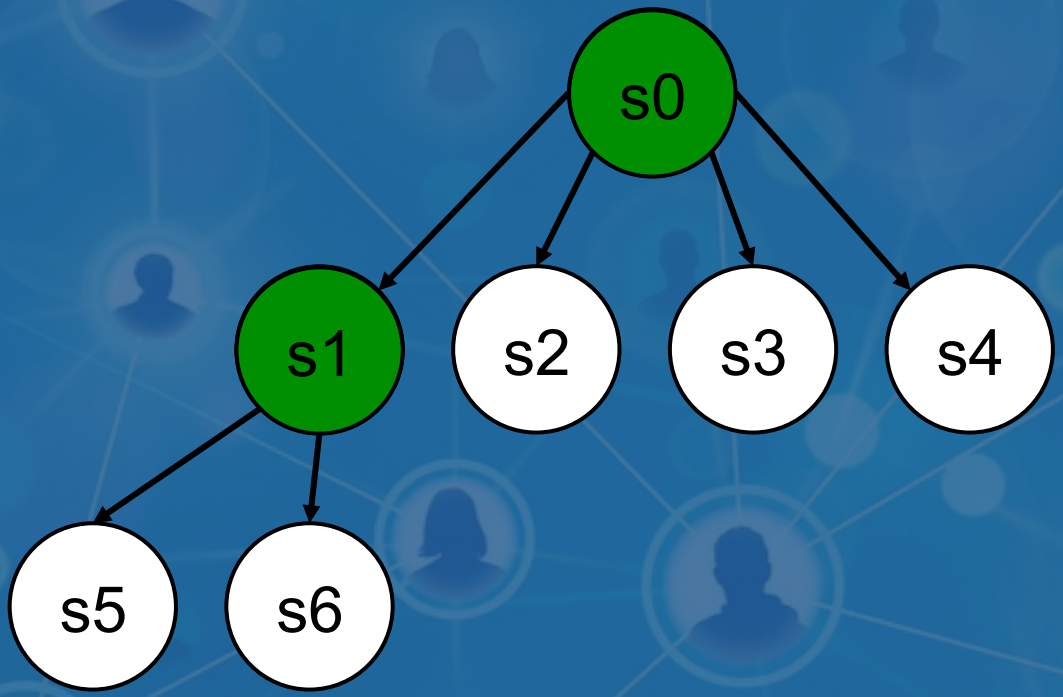
M(s4) = {(1,D)}

M(s5) = {(1,A) , (2,B)}





State Space Representation



M(s0) = {0}

M(s1) = {(1,A)}

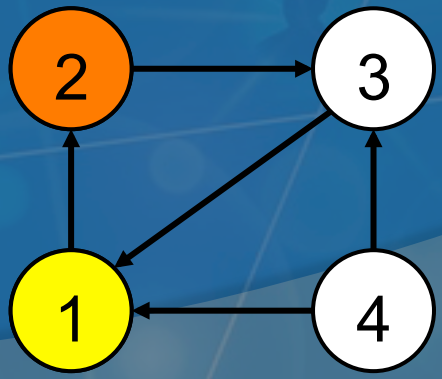
M(s2) = {(1,B)}

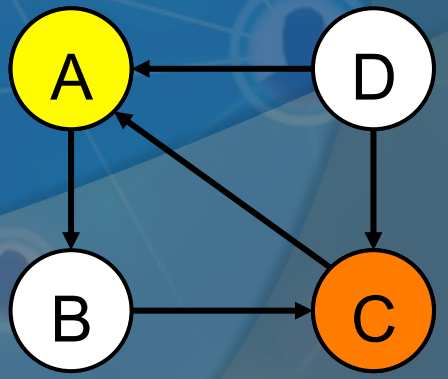
M(s3) = {(1,C)}

M(s4) = {(1,D)}

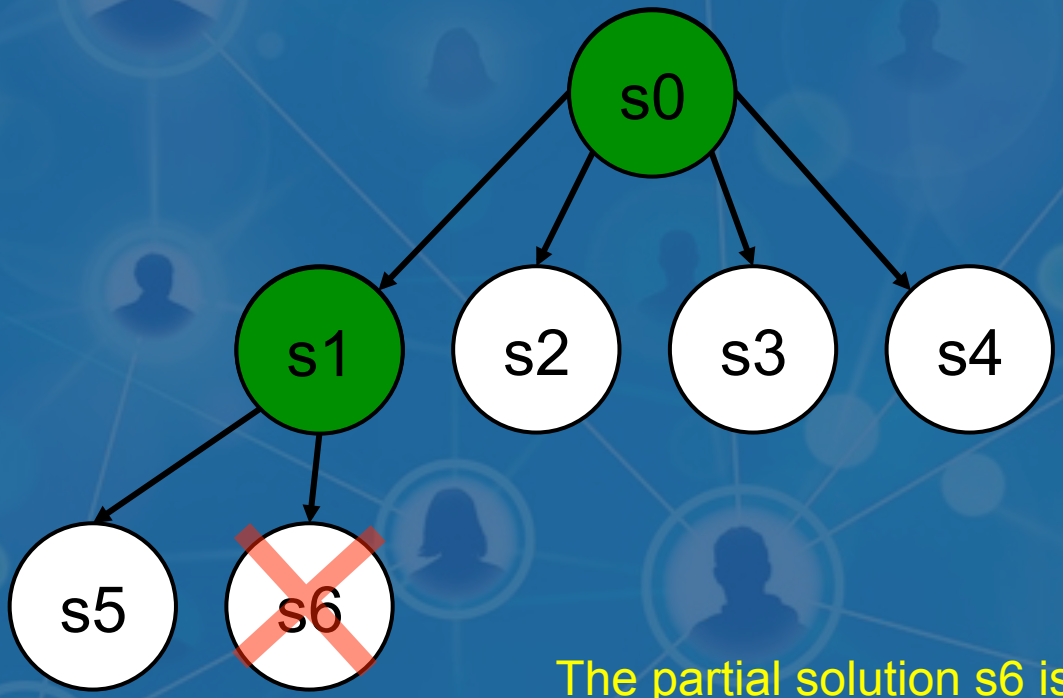
M(s5) = {(1,A) , (2,B)}

M(s6) = {(1,A) , (2,C)}





State Space Representation



M(s0) = {0}

M(s1) = {(1,A)}

M(s2) = {(1,B)}

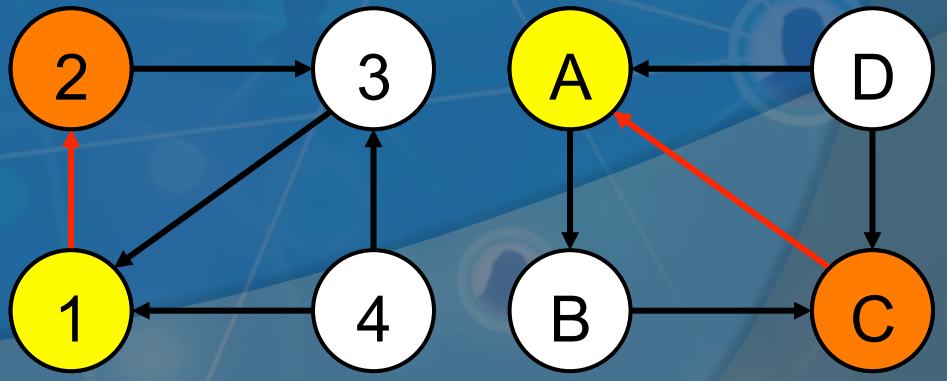
M(s3) = {(1,C)}

M(s4) = {(1,D)}

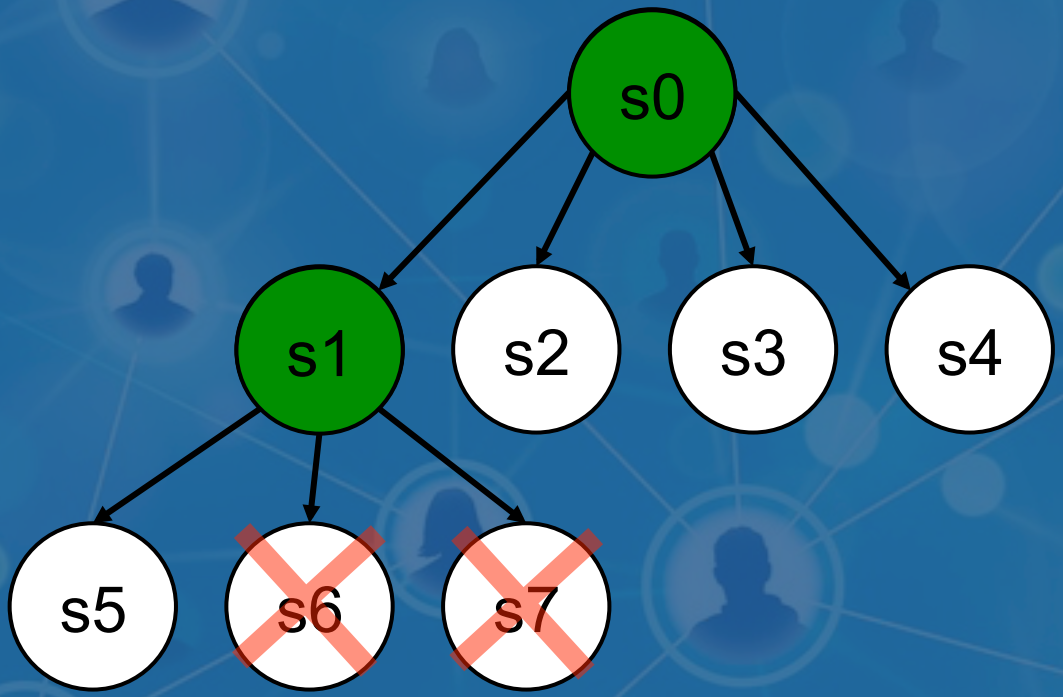
M(s5) = {(1,A) , (2,B)}

M(s6) = {(1,A) , (2,C)}

The partial solution s6 is inconsistent!



State Space Representation



M(s0) = {0}

M(s1) = {(1,A)}

M(s2) = {(1,B)}

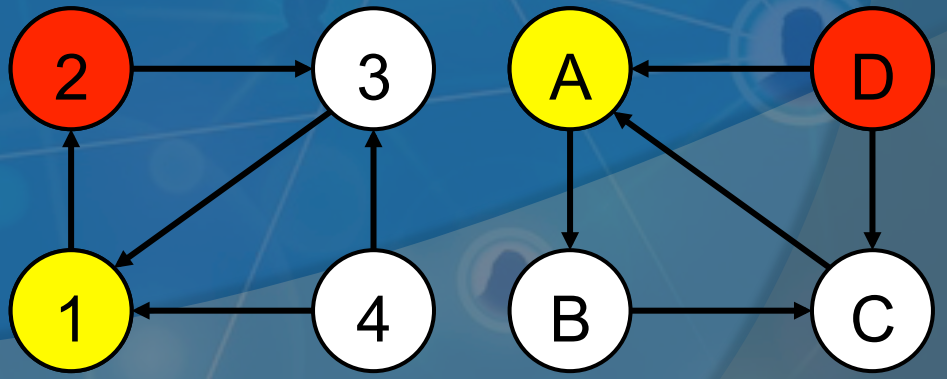
M(s3) = {(1,C)}

M(s4) = {(1,D)}

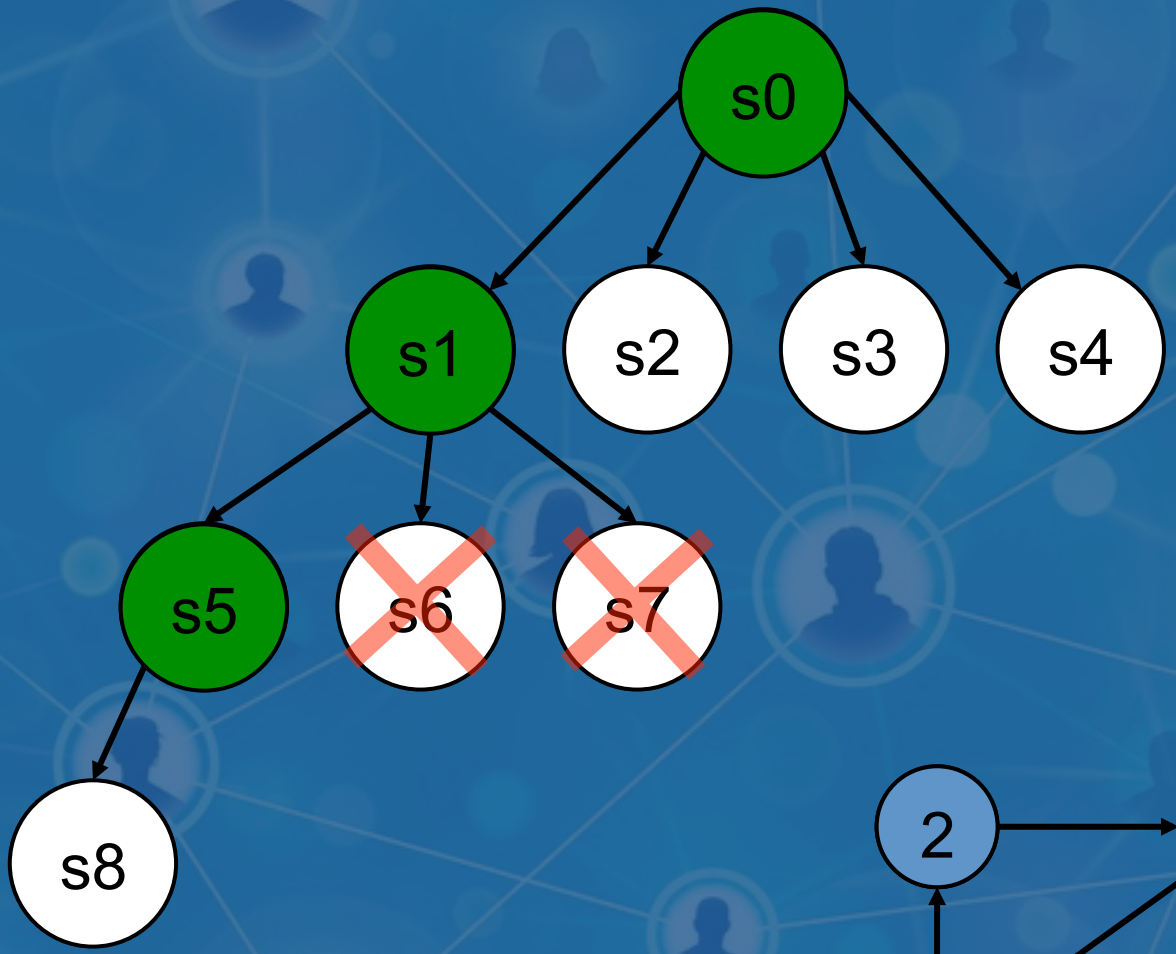
M(s5) = {(1,A) , (2,B)}

M(s6) = {(1,A) , (2,C)}

M(s7) = {(1,A) , (2,D)}



State Space Representation



M(s0) = {0}

M(s1) = {(1,A)}

M(s2) = {(1,B)}

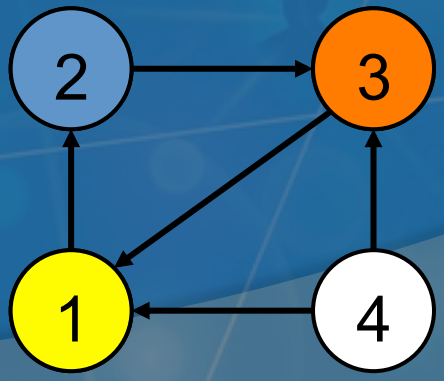
M(s3) = {(1,C)}

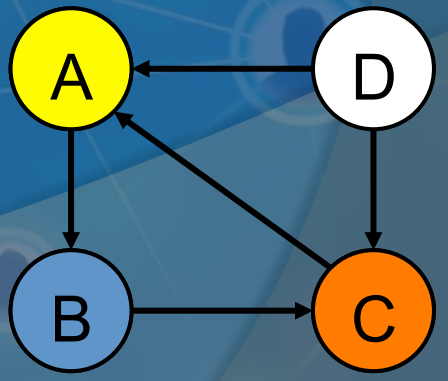
M(s4) = {(1,D)}

M(s5) = {(1,A) , (2,B)}

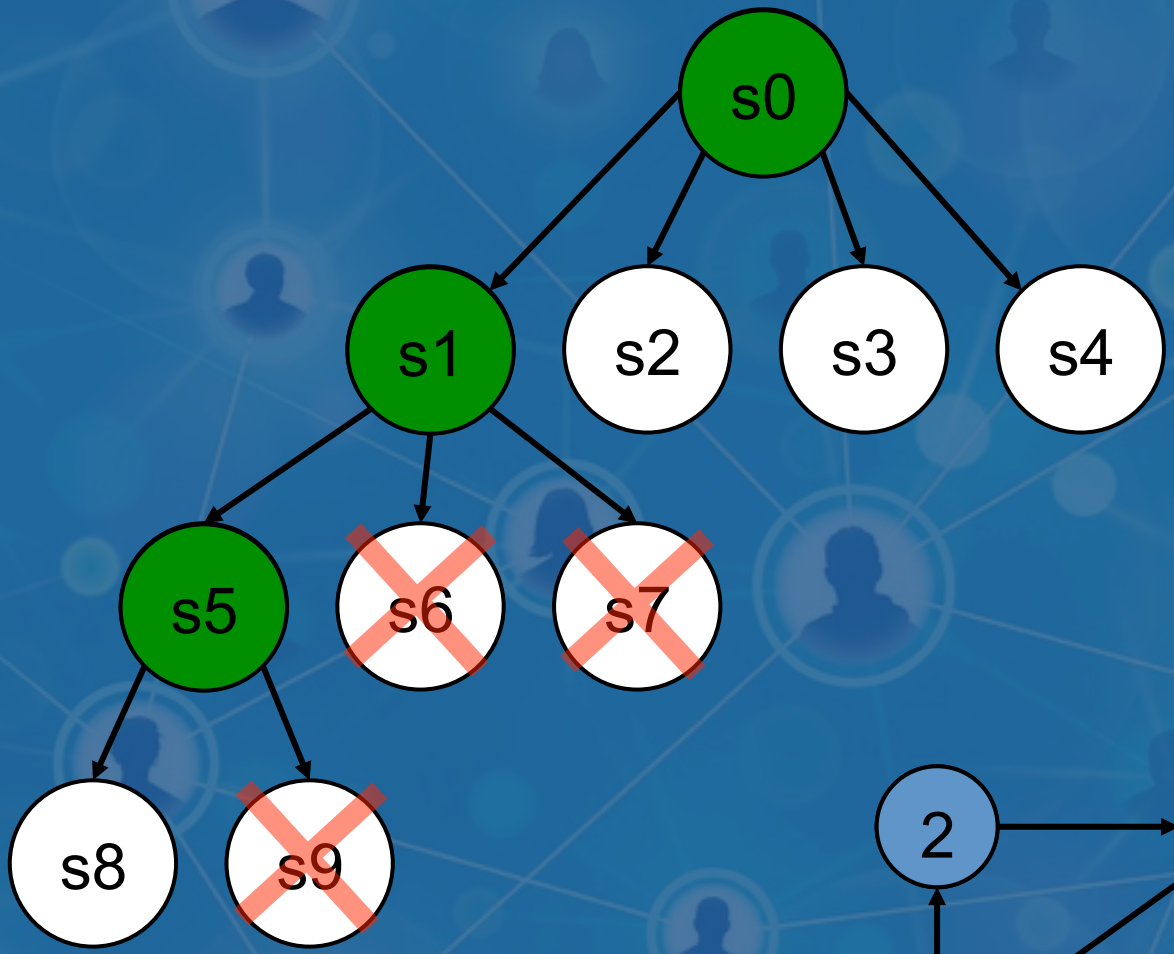
M(s6) = {(1,A) , (2,C)}

M(s8) = {(1,A) , (2,B), (3,C)} …





State Space Representation



M(s0) = {0}

M(s1) = {(1,A)}

M(s2) = {(1,B)}

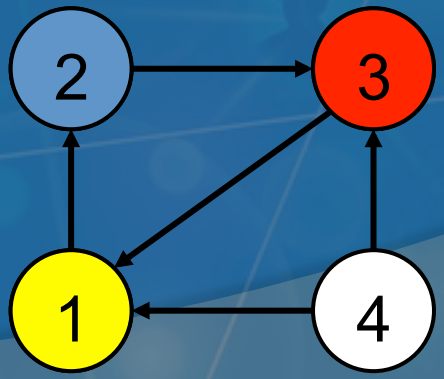
M(s3) = {(1,C)}

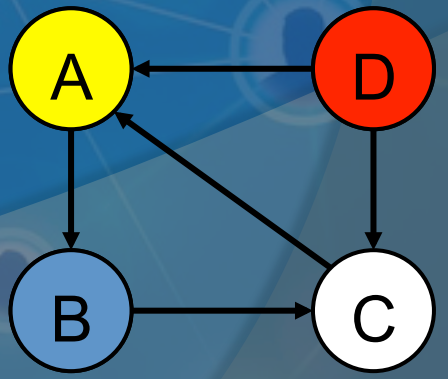
M(s4) = {(1,D)}

M(s5) = {(1,A) , (2,B)}

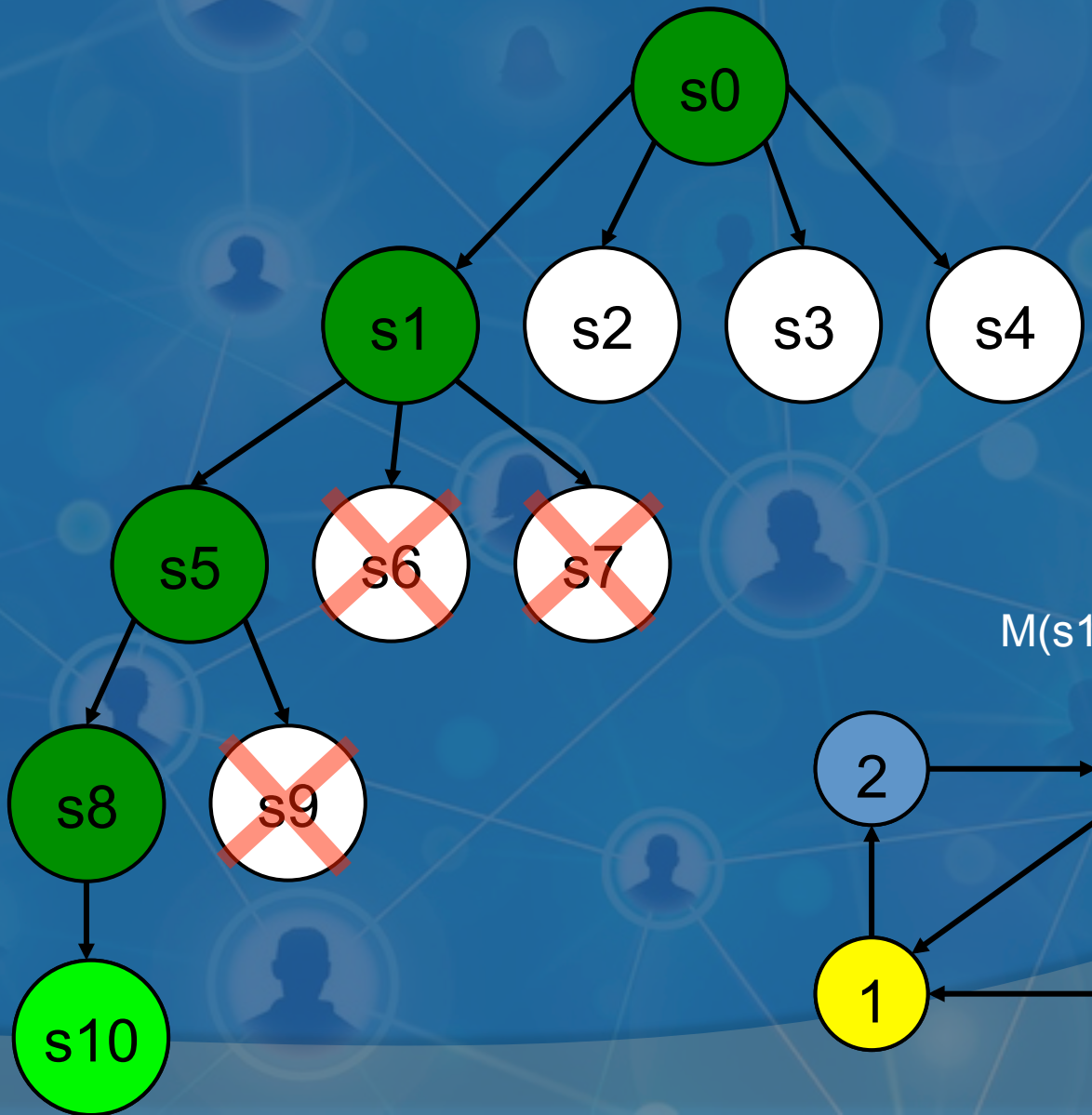
M(s6) = {(1,A) , (2,C)}

M(s9) = {(1,A) , (2,B), (3,D)} …





State Space Representation



M(s0) = {0}

M(s1) = {(1,A)}

M(s2) = {(1,B)}

M(s3) = {(1,C)}

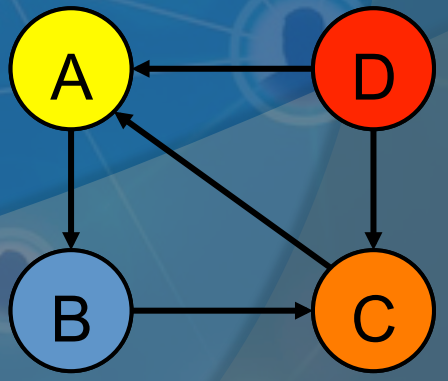
M(s4) = {(1,D)}

M(s5) = {(1,A) , (2,B)}

M(s6) = {(1,A) , (2,C)}

…

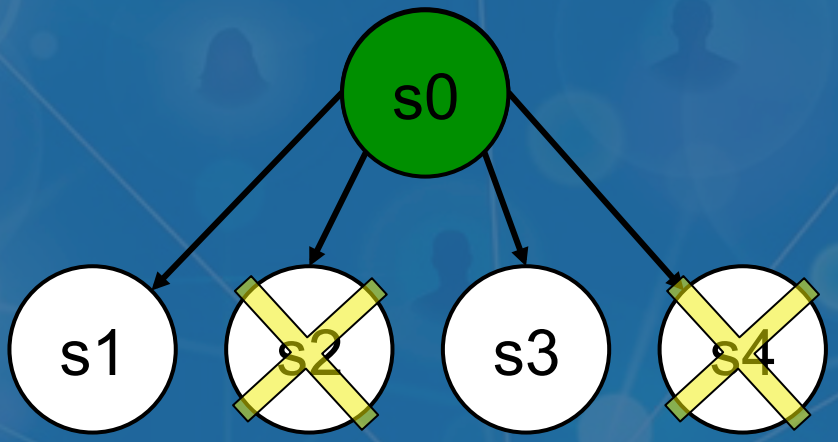
M(s10) = {(1,A) , (2,B) , (3,C) , (4,D)}



State Space search

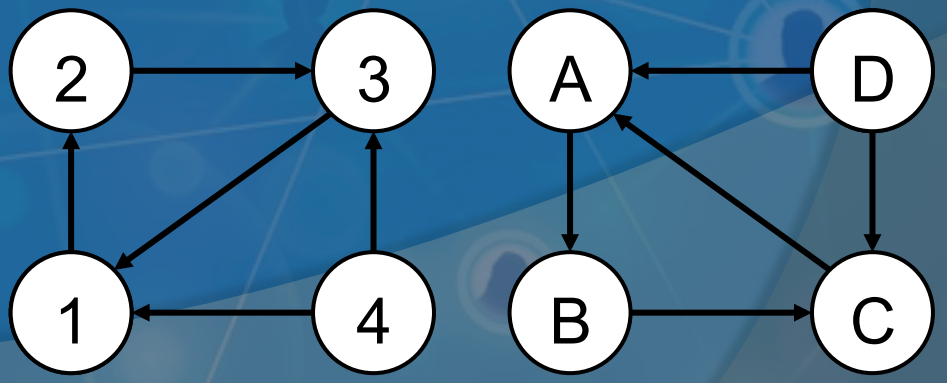
- If we are sure that the search space is a tree, then
  - The search space does not need to be entirely stored in memory while it is being explored if the tree is visited Depth First
  - No need to check if a state has already been visited…
  - … thus the algorithm is much more efficient!

Pruning the State Space



M(s0) = {0}
M(s1) = {(1,A)}
M(s2) = {(1,B)}
M(s3) = {(1,C)}
M(s4) = {(1,D)}

S2 and S4 are consistent states, but it can be shown that no solution can be derived from them…
If the algorithm can prune states like them in advance, it can save a lot of time,



An algorithm based on SSR

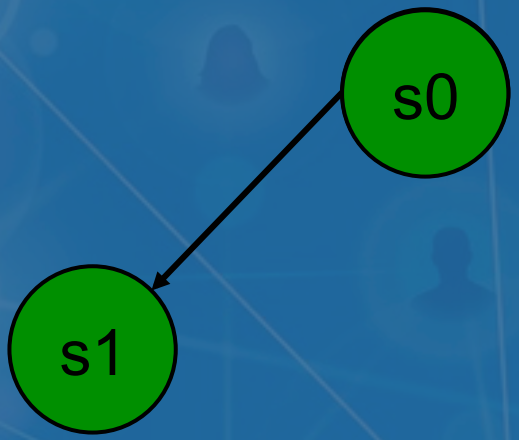
- How the next candidate pair is chosen?
  - This choice must ensure that the state space is a tree
  - The choice also impacts on the number of states needed to reach a solution
- How the unfruitful states are pruned?
  - A good pruning criterion can greatly reduce the number of states
  - … but beware: the criterion should be fast to compute!

Matching time = Number of states * Time spent on each state

Basic ideas of VF2

- For generating next states:
  - At each state, the algorithm keeps the so called *Terminal sets:*
    - **Tin** : set of nodes that have an edge going into a node already mapped
    - **Tout**: set of nodes that have an edge coming out of a node already mapped
  - For the pattern, the node in Tout with the smallest id is chosen; for the target, all nodes in Tout are chosen
    - (Unless Tout is empty; if it is, Tin is used instead)
  - Tin and Tout are built “incrementally” when visiting the state tree

Terminal sets



M(s0) = {}

M(s1) = {(1,A)}

Tin1(s1) = {3,4}

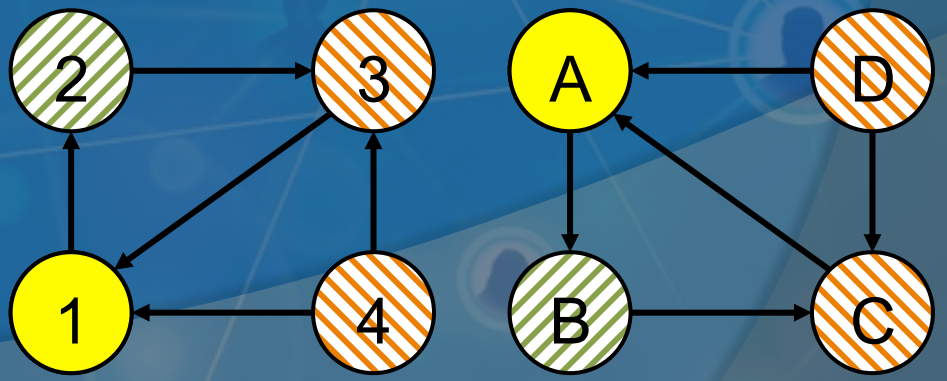
Tin2(s1) = {C,D}

Tout1(s1)={2}

Tout2(s1)={B}

Tin

Tout



Basic ideas of VF2

- For pruning the states:
  - Rules that check arc consistency (Rpred/Rsucc) and label compatibility
  - Rules that check the cardinality of the intersections of the predecessors/successors of a node with Tin/Tout (Rterm), giving a “1-step” look-ahead
  - Rules that check the cardinality of the difference between predecessors/successor of a node and Tin/Tout (“2-steps” look-ahead)

Pruning example

M(s0) = {}

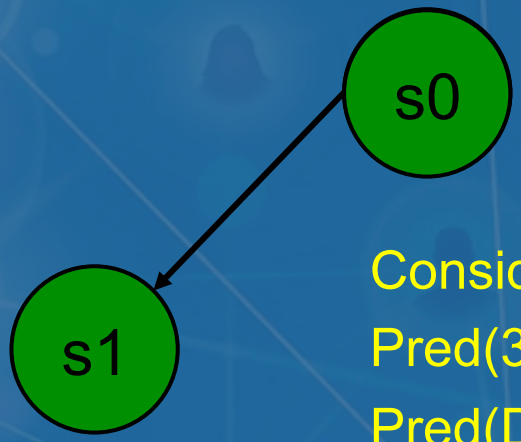
M(s1) = {(1,A)}

Tin1(s1) = {3,4}

Tin2(s1) = {C,D}

Tout1(s1)={2}

Tout2(s1)={B}



Consider the pair (3, D)

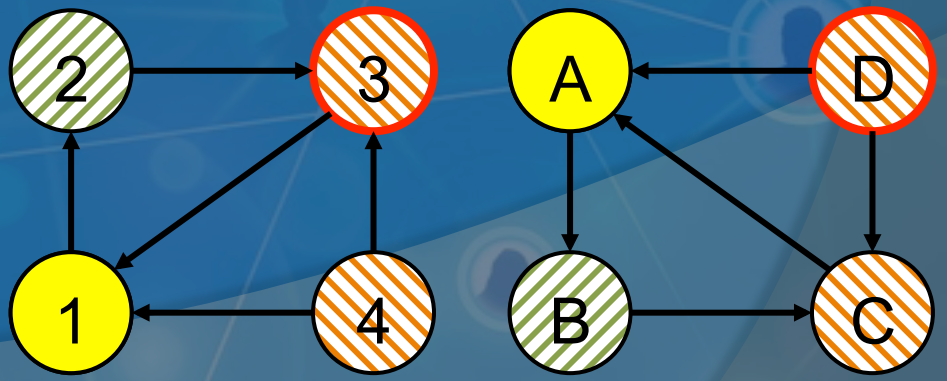
Pred(3) ∩ Tin(s1) = {4}

Pred(D) ∩ Tin(s1) = {}

Since Card({4}) > Card({}), the pair can be excluded

Tin

Tout



Limits of VF2

- Label information, that is usually available in real world examples, is not used neither for choosing the visit order nor for lookahead (it is only used for checking consistency)
- The choice of the next candidate does not attempt to choose earlier a candidate with more constraints (so as to maximize the effect of pruning)

VF3: the proposed improvement

- We change the order of visit of the state space by taking into account both the labels and the constraints relative to each node

- We extend the look-ahead rules so as to take into account labels and possibly other information (e.g. about structure)

First improvement: node reordering

- Before starting the matching, the nodes in the pattern are reordered by taking the following factors into account:
  - Number of edges connecting the node to the previous ones
    - Each edge will introduce a “constraint” during the matching; putting nodes with more constraints first is a heuristic commonly used in Constraint Satisfaction literature
  - Frequency of occurrence of the label among the nodes in the target
  - Frequency of occurrence of nodes with a compatible valence in the target
    - Thus, the nodes in the least repetitive parts of the target are tried first

First improvement: node reordering

- This reordering of the pattern nodes can be done before the matching starts, for a scenario of searching one pattern against multiple targets
- We have also modified the algorithm to precompute the Tin1 and Tout1 set for each state (they only depend on the “depth” of the state in the state tree)

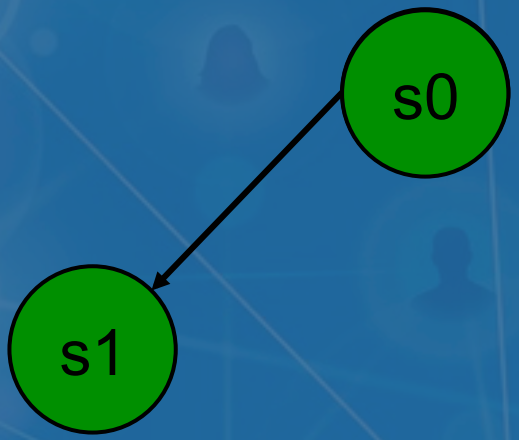
Second improvement: terminal set partitioning

- The nodes are divided into equivalence classes constructed so that two nodes that may be matched must be in the same class
  - In the simplest case, the node label is used for defining the classes
  - For graph isomorphism also other information can be included (e.g. node valence)

Second improvement: terminal set partitioning

- In each state, instead of having a single terminal set for each graph, we have a separate terminal set for each class
- The look-ahead rules evaluate separately the neighbor counts for each terminal set

Terminal set partitioning



M(s0) = {}

M(s1) = {(1,A)}

Tin1(s1) = {3,4}

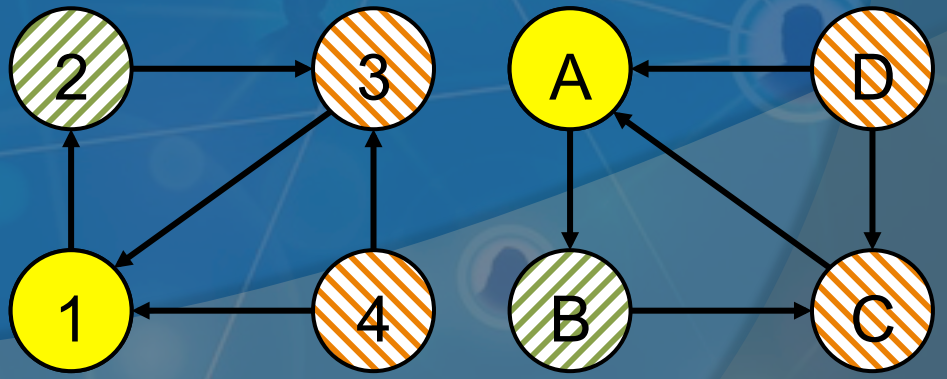
Tin2(s1) = {C,D}

Tout1(s1)={2}

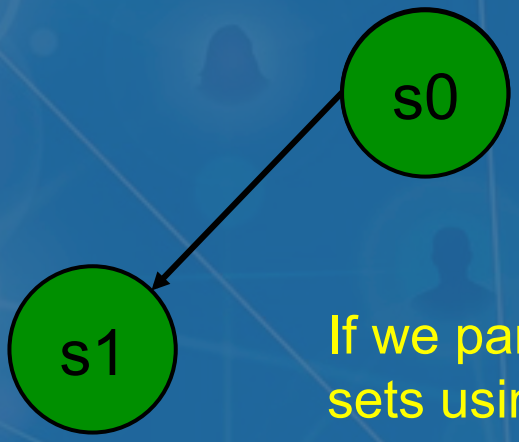
Tout2(s1)={B}

Tin

Tout



Terminal set partitioning



If we partition the terminal sets using information about the number of incoming/ outcoming edges…

M(s0) = {}

M(s1) = {(1,A)}

Tin1(s1) = {3,4}

Tin2(s1) = {C,D}

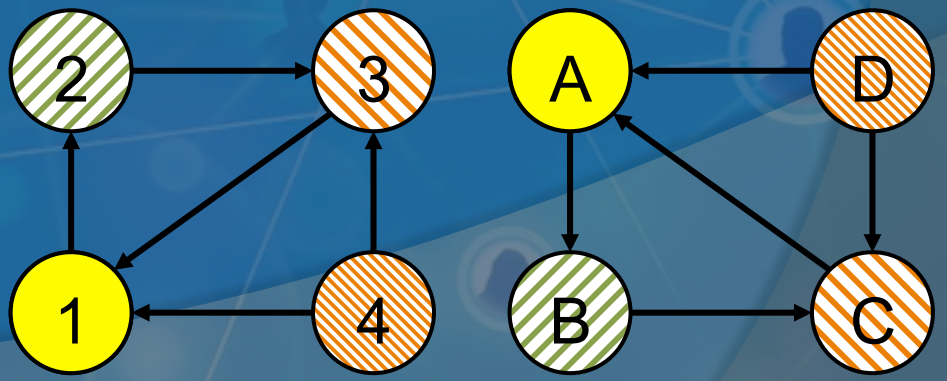
Tout1(s1)={2}

Tout2(s1)={B}

Tin, class 1

Tin, class 2

Tout



Second improvement: terminal set partitioning

- Trade off:
  - More classes => less candidate mappings need to be explored
  - More classes => greater cost for maintaining the terminal sets and checking the look-ahead rules
- Thus in some cases it may make sense to have less classes by grouping more than one label in a same class

Graphs in Biological applications

- Several biological data can be meaningfully represented by graphs
  - Molecules
  - Protein structures
  - Protein contact maps
  - Interaction networks
  - …

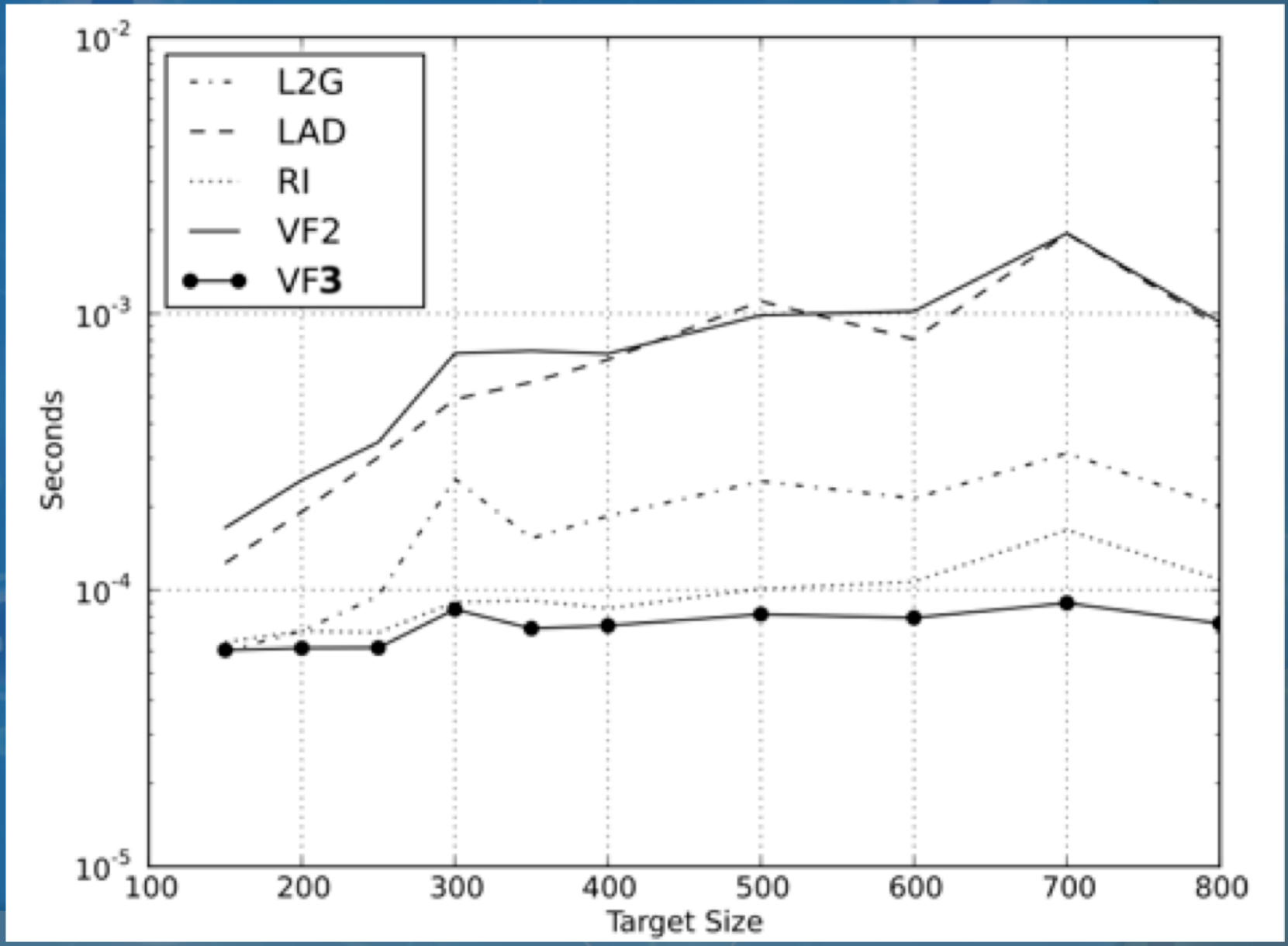
Graphs in Biological applications

- Graphs in Biological applications have some common traits:
  - Large to very large (up to tens of thousands of nodes)
  - Usually very sparse
  - Labels from a small set (e.g. elements, amino acids etc.)
  - Presence of highly repetitive substructures

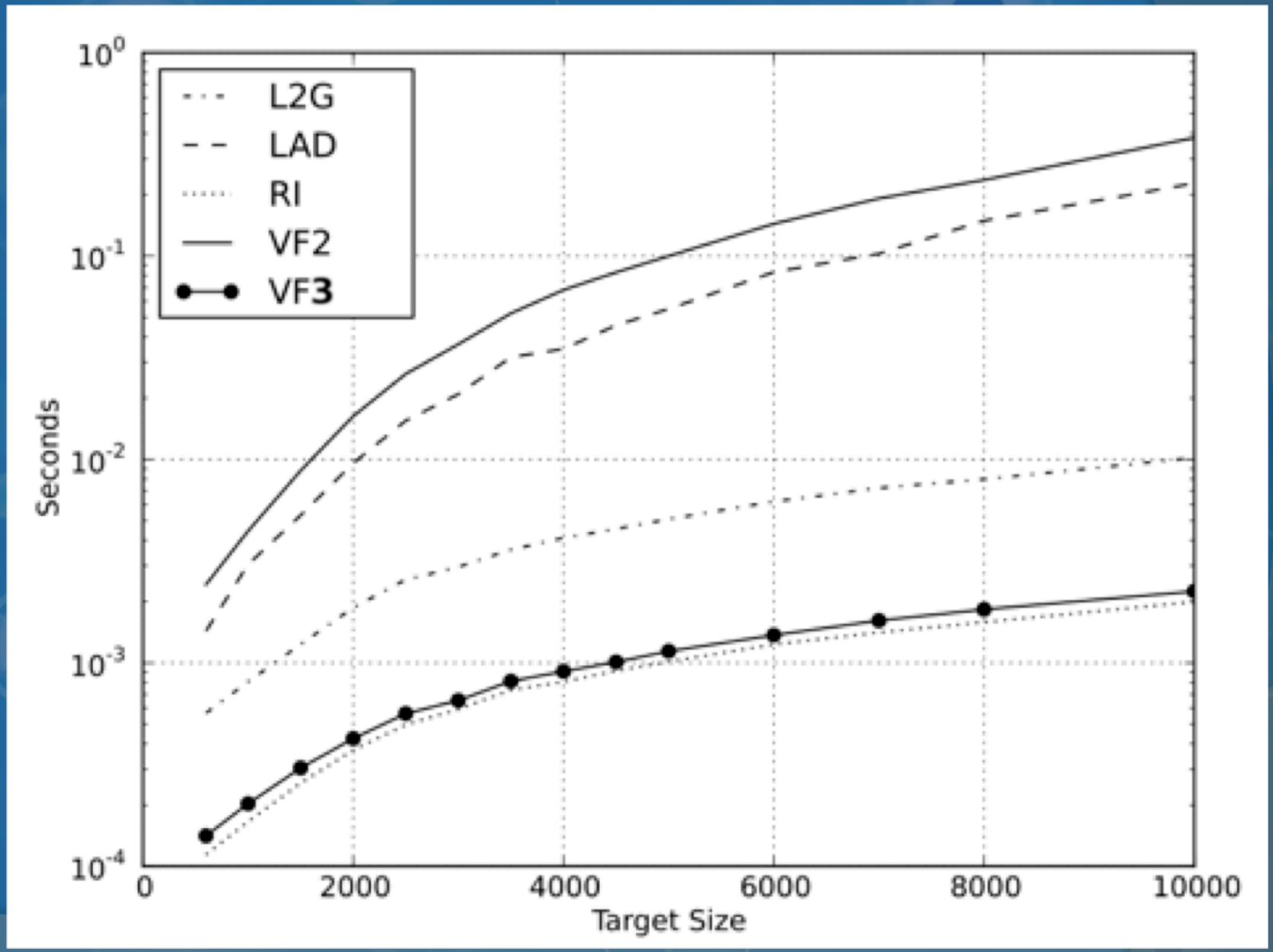
Experimental evaluation

- We have used a subset of the database of the ICPR2014 contest on Graph Matching for Pattern Search in Biological Database
  - Contact Maps
  - Protein structures
- We have compared VF3 with VF2 and with three recent algorithms participating to the contest: RI, LAD and L2G

Contact maps



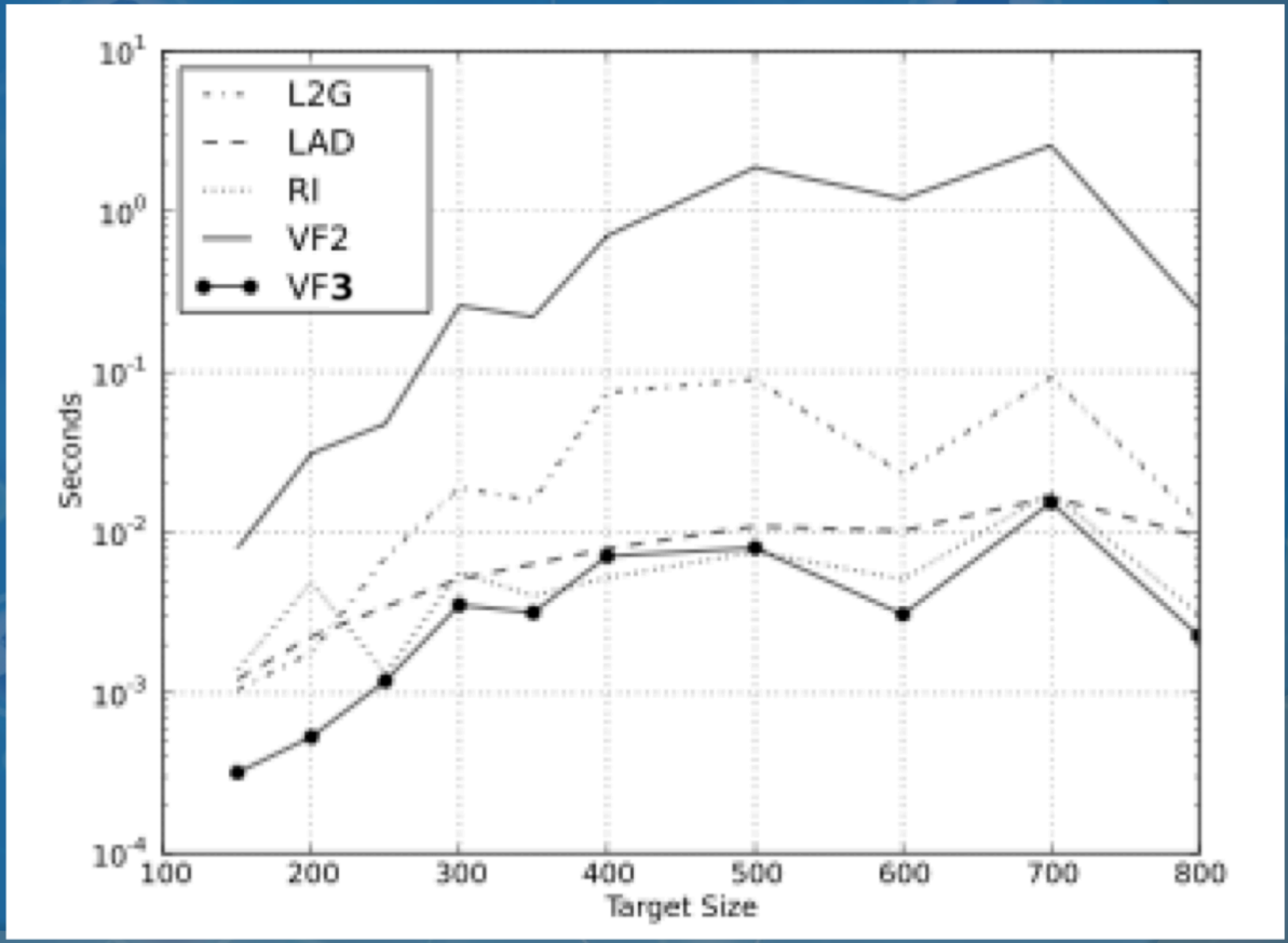
Proteins


L2G
LAD
RI
VF2
VF3
Seconds
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
0
2000
4000
6000
8000
10000
Target Size

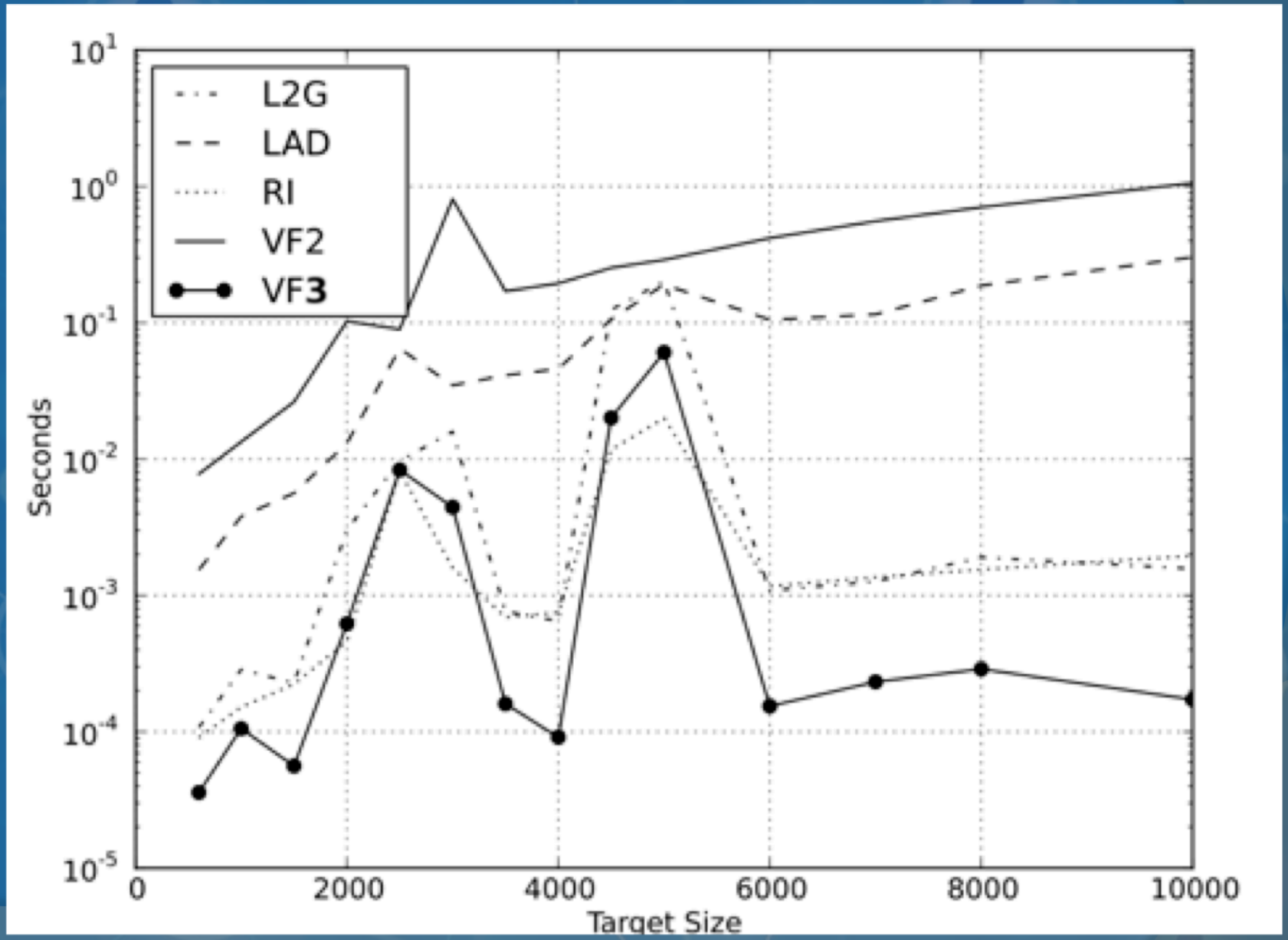
Experimental evaluation

- In order to check the dependency of the algorithm on the number of different labels, we have repeated the experiments by artificially reducing the label diversity in the graphs

Contact maps – reduced labels

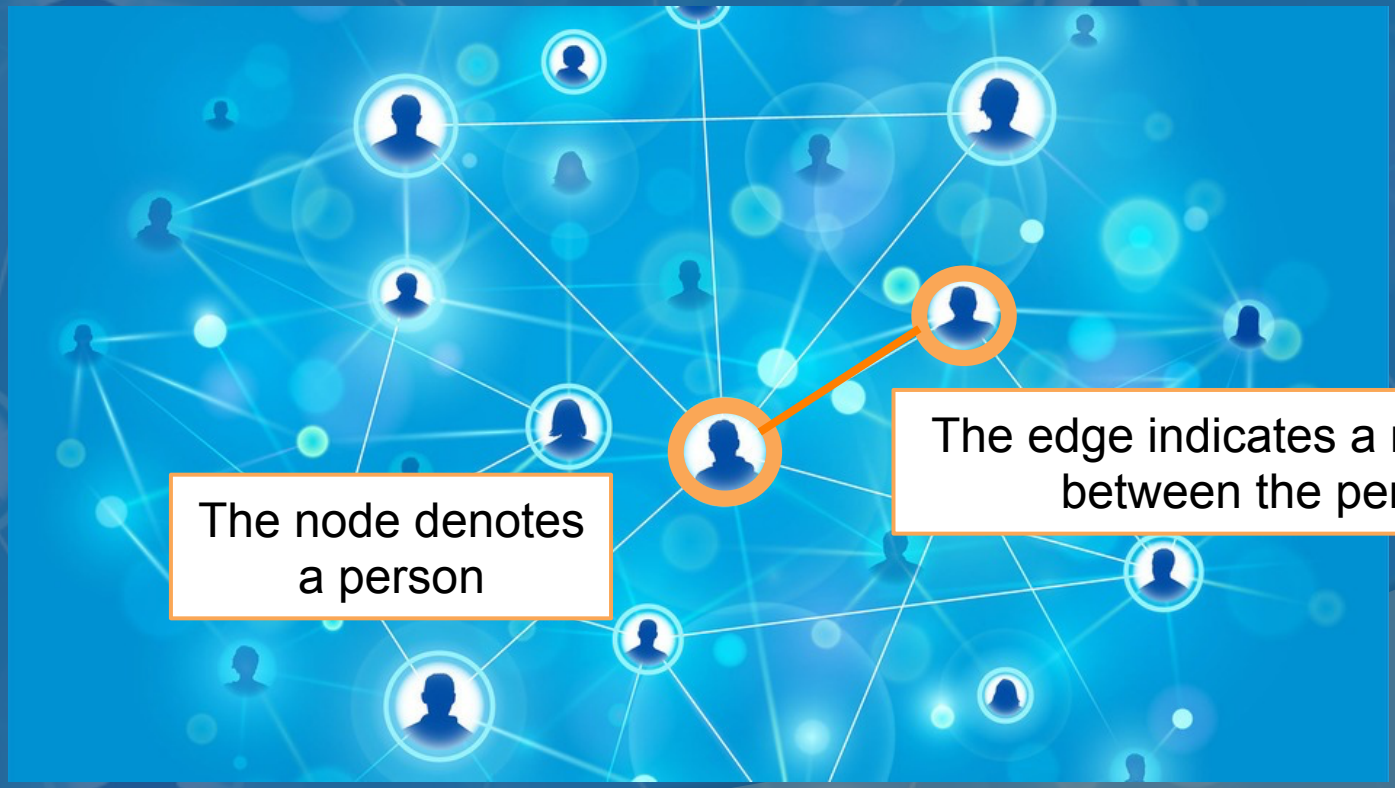


Proteins – reduced labels



Social network analysis

Social networks can be modeled as graphs



The problem: a few examples

- Equivalent actors occupy indistinguishable structural locations in a network. Structural location can be defined in terms of sub graph isomorphism.
- Frequent sub graph allows to discover frequent substructures
  - generate frequent substructure candidates;
  - process the generated candidate sub graphs so as to identify the desired frequent sub graphs by sub graph isomorphism).

Experimental Results

- DATASET: a synthetic dataset has been generated according to two well known models of complex networks:
  - Small World networks
  - Scale Free networks
- It has been shown (*) that such models approximate very well the characteristics of social networks.

(*) S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, D.-U. Hwang, Complex networks: Structure and dynamics, Physics Reports, 2006

Small World networks

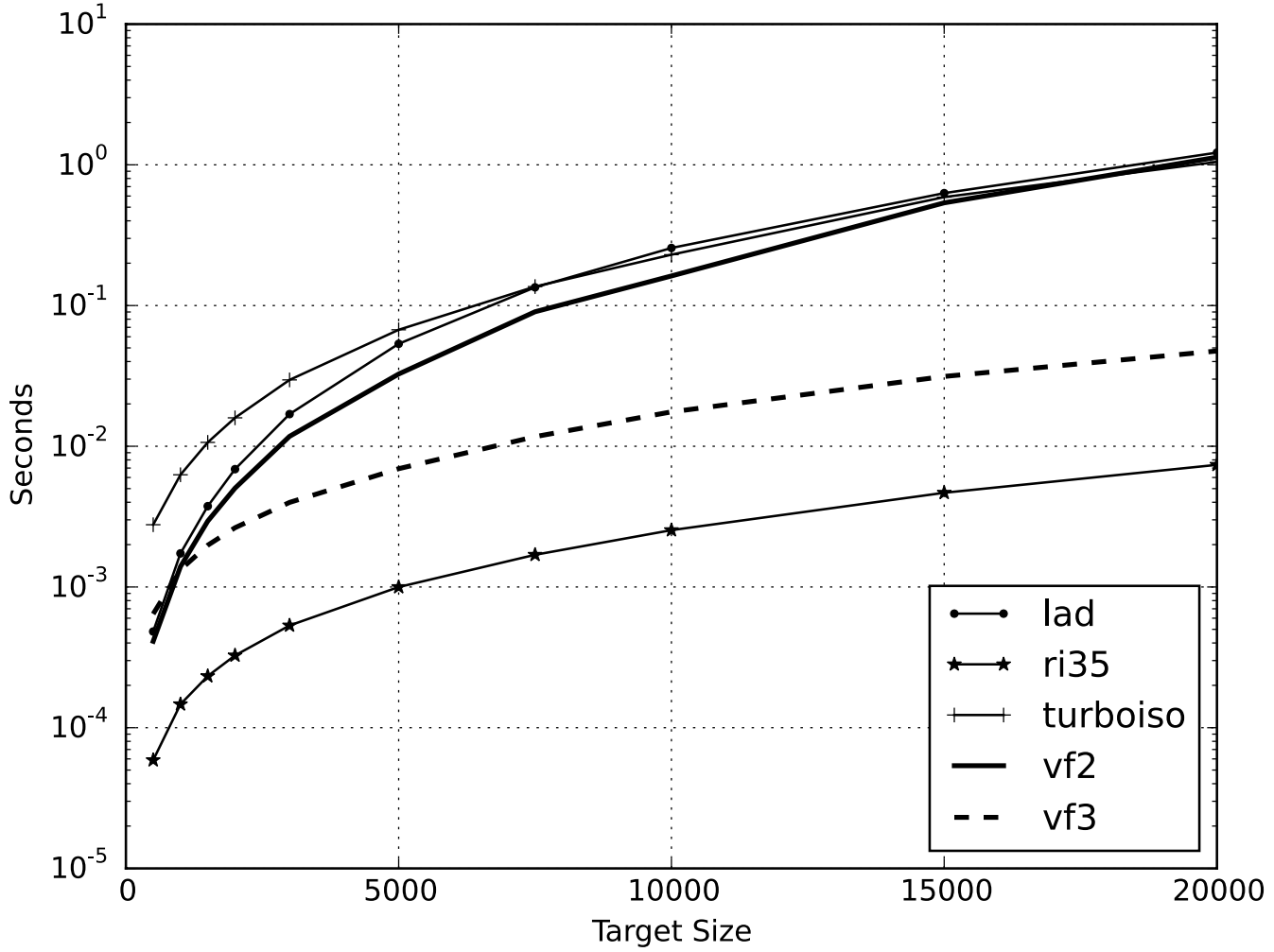
- Even though most pairs of nodes are not directly connected, there is an upper bound to the number of steps connecting any two nodes:
  - $D(n1,n2) \leq c*\log(N)$
- E.g. the theory about six degrees of separation between any two persons

Scale Free networks

- The degrees of the nodes are not uniformly distributed. There are many nodes with low degree and few nodes with high degree.
- More formally, if we sort the nodes according to their degree, the k-th node will have a degree:
  - $d_k = c^* e^{-\lambda^* k}$

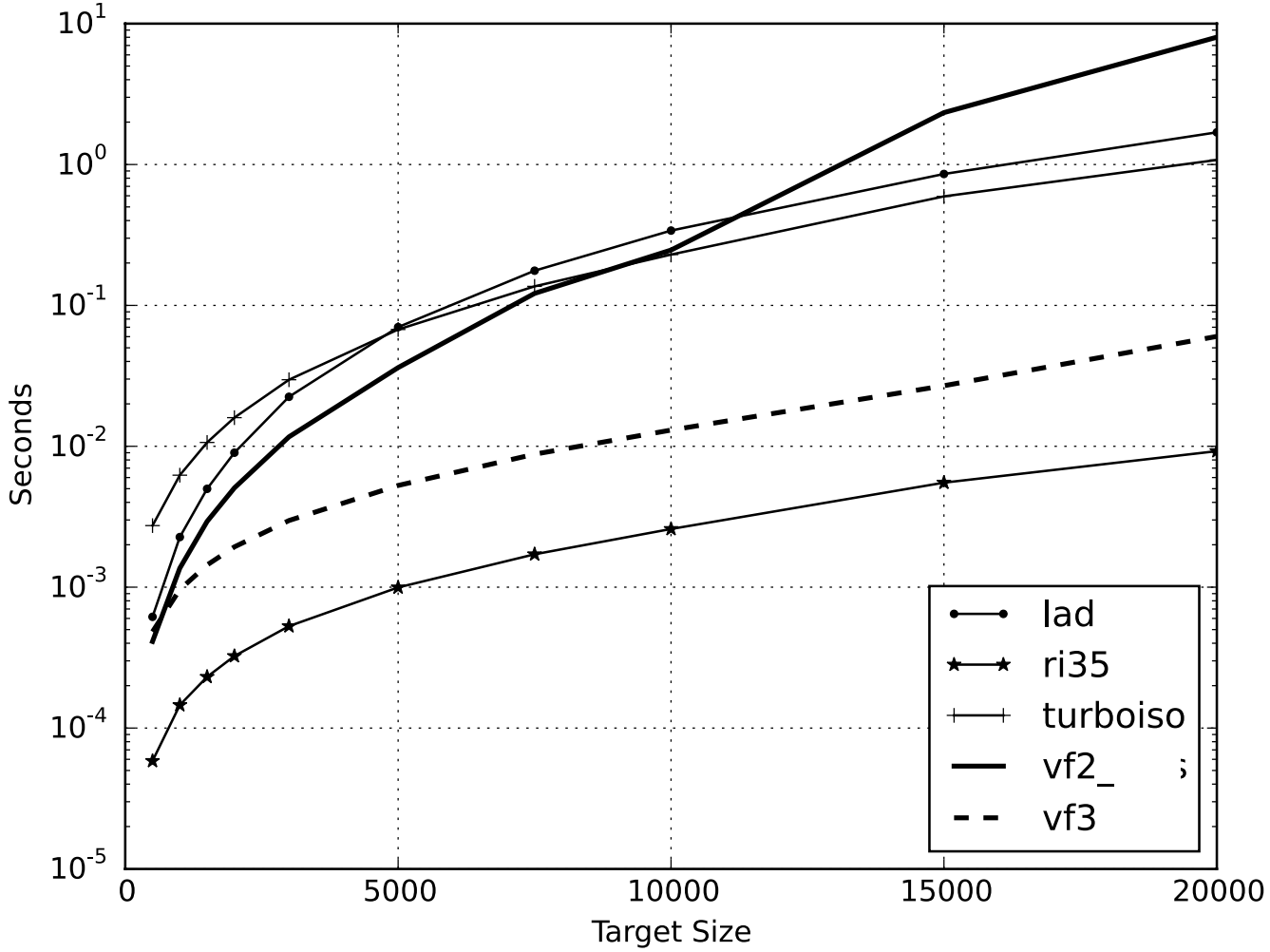
Experimental Results

Small word networks (256 labels)



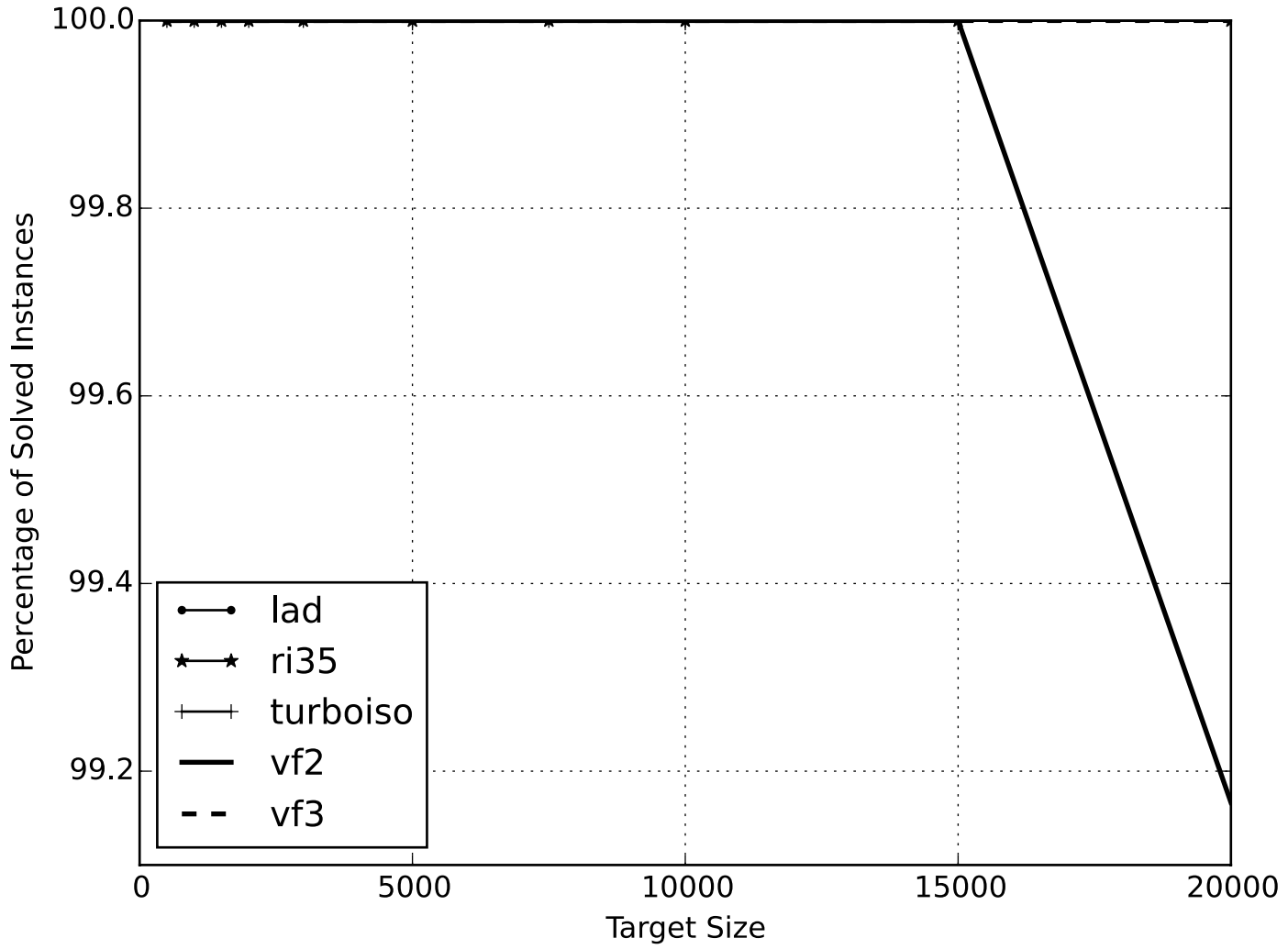
Experimental Results

Small world networks (128 labels)



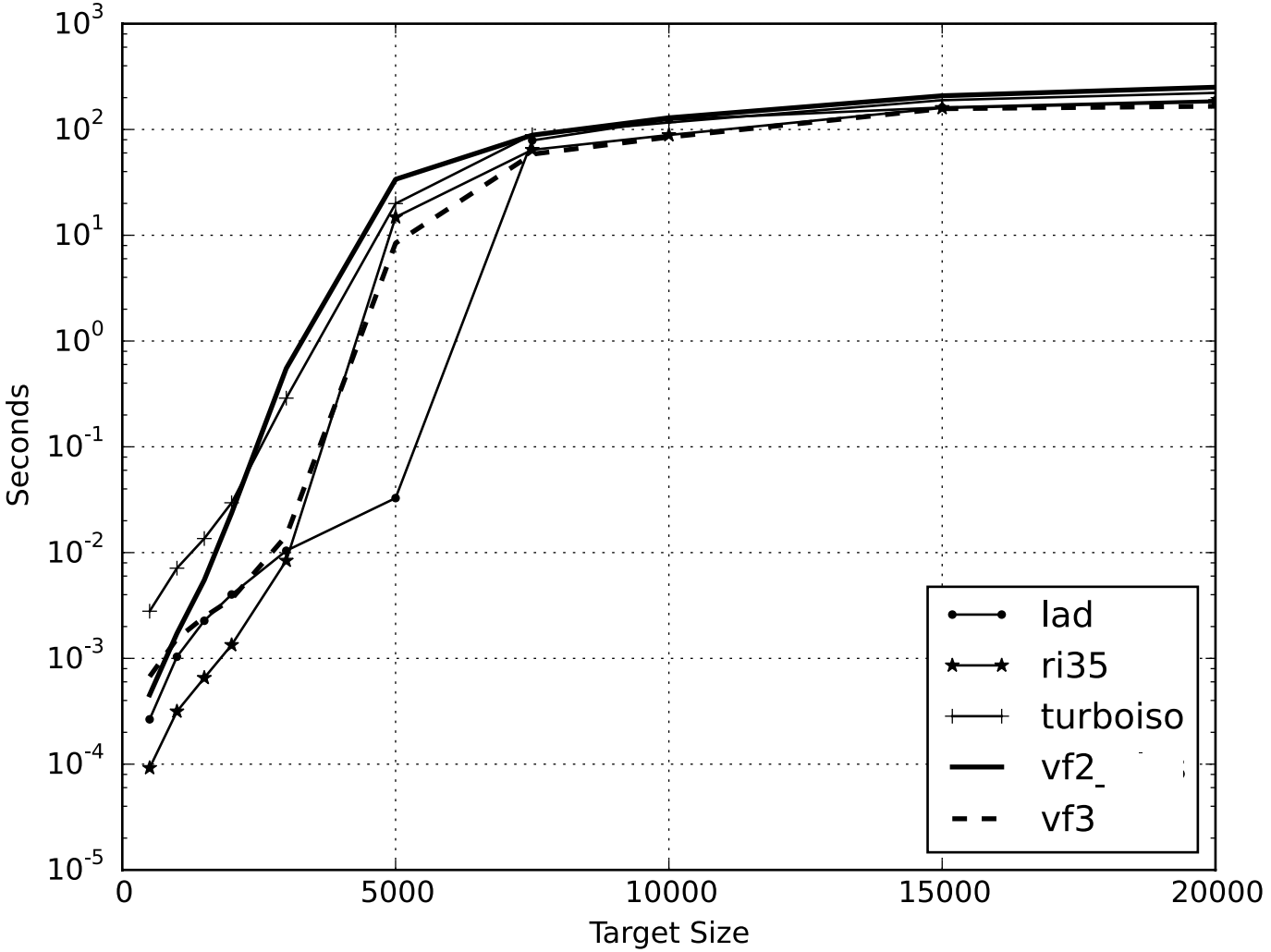
Experimental Results

Small world networks (128 labels)



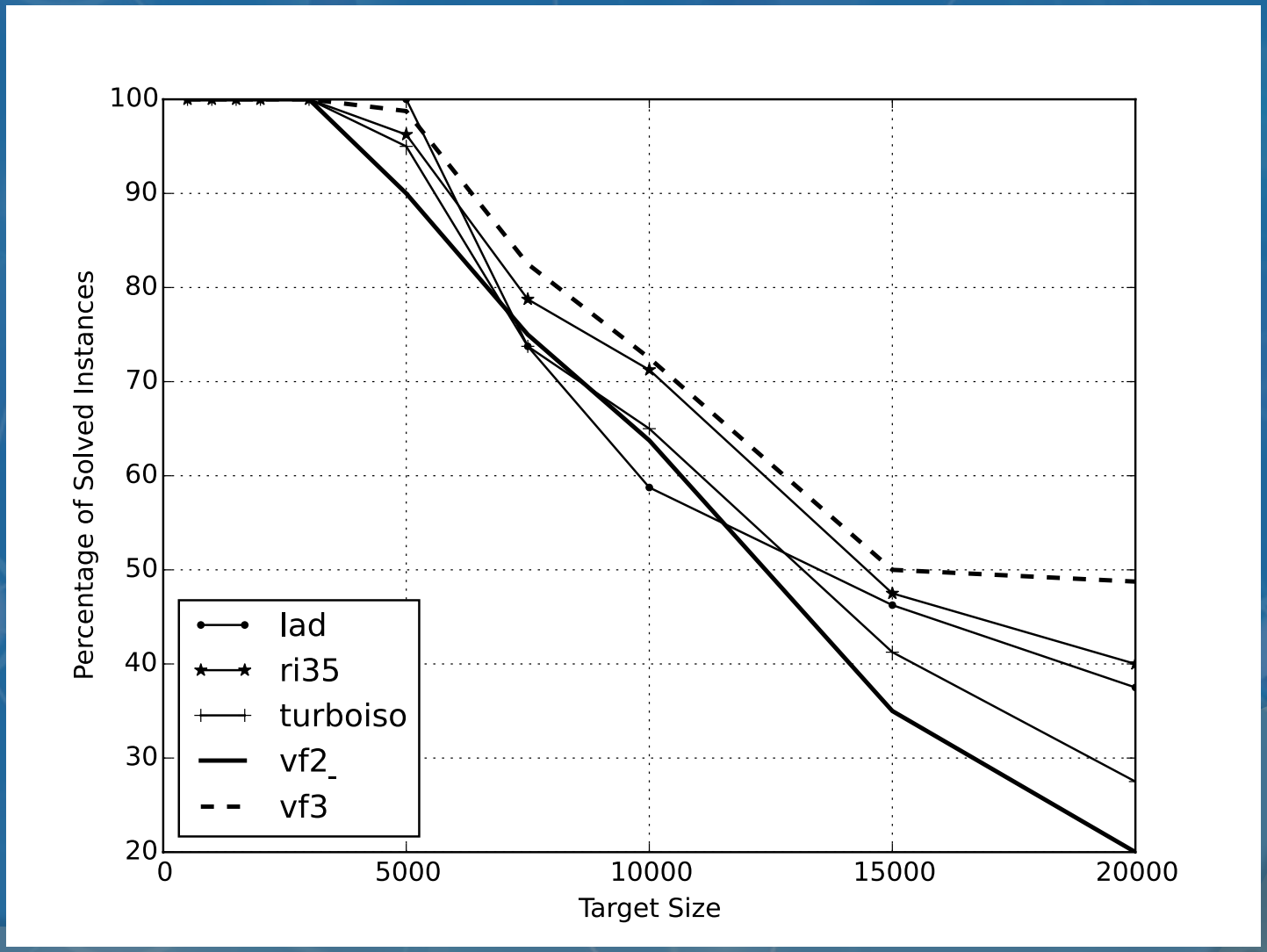
Experimental Results

Scale free graphs (256 labels)



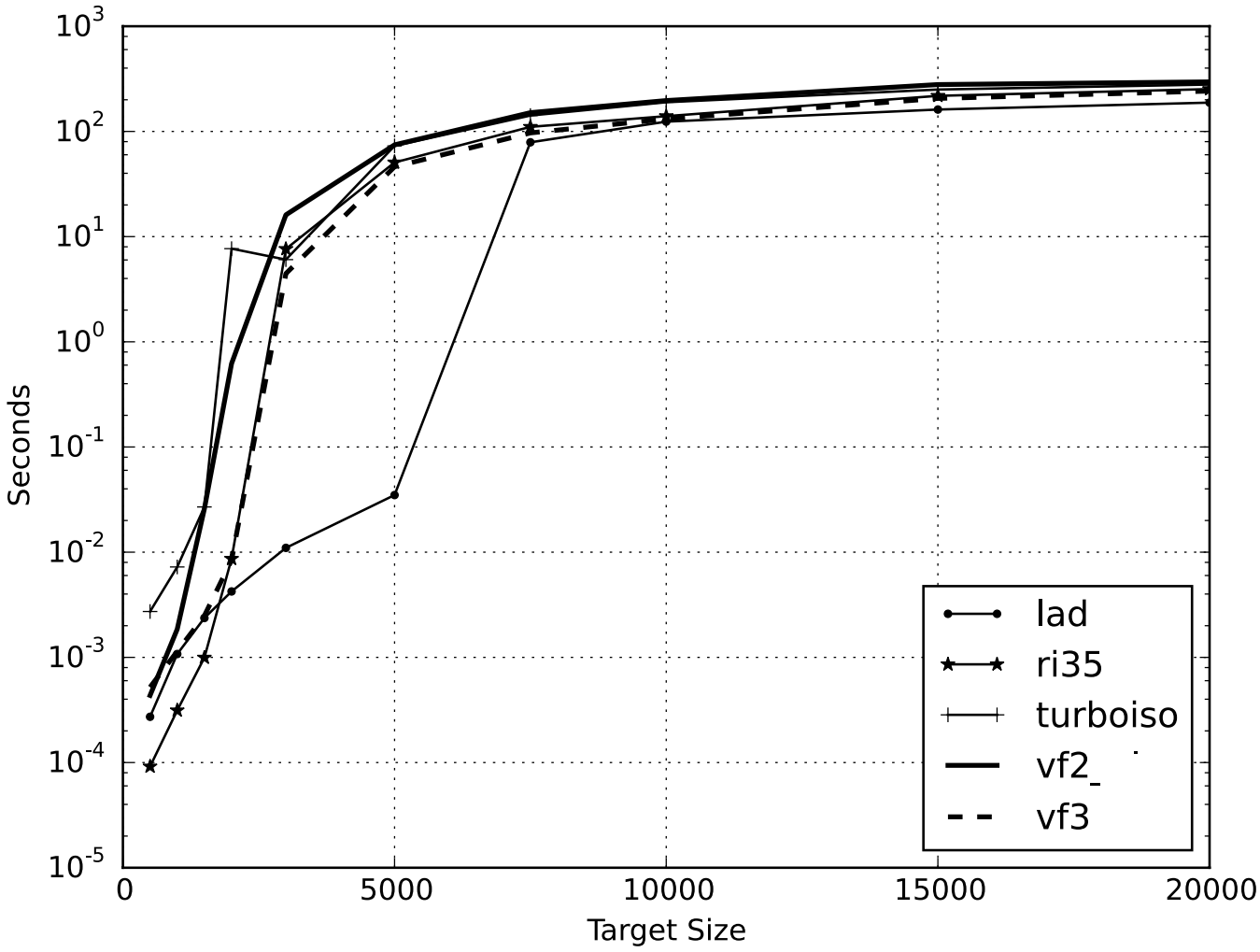
Experimental Results

Scale free graphs (256 labels)



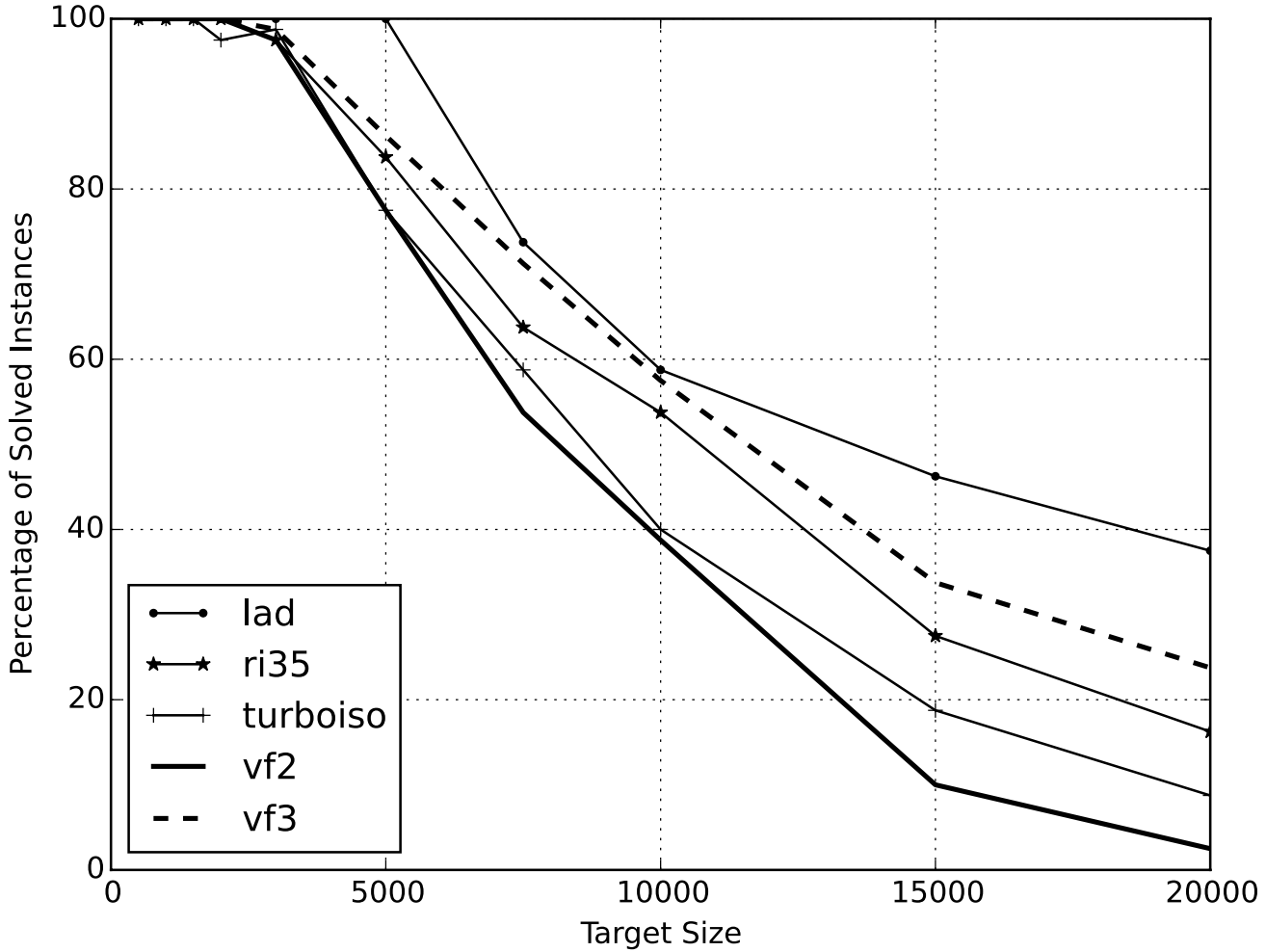
Experimental Results

Scale free graphs (128 labels)



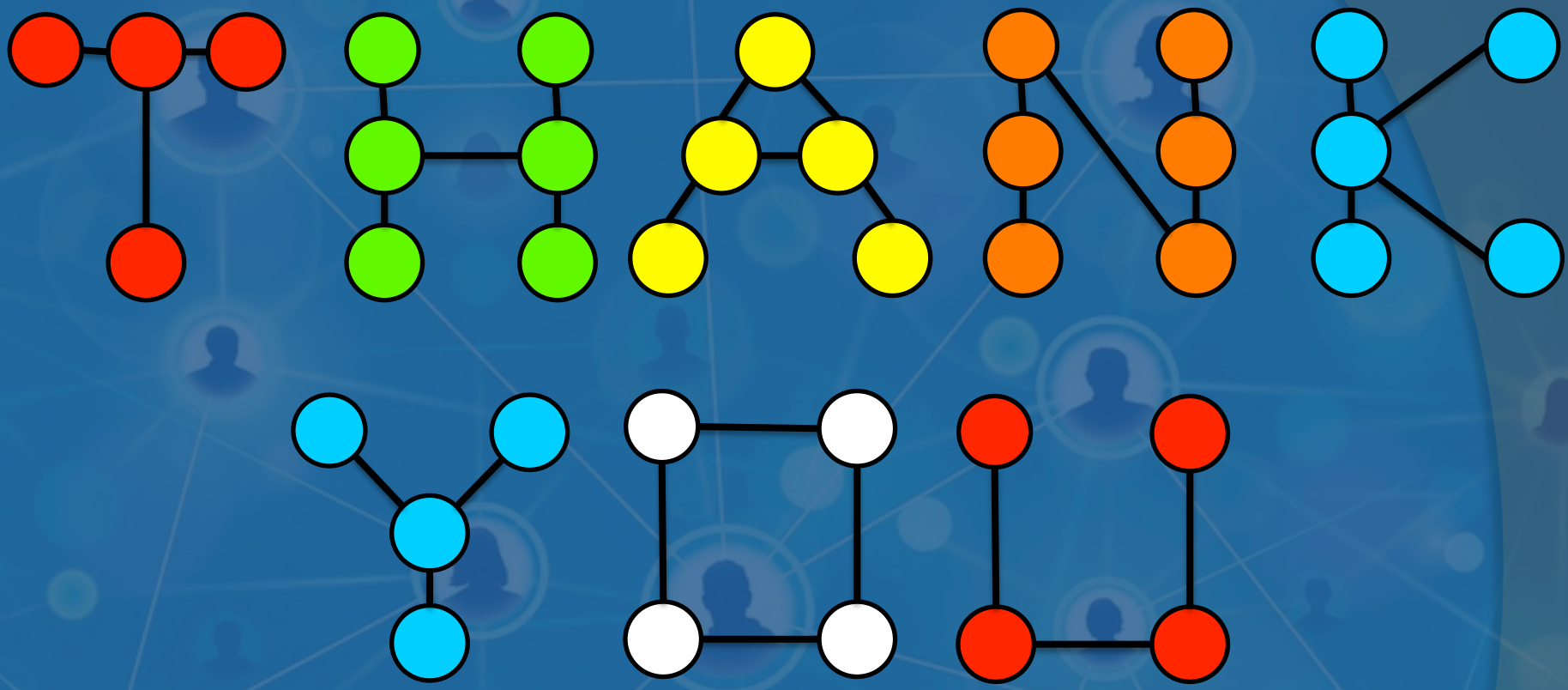
Experimental Results

Scale free graphs (128 labels)



Conclusions

- The improved VF3 is considerably faster then the original algorithm
- On both biological graphs and social network graphs it keeps a good performance (even when it is not the best, it is close to the best)
- It has a higher percentage of matchings completed before the timeout



MIVIA

QUESTIONS?