Solving Optimization and Competitive Diffusion Problems in Social Networks

Matthias Broecheler, Andrea Pugliese, Maria-Luisa Sapino, Paulo Shakarian & V.S. Subrahmanian

University of Maryland

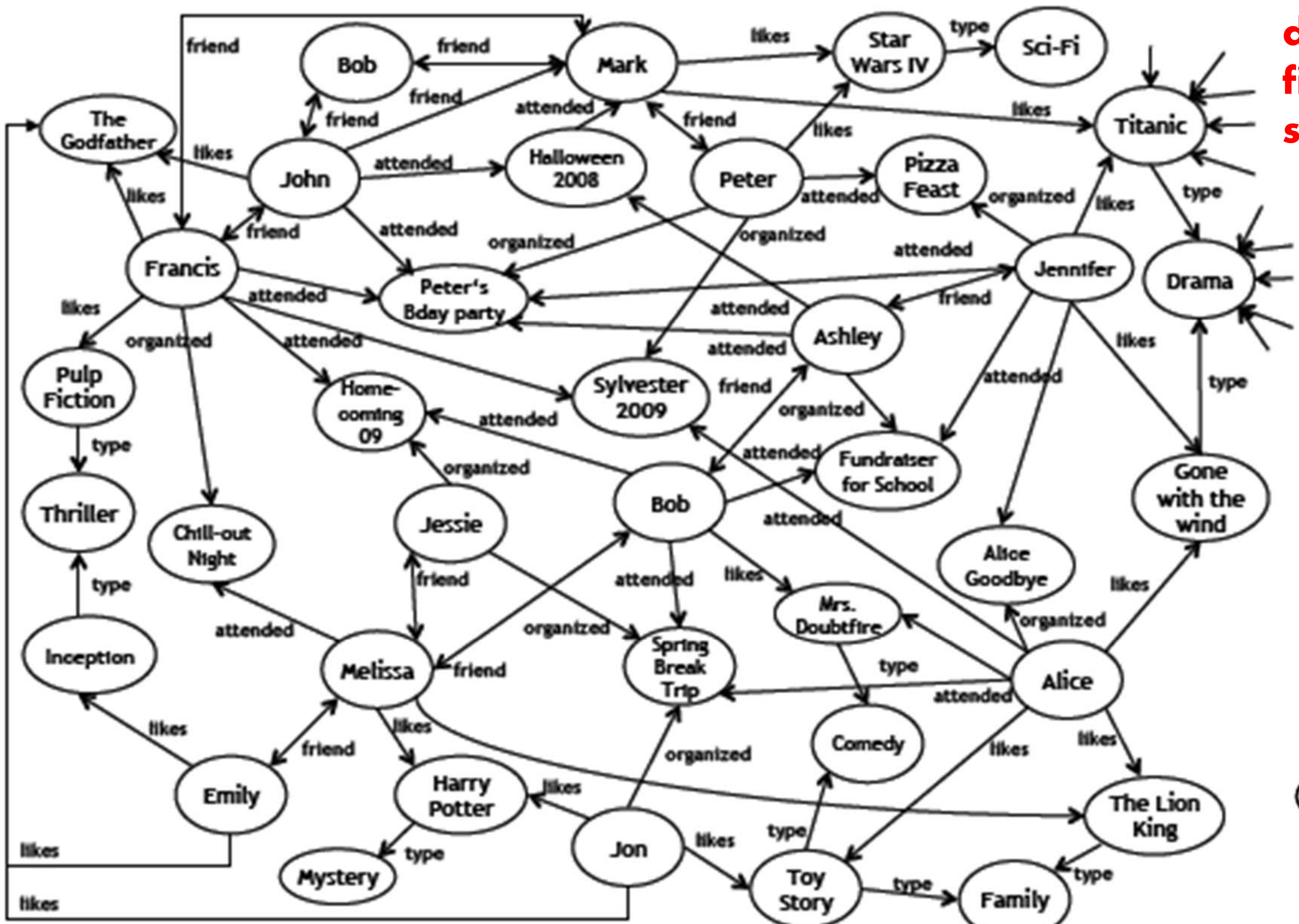
Outline

- Fast subgraph matching
- Social network optimization problems
- Competitive diffusion problems

500 million users

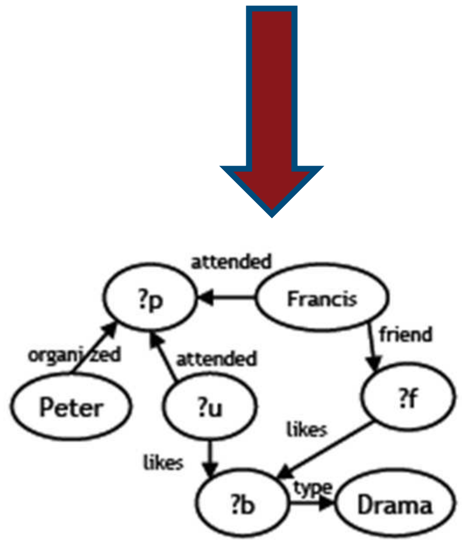
50M tweets / day

Fast subgraph matching



You have a graph database. You want to find all instances of some "pattern" in it.

query



DOGMA: Disk Subgraph Matching

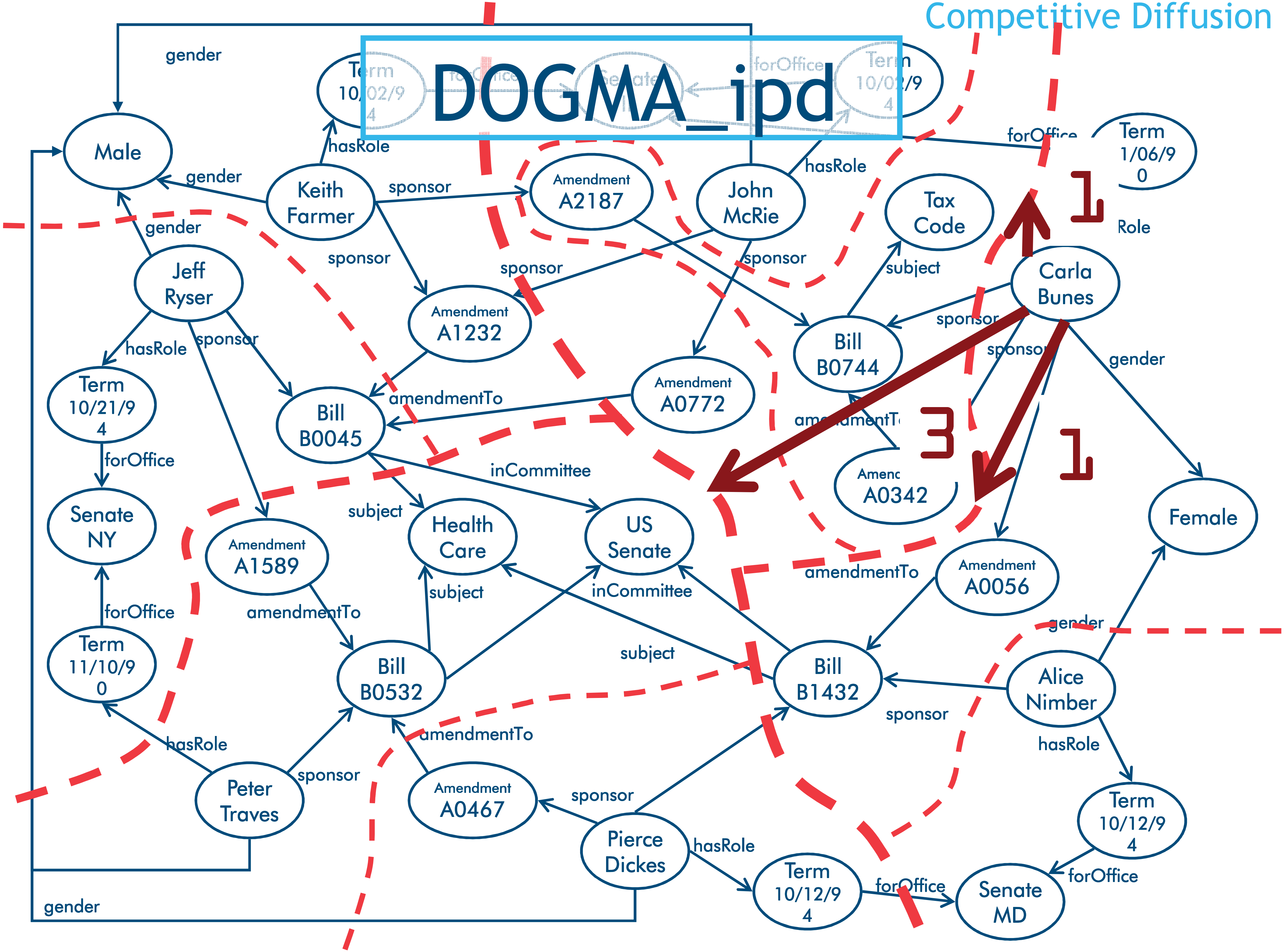
Iterative Coarsening

- Iteratively construct sequence $G_0, G_1, \ldots, G_n$ of graphs such that:
  - G_i has half as many vertices as G_{i-1}.
  - G_n fits on a disk page.
- When going from G_i to G_{i-1}, make sure you keep mappings describing which vertices in G_{i-1} are represented by a vertex in G_i.

Tree Construction

- Root of tree is G_n.
- For each unprocessed node:
  - Use a graph partitioning algorithm to split G_j into two parts LEFT and RT.
  - Expand LEFT and RT to double the number of vertices in each using the mappings.

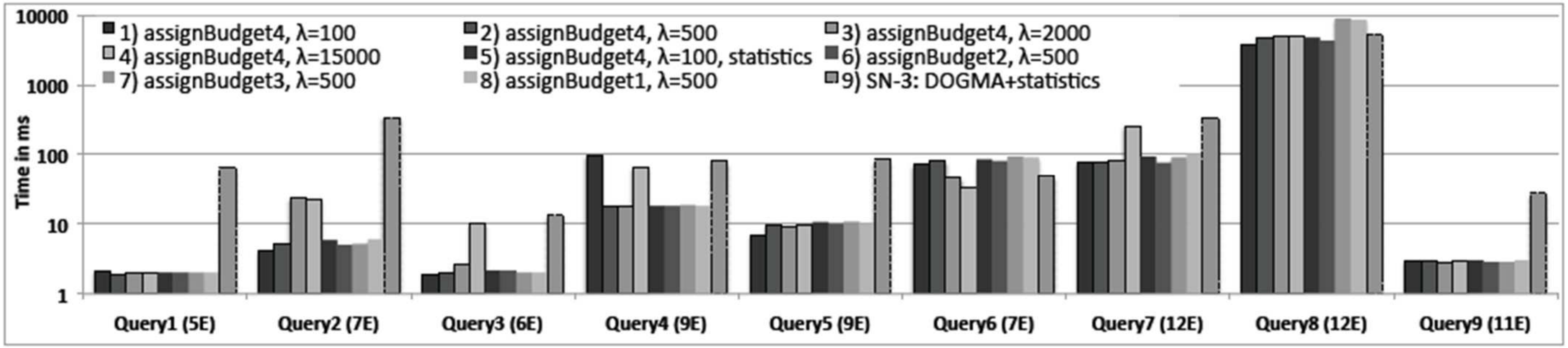
Competitive Diffusion



Fast subgraph matching

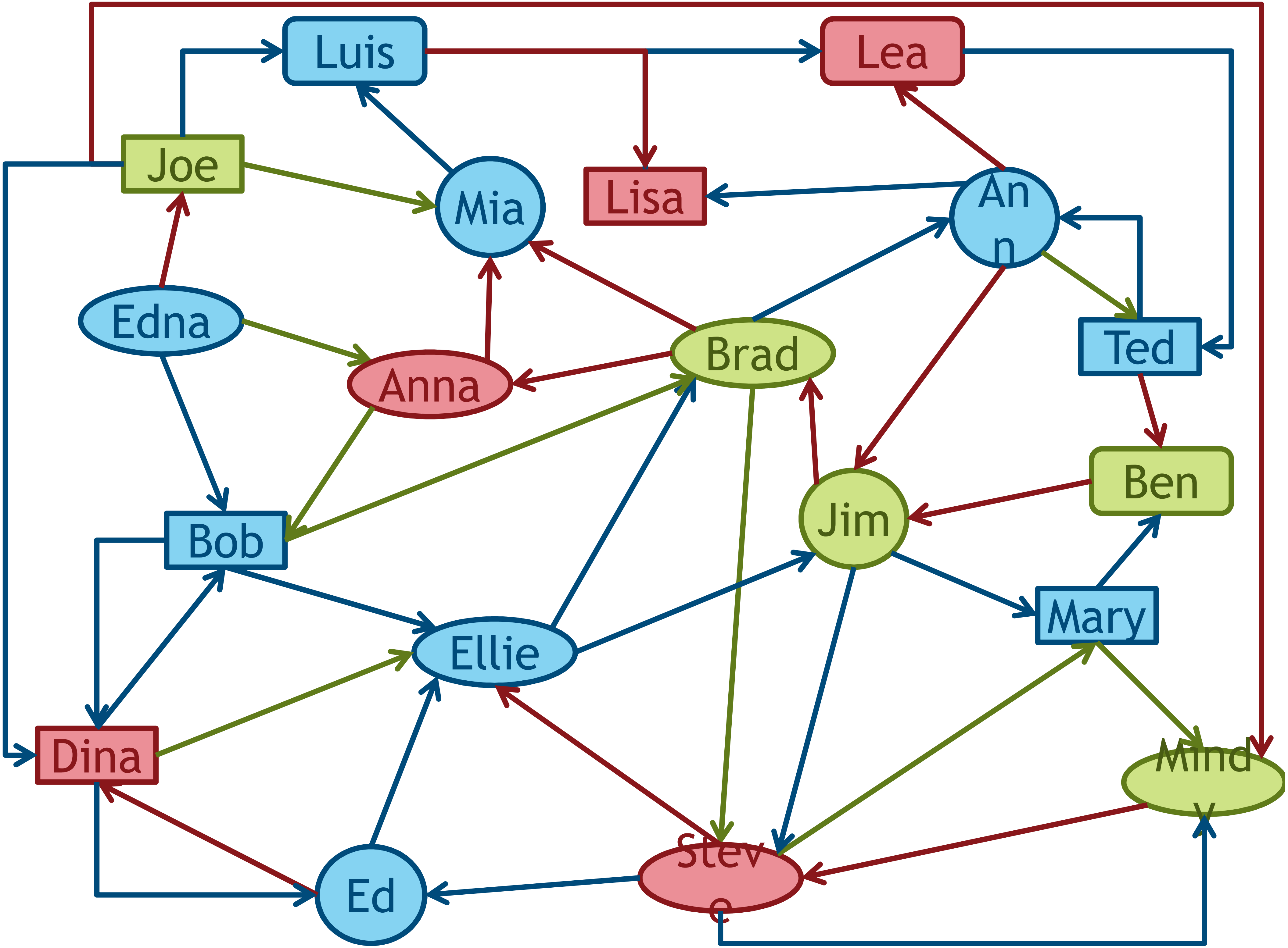
- Converted SN to a weighted graph.
- Our key theorem proved that the min-edge cuts of the weighted graph correspond to the best way of splitting the graph across a set of compute nodes.
- In 2010, our COSI system used this theorem to use a 16-node cloud to do the subgraph matching in under a second on a 1B+ edge Delicious data set. Confirmed with a second 778M edge dataset from Flickr, Orkut, LiveJournal.

• More recent BudgetMatch algorithm does this on a single machine in under a second – assigns budget to each query vertex !
• Planned extension to $100B+ edge dataset.
• More recent PMATCH algorithm solves "probabilistic" subgraph matches (where user does not know exactly what he is looking for) in < 1 second on two 1B+ edge data sets – one from Delicious, one from FaceBook.

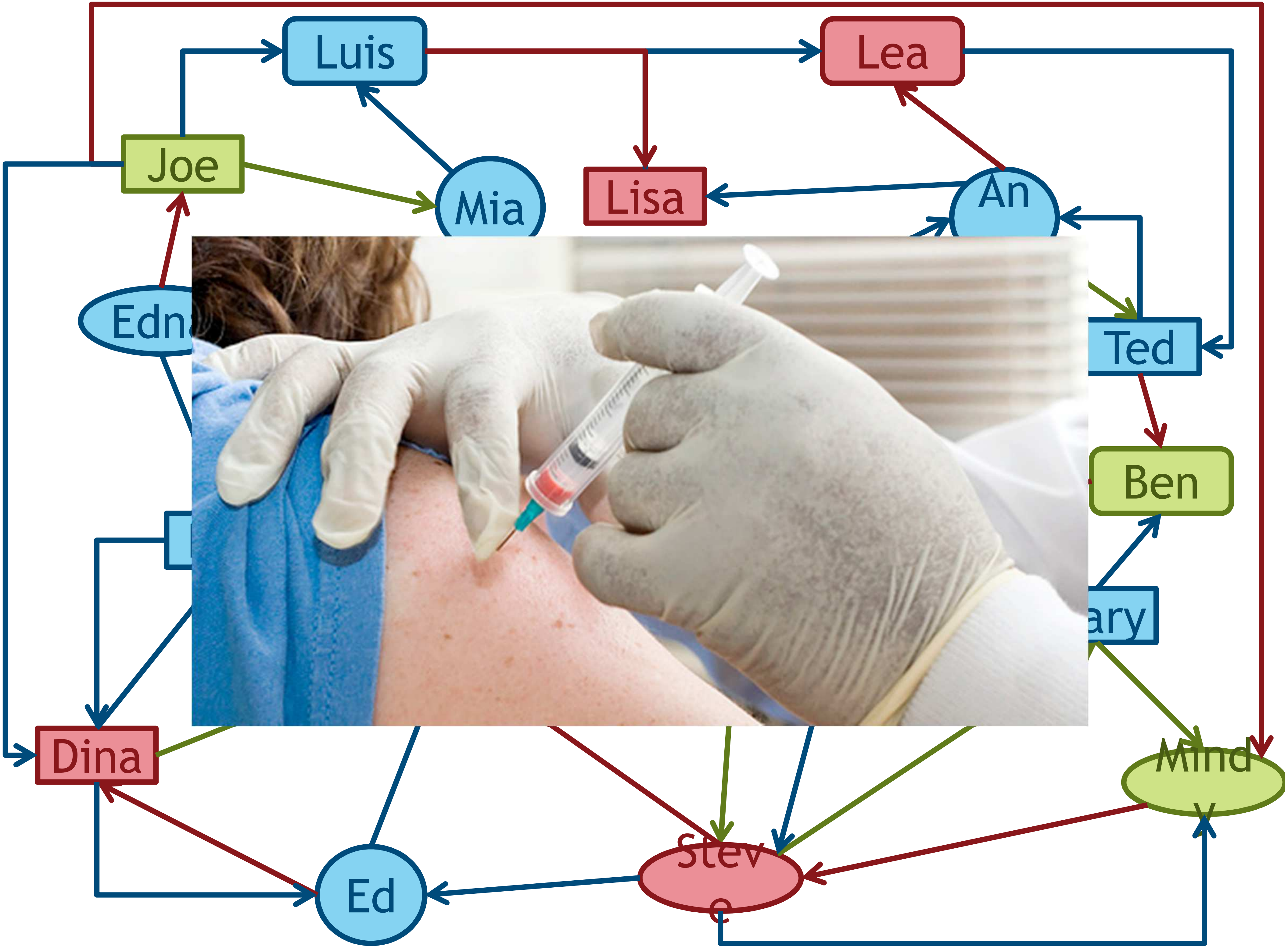


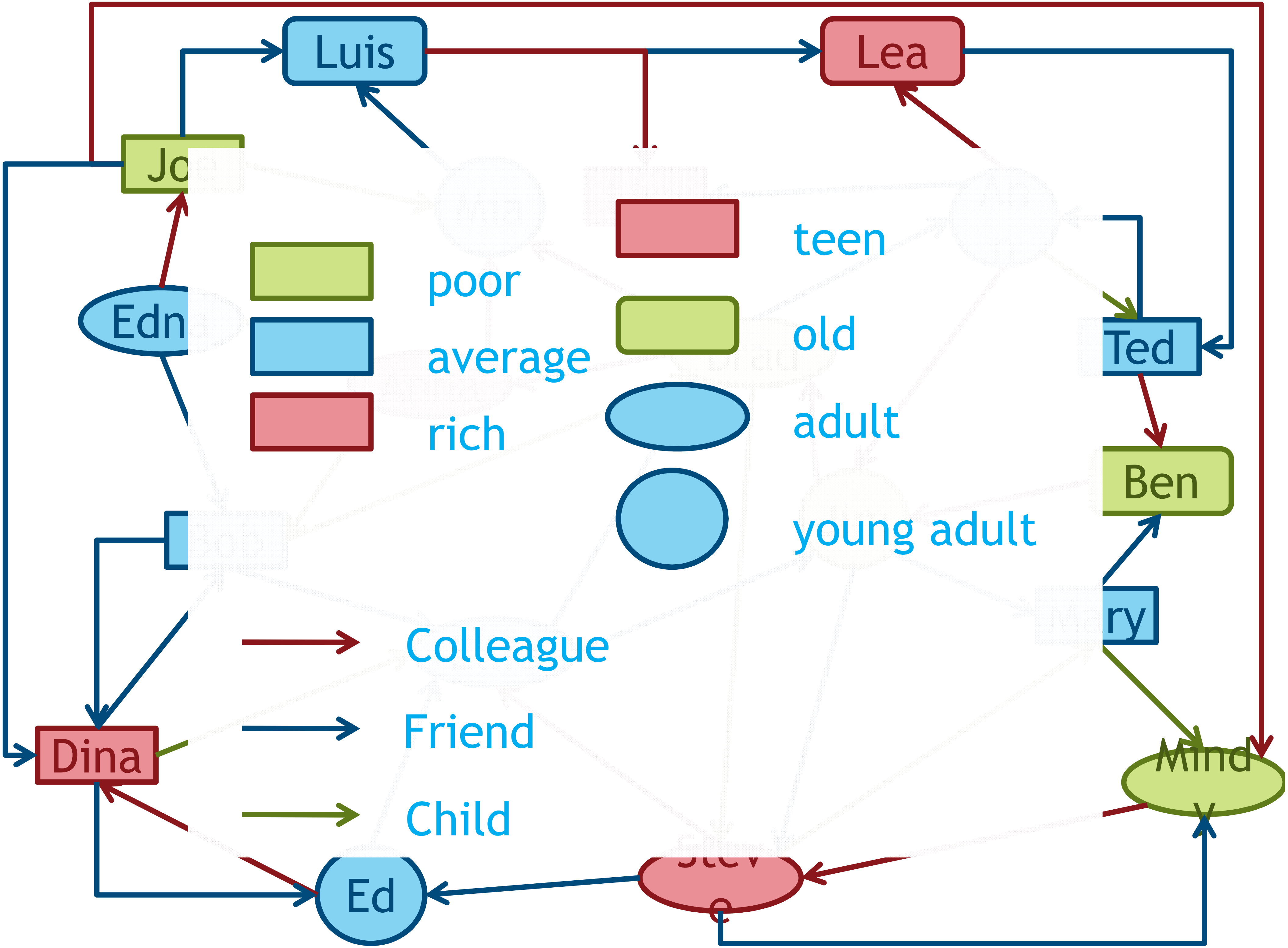
Outline

- Fast subgraph matching
- Social network optimization problems
- Competitive diffusion problems

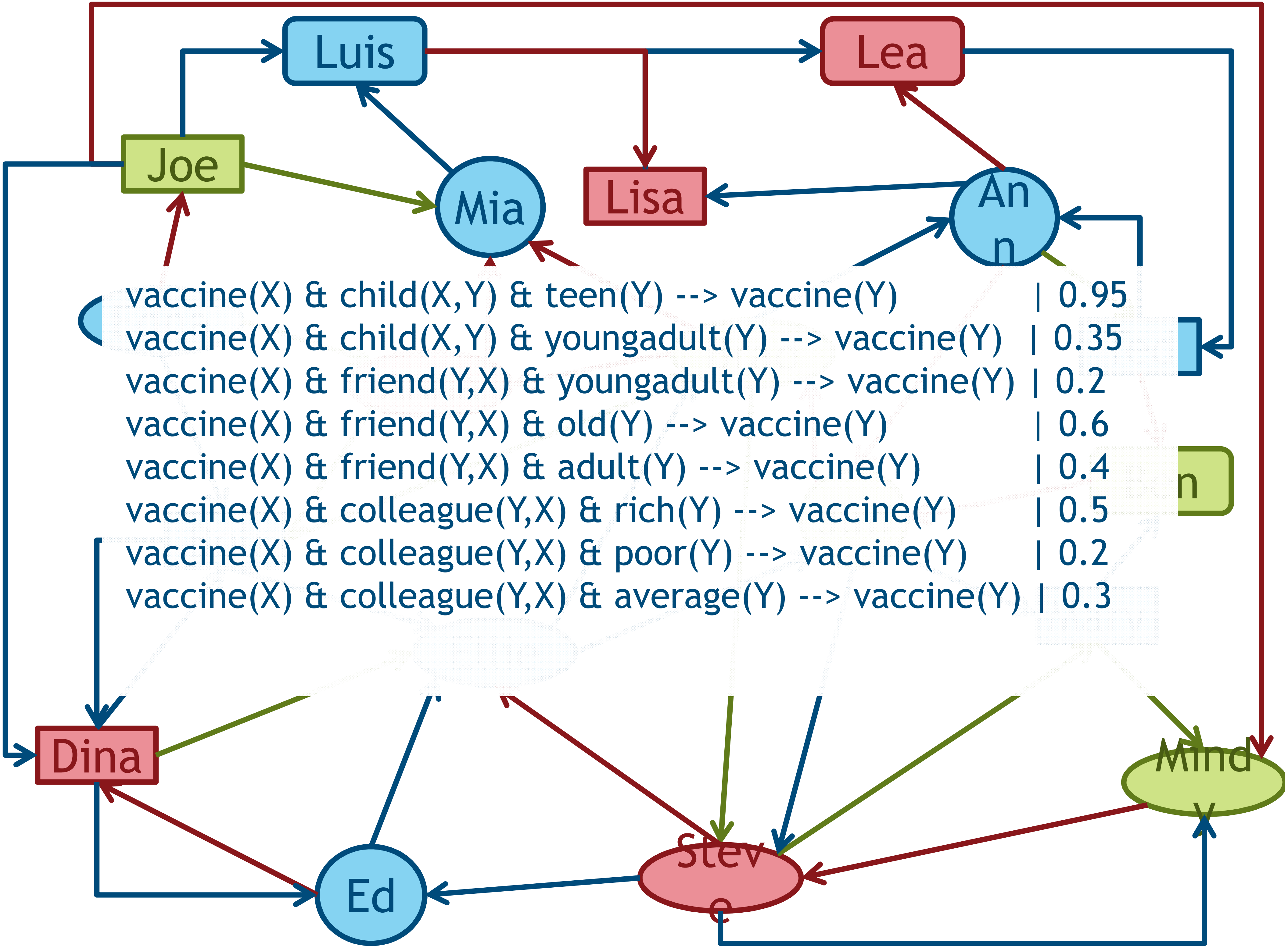


Luis
Lea
Joe
Mia
Lisa
An
n
Edna
Anna
Brad
Ted
Ben
Jim
Bob
Mary
Ellie
Mind
y
Dina
Stev
e
Ed

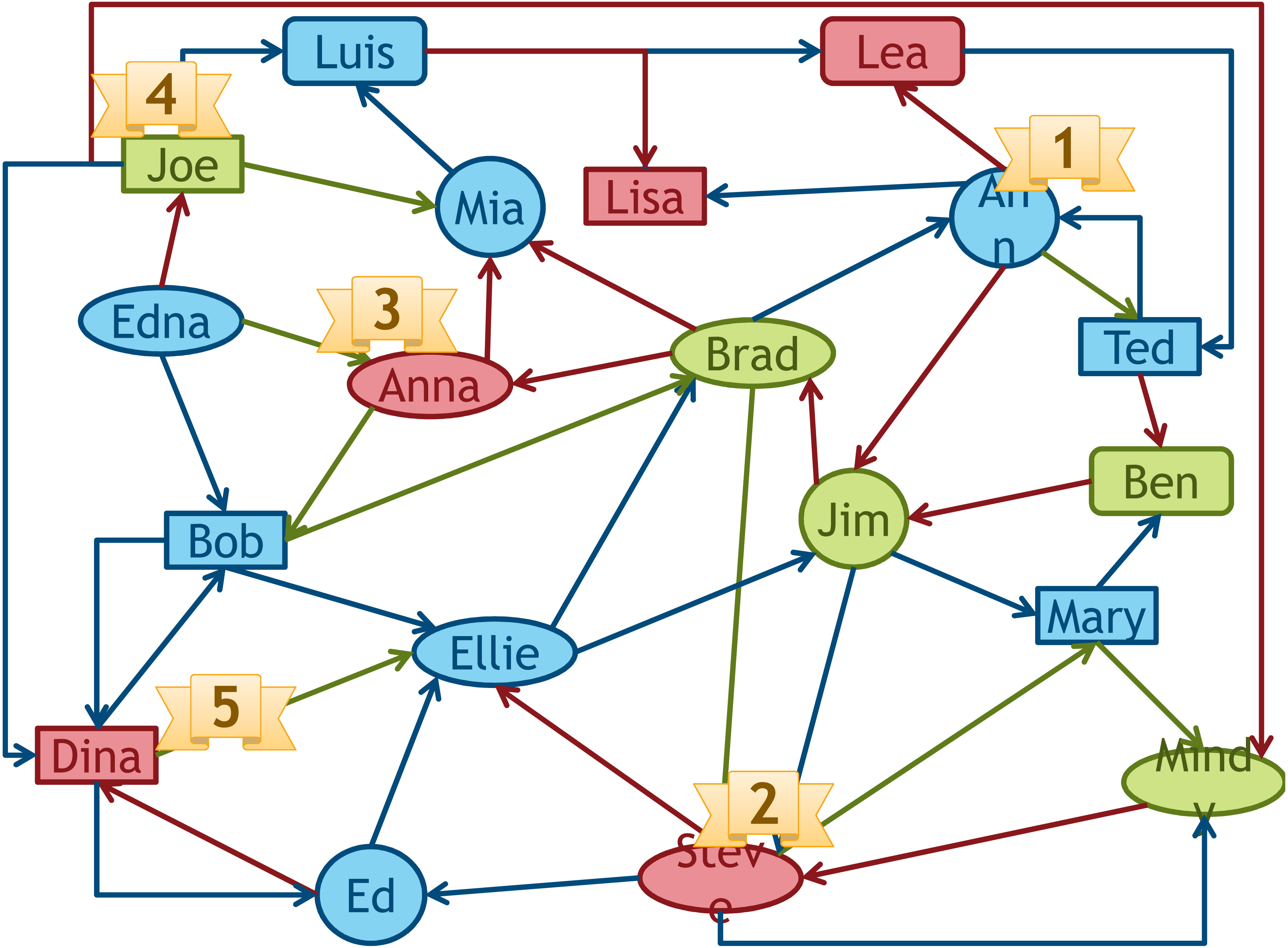

Luis
Lea
Joe
Mia
Lisa
An
Edn
Ted
Ben
ary
Dina
Mind
y
Stev
e
Ed

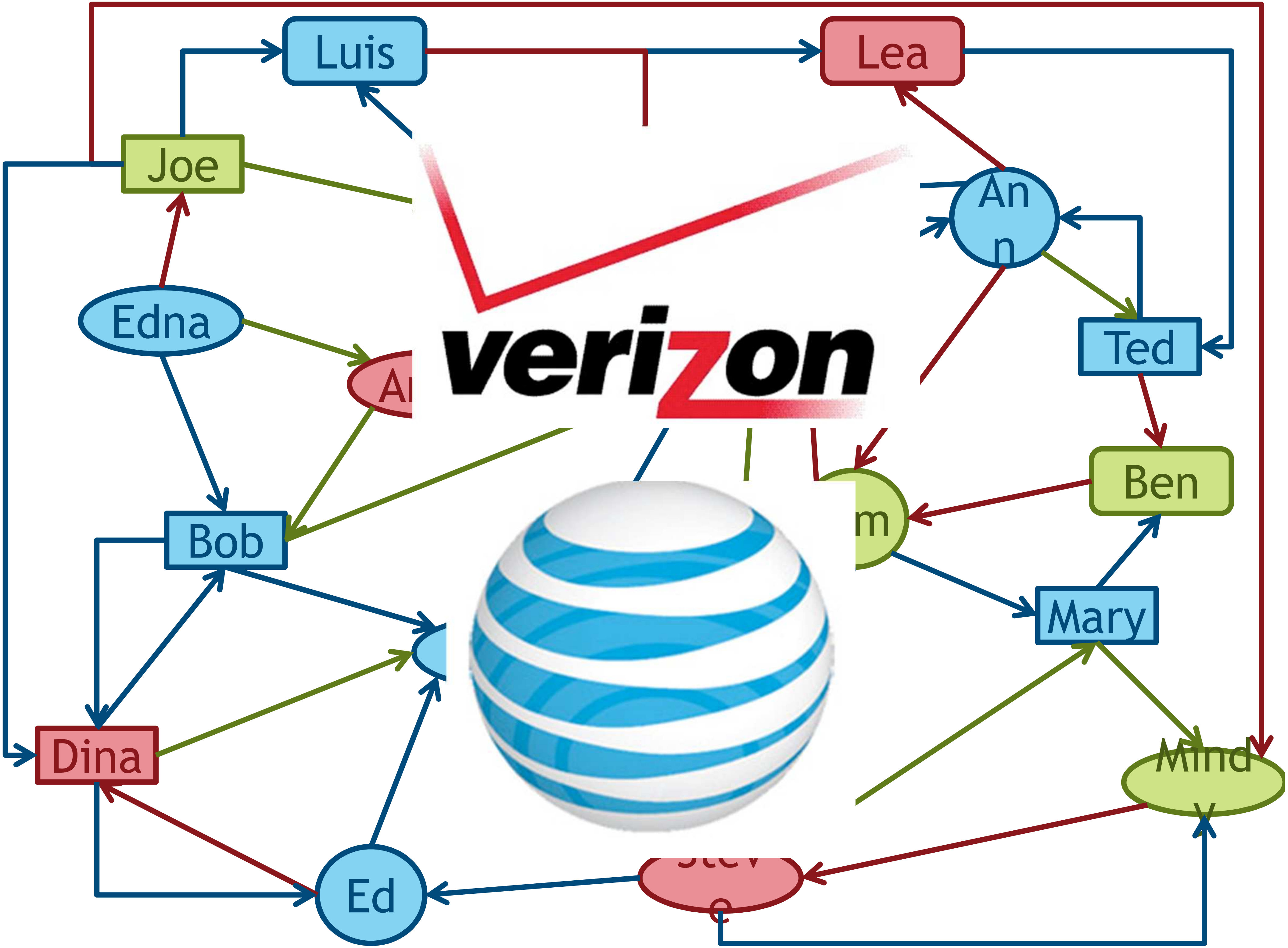


Luis
Lea
Ted
Ben
Mind
y
Dina
Ed
poor
average
rich
teen
old
adult
young adult
Colleague
Friend
Child

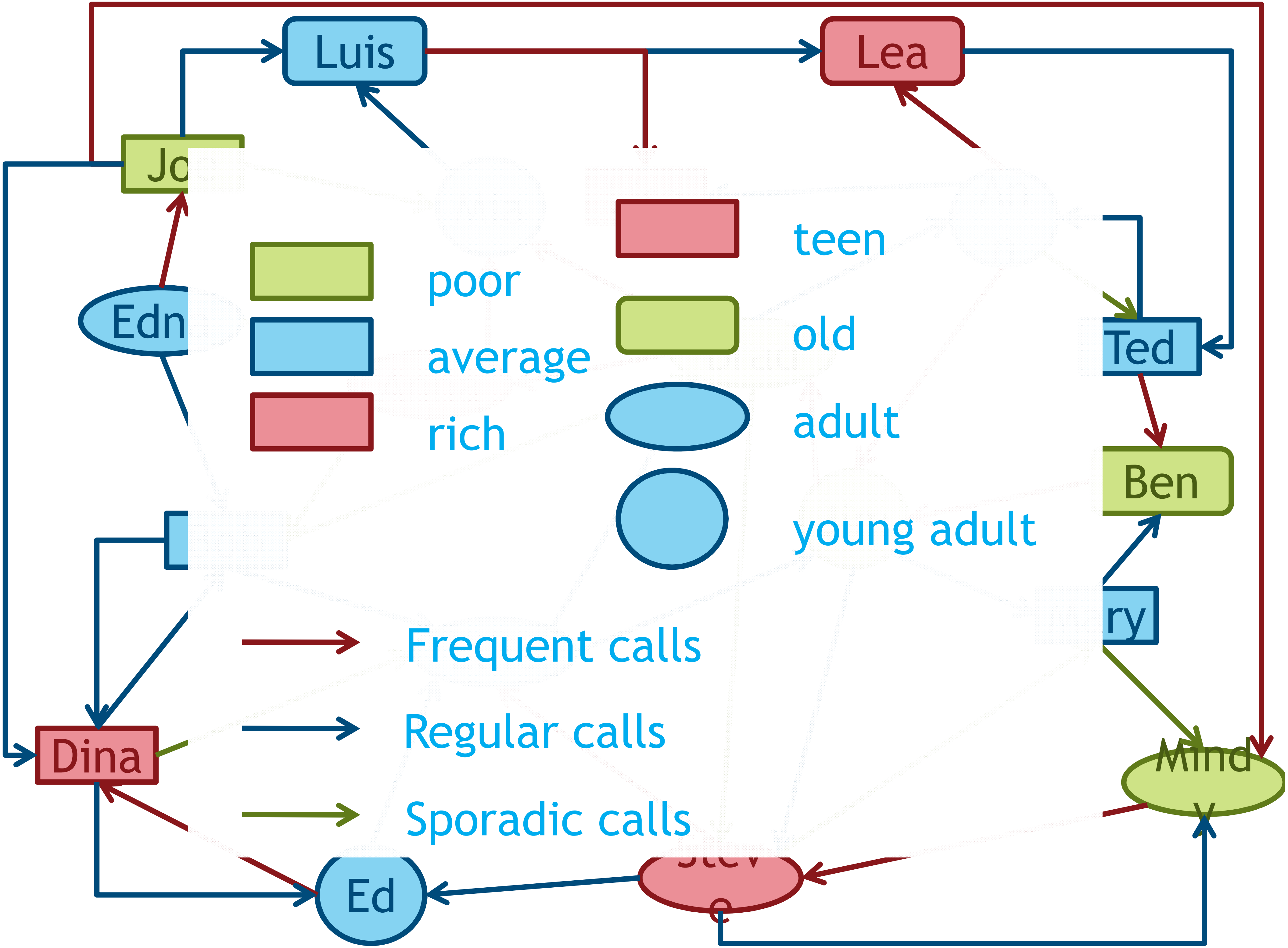


vaccine(X) & child(X,Y) & teen(Y) --> vaccine(Y) | 0.95
vaccine(X) & child(X,Y) & youngadult(Y) --> vaccine(Y) | 0.35
vaccine(X) & friend(Y,X) & youngadult(Y) --> vaccine(Y) | 0.2
vaccine(X) & friend(Y,X) & old(Y) --> vaccine(Y) | 0.6
vaccine(X) & friend(Y,X) & adult(Y) --> vaccine(Y) | 0.4
vaccine(X) & colleague(Y,X) & rich(Y) --> vaccine(Y) | 0.5
vaccine(X) & colleague(Y,X) & poor(Y) --> vaccine(Y) | 0.2
vaccine(X) & colleague(Y,X) & average(Y) --> vaccine(Y) | 0.3

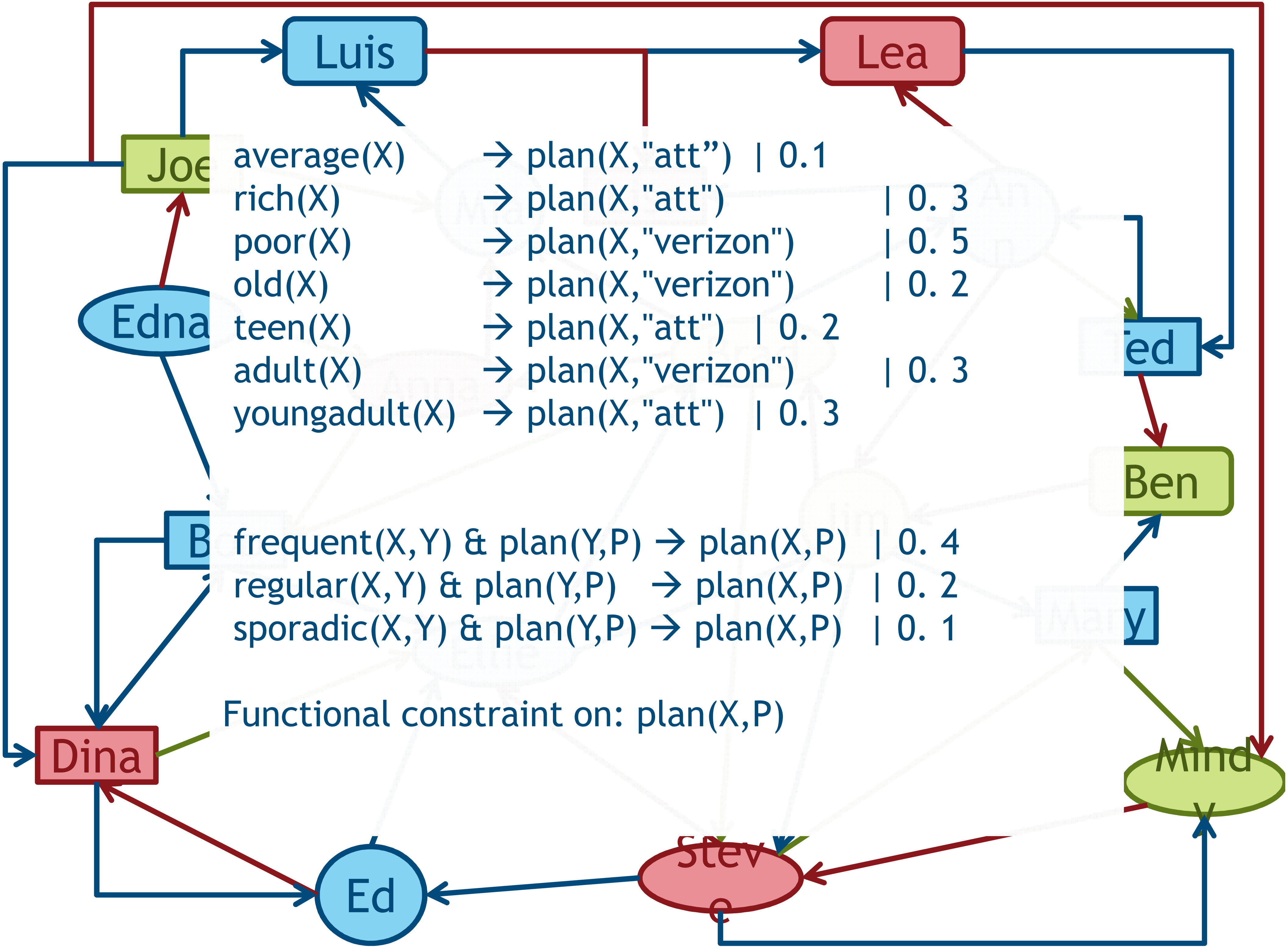

Luis
Lea
4
Joe
Mia
Lisa
1
Ann
Edna
3
Brad
Ted
Anna
Ben
Jim
Bob
Mary
Ellie
5
Dina
Mindy
2
Steve
Ed



Luis
Lea
Joe
An
n
Edna
Ted
Ben
Bob
m
Mary
Dina
Mind
y
Ed
verizon



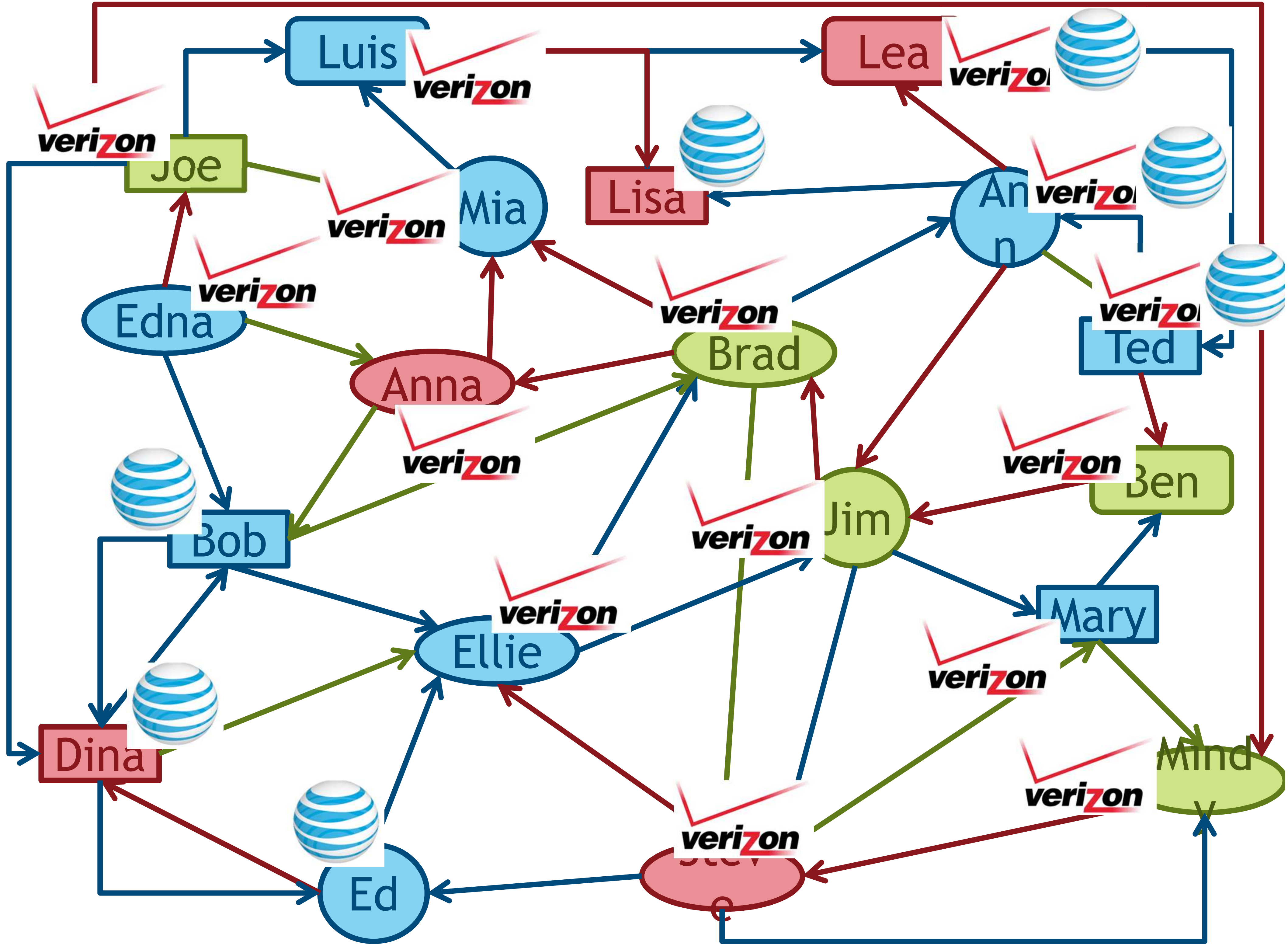
Luis
Lea
Edn
Ted
Ben
Dina
Ed
Mind
y
poor
average
rich
teen
old
adult
young adult
Frequent calls
Regular calls
Sporadic calls



average(X) → plan(X,"att”) | 0.1
rich(X) → plan(X,"att") | 0. 3
poor(X) → plan(X,"verizon") | 0. 5
old(X) → plan(X,"verizon") | 0. 2
teen(X) → plan(X,"att") | 0. 2
adult(X) → plan(X,"verizon") | 0. 3
youngadult(X) → plan(X,"att") | 0. 3

frequent(X,Y) & plan(Y,P) → plan(X,P) | 0. 4
regular(X,Y) & plan(Y,P) → plan(X,P) | 0. 2
sporadic(X,Y) & plan(Y,P) → plan(X,P) | 0. 1

Functional constraint on: plan(X,P)


Luis
Lea
Lisa
Joe
Mia
An
n
Edna
Anna
Brad
Ted
Ben
Bob
Jim
Mary
Ellie
Dina
Mind
y
Ed
e

Technical Preliminaries

- Set **V** – the set of vertices in the network

- Two types of predicates:
  - **VP** – *Vertex Predicates*
    - Unary predicates that specify attributes of a vertex
    - **Vertex atoms**: atoms consisting of a predicate in **VP** and a vertex
      - i.e. ***attribute(v)***
  - **EP** – *Edge Predicates*
    - Binary predicates that specify attributes of an edge
    - **Edge atoms**: atoms consisting of a predicate in **EP** and two vertices
      - i.e. ***ep(v,v')***

Technical Preliminaries

A **social network** is a 5-tuple:

(1) $\mathbf{V}$ *is a set whose elements are called vertices.*

(2) $\mathbf{E} \subset \mathbf{V} \times \mathbf{V}$ *is a multi-set whose elements are called edges.*

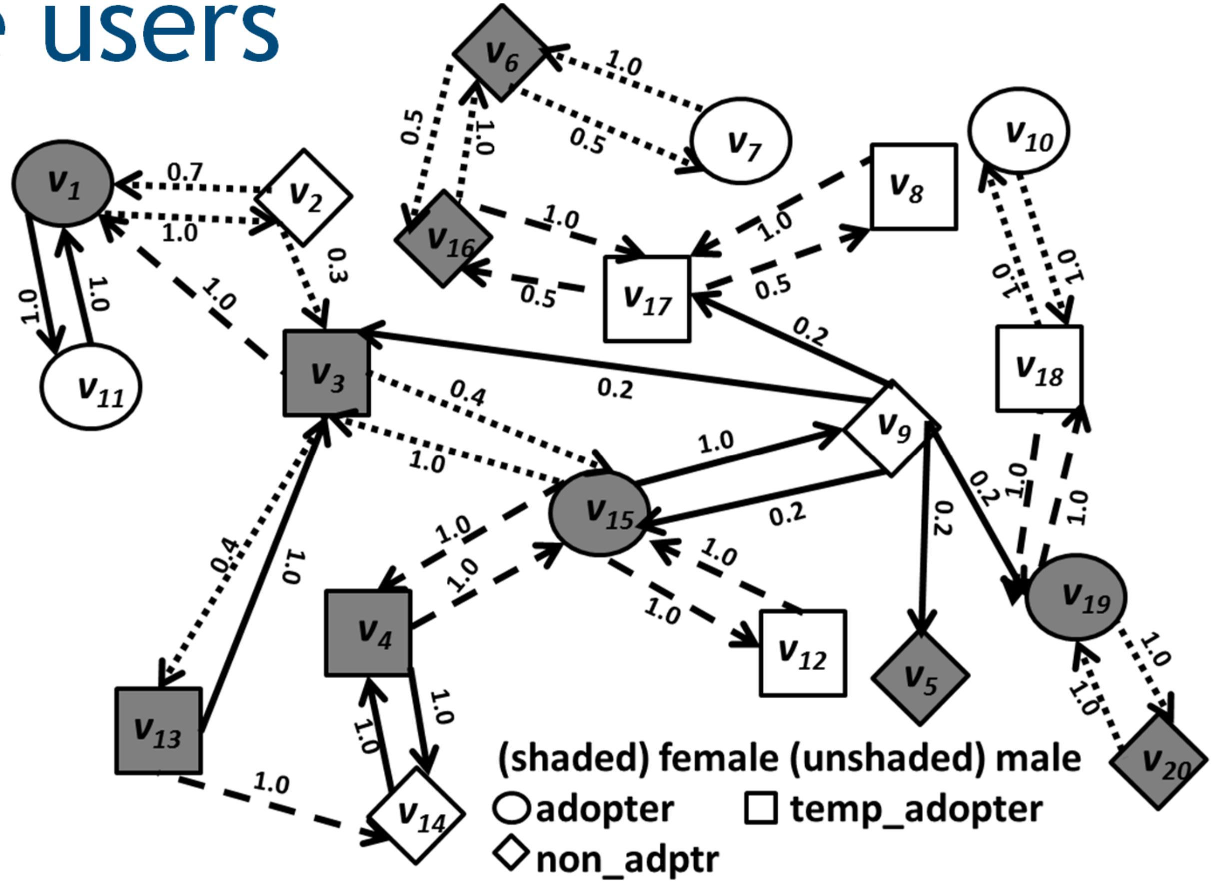
(3) $\ell_{vert} : \mathbf{V} \rightarrow 2^{VP}$ *is a function, called* vertex labeling function.

(4) $\ell_{edge} : \mathbf{E} \rightarrow EP$ *is a function, called* edge labeling function.

(5) $w : \mathbf{E} \times EP \rightarrow [0,1]$ *is a function, called* weight function.

Cell Phone Example

Example social network of cell-phone users



dashed/dotted/solid edges are email/instant message/cell phone relationships (resp.)

Gen. Annotated Programs

- Annotated term:
  - Variable symbol, number in [0,1], or function over [0,1] where the arguments are annotated terms
- Annotated atom:
  - If ***x*** is an annotated term and ***A*** is a ground atom (i.e. a vertex or edge atom) then ***A:x*** is an annotated atom
- Annotated rule:
  - Rule of the following form:

$$A_0 : \mu_0 \leftarrow A_1 : \mu_1 \wedge \ldots \wedge A_n : \mu_n$$

- Annotated program:
  - Set of annotated rules

Kifer & VS, 1989, 1992.

SNs can be embedded in GAPs

Every social network can be embedded into an annotated program:

- For all $v \in \boldsymbol{V}$, add:
  - $vert\_pred(v){:}1 \leftarrow TRUE \mid where\ vert\_pred \in \ell_{vert}(v)$
- For all $(v,v') \in \boldsymbol{E}$, add:
  - $edge\_pred(v,v'){:}w(v,v',edge\_pred) \leftarrow TRUE \mid where\ \ell_{edge}(v,v') = edge\_pred$

An Example

In addition to a social network, we can embed network diffusion rules in an annotated program as well

(1) $will\_adopt(V) : 0.8 \times X + 0.2 \leftarrow adopter(V) : 1 \wedge male(V) : 1 \wedge IM(V, V') : 0.3 \wedge female(V') : 1 \wedge will\_adopt(V') : X.$

(2) $will\_adopt(V) : 0.9 \times X + 0.1 \leftarrow adopter(V) : 1 \wedge male(V) : 1 \wedge IM(V, V') : 0.3 \wedge male(V') : 1 \wedge will\_adopt(V') : X.$

(3) $will\_adopt(V) : 1 \leftarrow temp\_adopter(V) : 1 \wedge male(V) : 1 \wedge email(V', V) : 1 \wedge female(V') : 1 \wedge will\_adopt(V') : 1.$

Rule (1) says that if V is a male adopter and V' is female and the weight of V's instant messages to V' is 0.3 or more, and we previously thought that V would be an adopter with confidence X, then we can infer that V will adopt the new plan with confidence $0.8 \times X + 0.2$. The other rules may be similarly read.

Linear GAPs

- We say an annotated program is "**linear**" if each ground rule is of the following form:

$$pred(V) : c_0 + c_1 \cdot X_1 + \ldots + c_i \cdot X_i + \ldots + c_n \cdot X_n \leftarrow \bigwedge_{A_i \in \mathcal{A}} A_i : X_i$$

where A is the set of all ground atoms, each X_i is a variable symbol, and $\Sigma c_i \in [0,1]$

Semantics of Annotated Programs

- An interpretation, ***I***, is simply a mapping of ground atoms to *[0,1]*
- An interpretation, ***I***, *satisfies a rule*

  $A{:}\mu \leftarrow AA_1 \wedge \ldots \wedge AA_n$

  iff $\mu \leq I(A)$ or for some $i \in [1,n]$, ***I*** does not satisfy AA_i
- An annotated program entails an annotated atom iff for every interpretation satisfying all rules in the program, that interpretation also satisfies the annotated atom

The Fixed-Point Operator

- The ***T*** operator maps interpretations to interpretations, wrt annotated program Π and is defined as follows:
  - $T_{\Pi}(I)(A)$ = ***sup*** $\{ \mu \mid A{:}\mu \leftarrow AA_1 \wedge \ldots \wedge AA_n$ is a ground rule in Π and for *all* $i \in [1,n]$, $I \models AA_i \}$
- **Theorem (Kifer '92):** The ***T*** operator is monotonic and has a least fixed point ($lfp(T_{\Pi}$ $))$s.t. Π entails $A{:}\mu$ iff $\mu \leq lfp(T_{\Pi})(A)$
- **Hence, for an annotated program consisting of a social network and diffusion rules, the least fixed point of *T* coincides with the maximum extent of the diffusion.**

Aggregates and Vertex Conditions

- An **aggregate** is simply a mapping of all finite multisets of real numbers in [0,1] to a real number
  - SUM, COUNT, AVG are all examples of aggregates

- A **Vertex Condition** is a conjunction of annotated vertex atoms containing exactly one variable. A vertex condition can be specified in two ways:
  - **A-Priori:** a condition enforced before diffusion occurs (only on the embedding of the social network)
    - In the remainder of this presentation, we shall assume an a-priori vertex condition
  - **A-Posteriori:** enforced after diffusion occurs

SNOP Query

- A **SNOP query** is a 4-tuple:
  - *agg*: an aggregate
  - *VC*: vertex condition
  - *k*: natural number *>0*
  - *g(V)*: goal atom (a non-ground atom, *g* is one of the vertex predicates)

- For a given SNOP query, we define a **pre-answer** as a set of vertices **V'** $\subseteq$ **V** s.t.
  - $|V'| \leq k$
  - For all *v'* $\in$ ***V'***
    - *{g(v'):1 | v' in **V'**}* U (the embedding of the social network + logic rules) entail *VC*

SNOP Query

For a given SNOP query and pre-answer, ***V'***, we define ***value(V')*** to be a real number defined as follows:

- $agg(\{\, lfp(T_{\Pi \cup \{g(v'):1 \leftarrow TRUE \mid v' \in V'\}})(g(V) \mid V \in \mathbf{V}\, \}\,)$
- In other words, ***value(V')*** is the aggregate of all annotations of goal atoms if every goal atom formed with a vertex in ***V'*** is annotated with ***1*** and the diffusion process completes

The Complexity of SNOP Queries

- An **answer** to a SNOP-query is a pre-answer **V'** $\subseteq$ **V** s.t. ***value(V')*** is maximized

- Theorem: Answering a SNOP-query is NP-hard.
  - Associated decision problem is NP-complete provided annotation and aggregate functions are computable in PTIME
  - NP-hardness shown by a reduction from MAX-K-COVER
  - NP-hardness holds even if:
    - The annotated program is linear
    - The aggregate is SUM
    - ***value(∅) = 0***

Limits of Approximation for SNOP Queries

Theorem: A SNOP query cannot be approximated in PTIME within ***(e-1)/e*** - ***b*** (where ***b > 0***) unless P=NP.

- Follows directly from the previous complexity result and non-approximation result of MAX-K-COVER
- Still holds under the following conditions
  - The annotated program is linear
  - The aggregate is SUM
  - ***value(∅) =0***

- ***Recall e=2.718, so (e-1)/e = 0.63.***

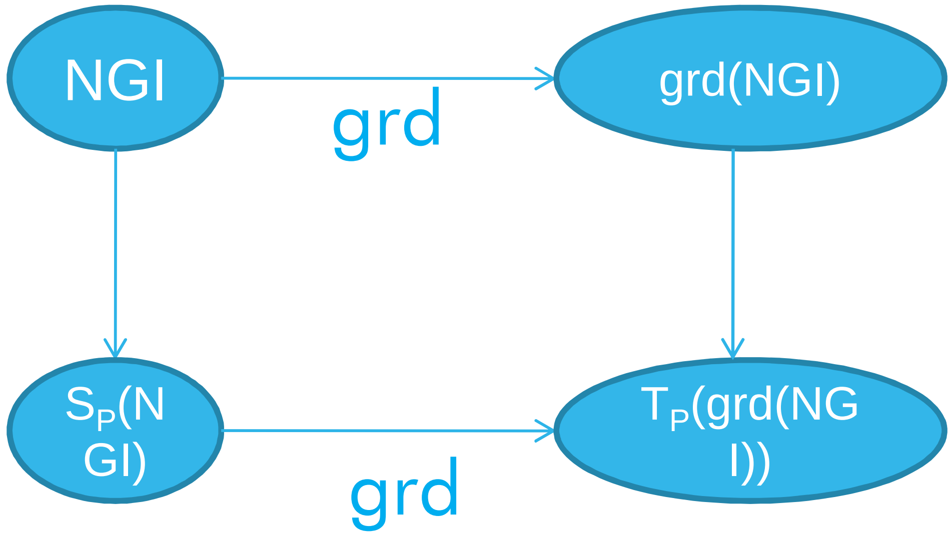
SNOP Queries and Submodularity

- **Theorem:** For a given SNOP-query, if the annotated program is linear, *VC* is applied a-priori, and *agg* is a positive, weighted sum, then ***value*** is a *sub-modular* function.
  - In other words, for sets **V' $\subseteq$ V''**, and ***v*** $\notin$ ***V''***, the following holds:

$$value(V' \cup \{v\}) - value(V') \geq value(V'' \cup \{v\}) - value(V'')$$

SNOPs: Key Approach

1. Given a GAP P, Kifer-Subrahmanian defined a fixpoint operator $T_P(I)(A)$ = LUB { μ | A: μ <= B_1: μ_1 & … & B_n: μ_n is a ground instance of a rule in P and $I(B_i) >= \mu_i$ for all i in {1,..,n} }.
2. Non-ground interpretation NGI maps atoms (not necessarily ground) to reals in the [0,1] interval.
3. This paper defines a non-ground fixpoint operator S_P such that $grd(S_P(NGI)) = T_P(grd(NGI))$.
4. Search algorithm to solve any SNOP based on the S_P operator.



SNOP-Mon Algorithm

SNOP-Mon$(\Pi, agg, VC, k, g(V))$

(1) The variable $Curr$ is a tuple consisting of a GAP and natural number. We initi
$Curr.Prog = \Pi$; $Curr.Count = 0$.

(2) $Todo$ is a set of tuples described in step 1. We initialize $Todo \equiv \{Curr\}$

(3) Initialize the real number $bestVal = 0$ and GAP $bestSOL = NIL$

(4) **while** $Todo \neq \emptyset$ **do**

  (a) $Cand$ = first member of $Todo$; $Todo = Todo - \{Cand\}$

  (b) **if** $value(lfp(\mathbf{S}_{Cand.Prog})) \geq bestVal \wedge lfp(\mathbf{S}_{Cand.Prog}) \models VC$ then

    i. $bestVal = value(lfp(\mathbf{S}_{Cand.Prog}))$; $bestSOL$ = GAP

  (c) **if** $Cand.Count < k$ **then**

    i. For each ground atom $g(V)\theta$, s.t. $\nexists OtherCand \in Todo$ where $OtherCand.Prog \supseteq Cand.Prog$, $|OtherCand.Prog| \leq |Cand.Prog| + 1$, and $lfp(\mathbf{S}_{OtherCand.Prog}) \models g(V)\theta : 1$, do the following:

      A. Create new tuple $NewCand$.
      Set $NewCand.Prog = Cand.Prog \cup \{g(V)\theta : 1 \leftarrow\}$.
      Set $New.Count = Cand.Count + 1$)

      B. Insert $NewCand$ into $Todo$

    ii. Sort the elements of $Element \in Todo$ in descending order o $value(Element.Prog)$, where the first element, $Top \in Todo$, has the greatest such value (i.e. there does not exist another element Top' s.t. $value(Top'.Prog) > value(Top.Prog)$)

(5) **if** $bestSOL \neq NIL$ then **return** $(bestSOL.Prog - \Pi)$ **else** return NIL.

Start with a GAP

Pick a GAP and see if it's an answer

Expand GAP by adding new atoms.

Process the highest value GAP next.

Greedy SNOP

GREEDY-SNOP$(\Pi, agg, VC, k, g(V))$ returns $SOL \subseteq \mathbf{V}$

(1) Initialize $SOL = \emptyset$ and $REM = \{v \in \mathbf{V} | \left(g(v) : 1 \wedge \bigwedge_{pred \in \ell_{vert}(v)} pred(v) : 1\right) \models VC[V/v]\}$

(2) While $|SOL| < k$ and $REM \neq \emptyset$

(a) $v_{best} = \text{null}$, $val = value(SOL)$, $inc^{(alg)} = 0$

(b) For each $v \in REM$, do the following

i. Let $inc^{(alg)}_{new} = value(SOL \cup \{v\}) - val$

ii. If $inc^{(alg)}_{new} \geq inc^{(alg)}$ then $inc^{(alg)} = inc^{(alg)}_{new}$ and

(c) $SOL = SOL \cup \{v_{best}\}$, $REM = REM - \{v_{best}\}$

(3) Return SOL

Find all *v* satisfying VC

Compute "marginal" diff for each possible *v*. *Submodularity used here.*

Expand SOL greedily with best *v*.

Greedy SNOP

- **Theorem.** Greedy SNOP runs in time **O**(k*|V|*F(|V|)) where F(|V|) is the time to compute *value(-)*.
- **Theorem.** When the GAP is linear, VC is a priori, agg is positive linear and value is zero-starting, then GREEDY-SNOP is an (e/e-1)-approximation algorithm for the query. (best possible unless P=NP)
- Developed several additional approximations and heuristics.

Three classes of diffusion models

- **Tipping models** (Granovetter, Schelling). Vertex adopts behavior based on number of neighbors that do.
- **Cascading models**. Vertex adopts behavior based on the strength of relationships with neighbors.
- **Homophilic models**. Vertex adopts behavior on the basis of similarity (in terms of properties) of other vertices.

Tipping: Jackson-Yariv Product Adoption Model

- Node v_i **switches to behavior** B iff $(b_i/c_i)*g(d_i)*p_i \geq 1$ where:
  - b_i is benefit to v_i to adopt behavior B.
  - c_i is cost to v_i to adopt behavior B.
  - p_i is the percentage of v_i's neighbors that adopted behavior B.
  - $g(d_i)$describes how the number of neighbors of v_i adopting behavior B affects benefits to v_i

$$B(V_i) : \lfloor \frac{b_i}{c_i} \cdot g(\sum_j E_j) \cdot \frac{\sum_j X_j}{\sum_j E_j} \rfloor \leftarrow \bigwedge_{V_j | (V_j, V_i) \in \mathbf{E}''} (edge(V_j, V_i) : E_j \wedge B(V_j) : X_j)$$

Cascade Model: SIR Model of Disease Spread

- SIR model says each vertex is either
  - *Susceptible* (not had the disease, but can get it)
  - *Infected* (has had the disease for less than t_{rec} time units)
  - *Removed* (vertex cannot catch or transmit the disease)
- An infected vertex *v* can infect a neighbor *v'* with a probability $P_{v,v'}$.
- GAPs can express the SIR model and many other models.

Cascade Model: SIR Model of Disease Spread

- SIR model says each vertex is either
  - *Susceptible* (not had the disease, but can get it)
  - *Infected* (has had the disease for less than t_{rec} time units)
  - *Removed* (vertex cannot catch or transmit the disease)
- An infected vertex *v* can infect a neighbor *v'* with a probability $P_{v,v'}$.
- GAPs can express the SIR model and many other models.

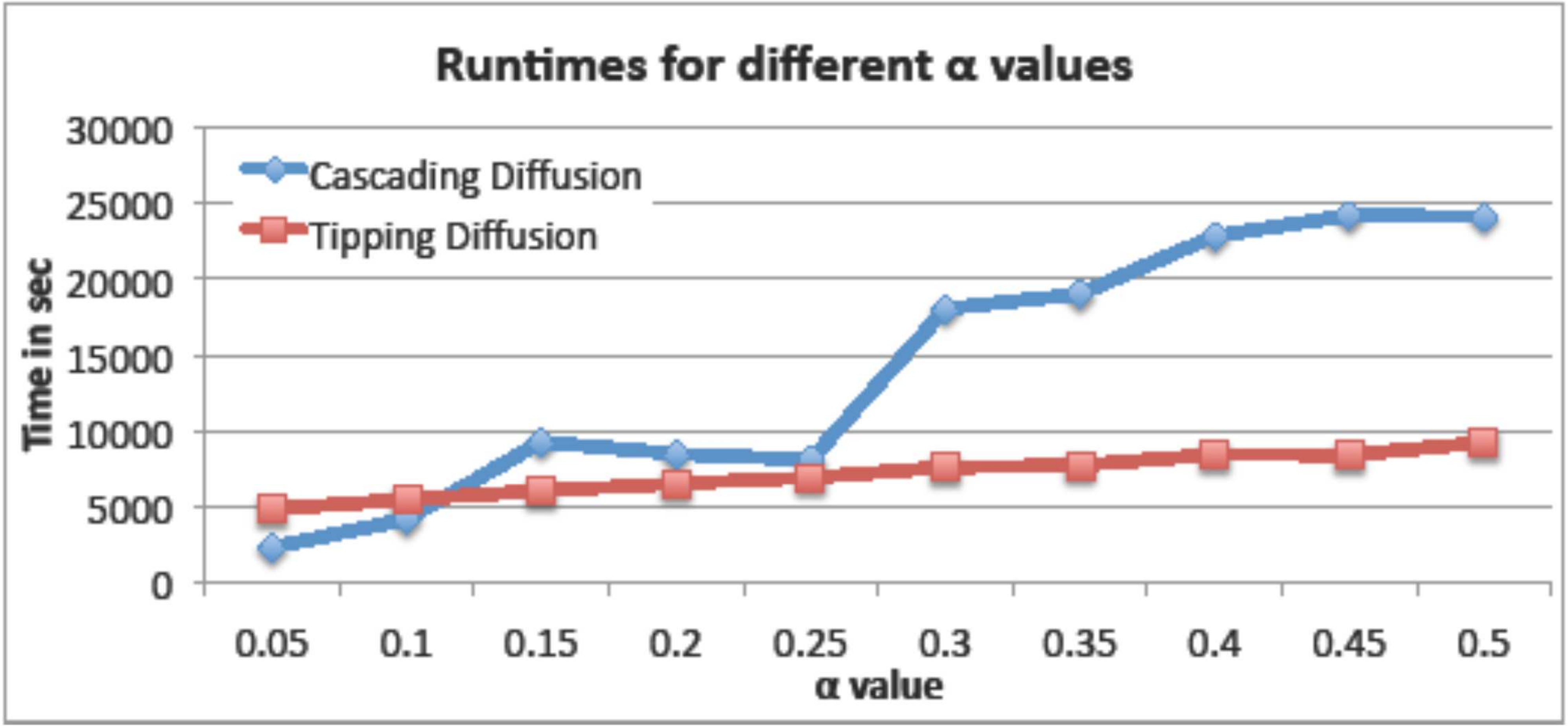
for each $i = \{2, \ldots, t_{rec}\}$ - *starting with* t_{rec}:

$$rec_i(V) : R \leftarrow rec_{i-1}(V) : R$$
$$rec_1(V) : R \leftarrow inf(V) : R$$
$$inf(V) : (1-R) \cdot P_{V',V} \cdot (P_{V'} - R') \leftarrow rec_{t_{rec}}(V) : R \wedge e(V', V) : P_{V',V} \wedge inf(V') : P_{V'} \wedge rec_{t_{rec}}(V') : R'.$$

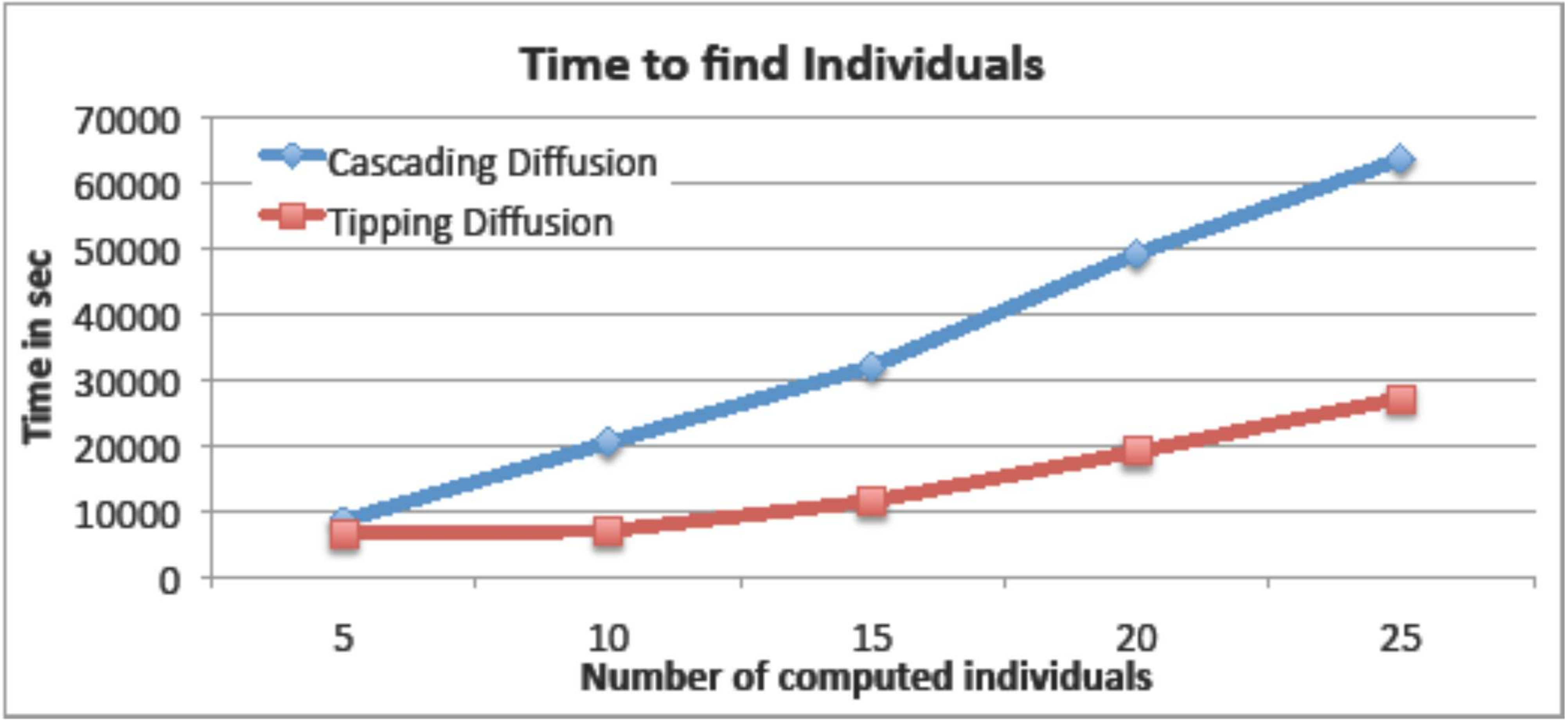
Experiments

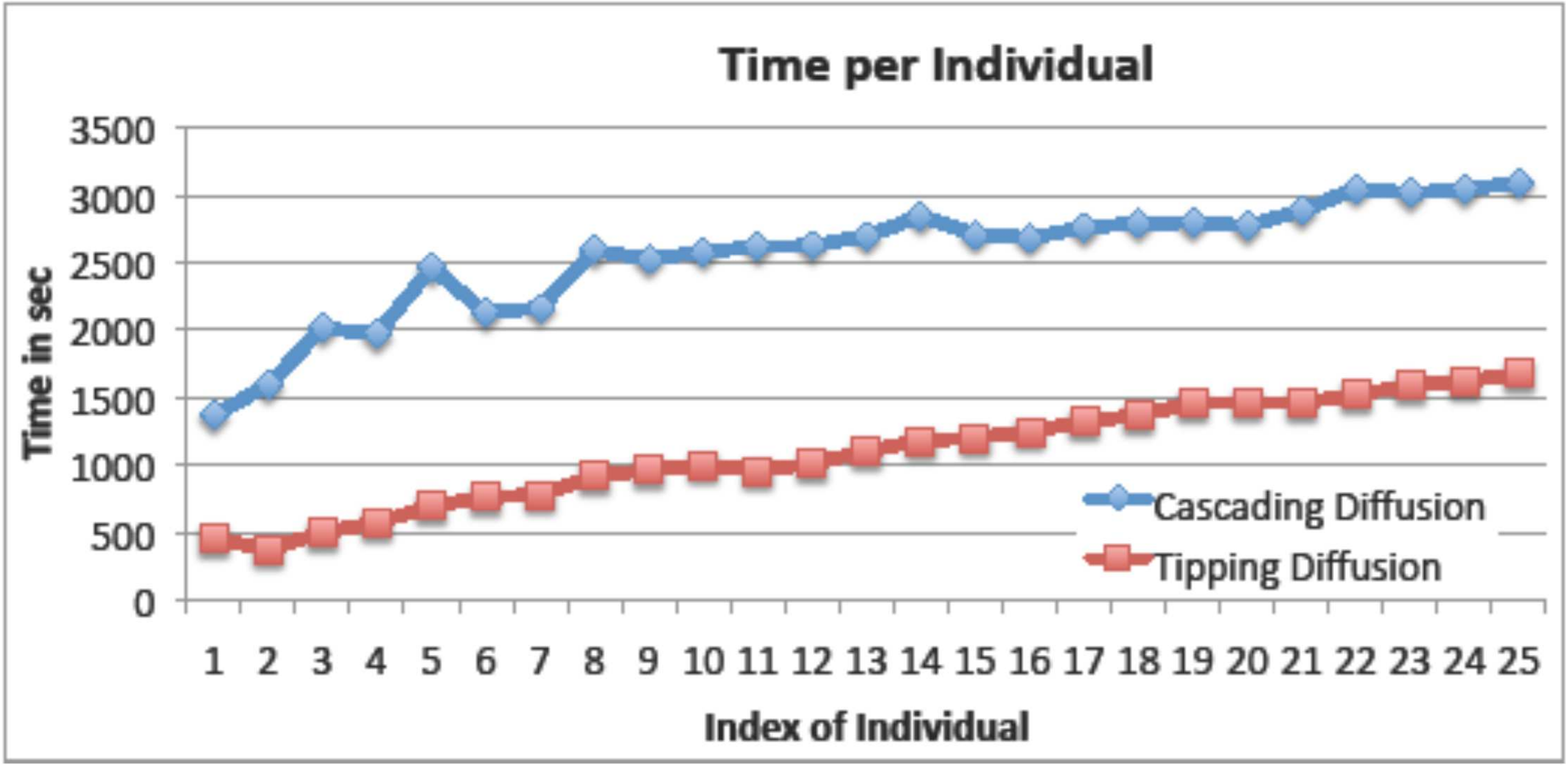
- Wikipedia allows admins and eligible users to vote for new admins.
- Social network consists of admins/eligible users.
- Edge from *j* to *i* if an admin/elig user *j* voted for an eligible user *i*.
- Looked at just under 2800 elections.
- SN has 7K nodes and over 103K edges.
- Parameter α specifies level of influence of a candidate on voters. Higher α => more influence.
- Queries tested looks to find set of *K* users who jointly wield the most influence (i.e. yield the highest expected number of influenced voters).
- Tipping model based on Flickr photo diffusion.
- Cascade model based on Jackson-Yariv.

Experiments



α specifies level of influence of a candidate on voters.
Higher α => more influence.


Time to find Individuals
Cascading Diffusion
Tipping Diffusion
Time in sec
70000
60000
50000
40000
30000
20000
10000
0
5
10
15
20
25
Number of computed individuals



α specifies level of influence of a candidate on voters.
Higher α => more influence.

Outline

- Fast subgraph matching
- Social network optimization problems
- Competitive diffusion problems

Weighted GAP Rules

Ground Rule

$$B_1:X_1 \wedge \ldots B_n:X_n \rightarrow H:f(X_i) \mid w$$

Given Interpretation I:

Satisfaction:

$$I(H) \geq f(I(B_1),\ldots,I(B_n))$$

Weighted Distance from Satisfaction:

$$w * \max(0,\ f(I(B_1),\ldots,I(B_n)))-I(H)\ )$$

Competition

- Hard competition expressed as constraints
  - Example: A person has only one vote
    vote(A,Dem) + vote(A,Republican) ≤ 1
- Soft competition expressed by rule weights which represent the relative probability that the described diffusion will happen
  - Example: If person B votes democratic, then B's husband is likely to vote democrats as well (but not necessarily):

vote(B,Dem):X ∧ wife(A,B):1 → vote(A,Dem):X | 0.8

Probabilistic Model Semantics

- We use the rules and their weights to define a probability distribution over the space of "possible unfoldings of the diffusion process"
  - i.e. interpretations or confidence assignments
  - Exponential family distribution (as used e.g. in p* models)
- $d(P,I) = d(R,I)_x = \left\| \begin{bmatrix} d(R_1,I) \\ \vdots \\ d(R_n,I) \end{bmatrix} \right\|_x$

All ground rules. Compute norm of vector.

P = set of rules
R_i = ground rule

- $\mathbb{P}(I \mid P) = {}^1/_Z \exp(-d(P,I))$
- $Z = \int_{I'} \exp(-d(P \cup IC, I))$

Most Probable Interpretation

- Finding the most probable interpretation (MPI) is an optimization problem
  - Find I which has the highest probability of explaining P

$$\text{argmax}_I \, \mathbf{P}(I \mid P) = \text{argmin}_I \, d(P,I)$$

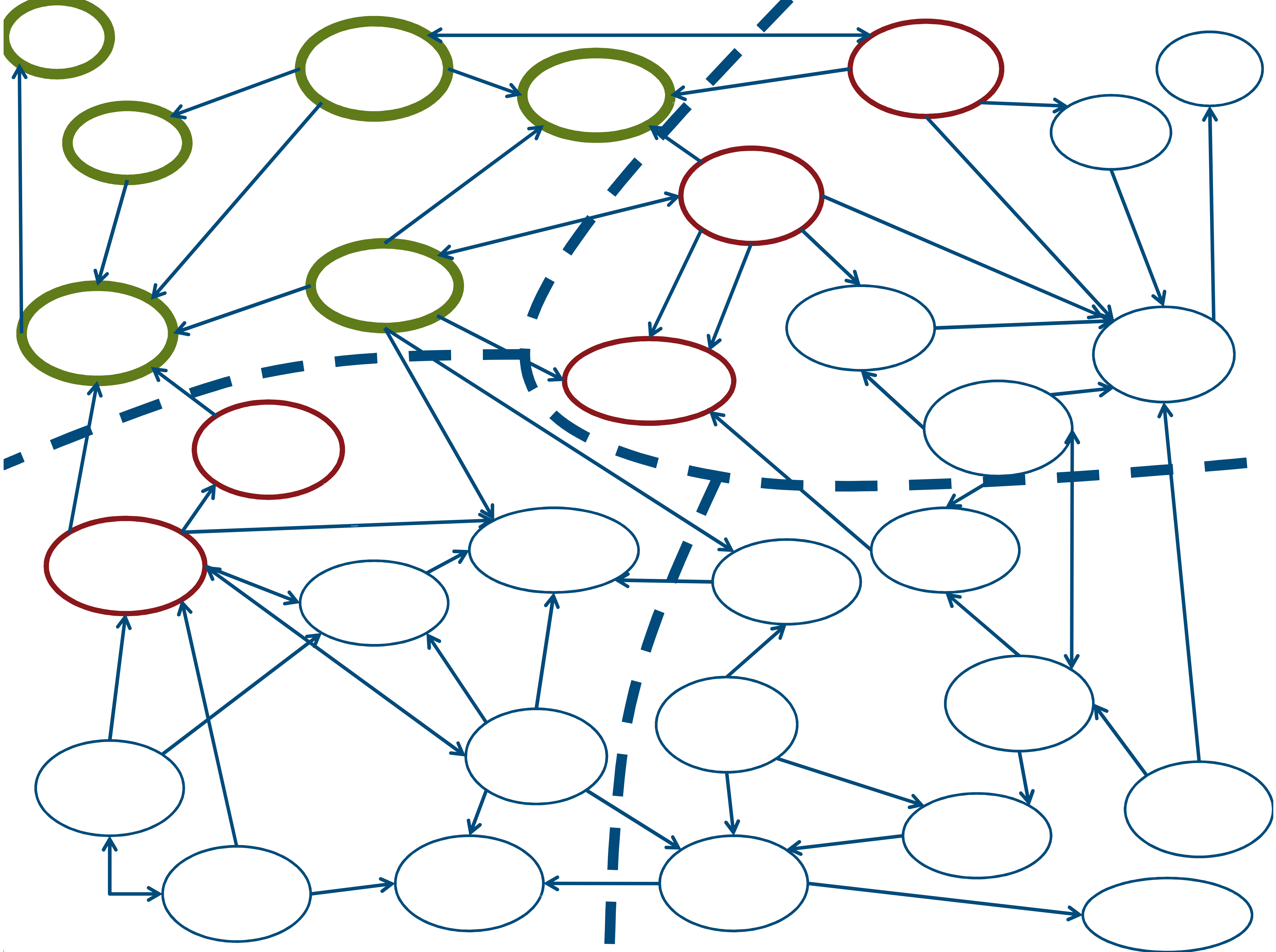
- Restricting the GAP annotations to be convex makes the problem tractable
  - We currently focus on conic annotations which give $O(n^{3.5})$ complexity (i.e. SOCP)
  - n=number of ground rules

Some standard stuff

- Use fixpoint operator to determine the minimal non-ground interpretation
  - Keep the number of ground atoms small
  - Intuition: If there is no evidence for it, we don’t consider it
    - If John and Jane aren’t married, don’t need to consider rules with *wife(John,Jane)*
  - Implementation:
    - Ground out rules iteratively as needed until no further ground atoms are added to the interpretation.
    - Split based on dependency graph analysis.

Approximate Algorithm

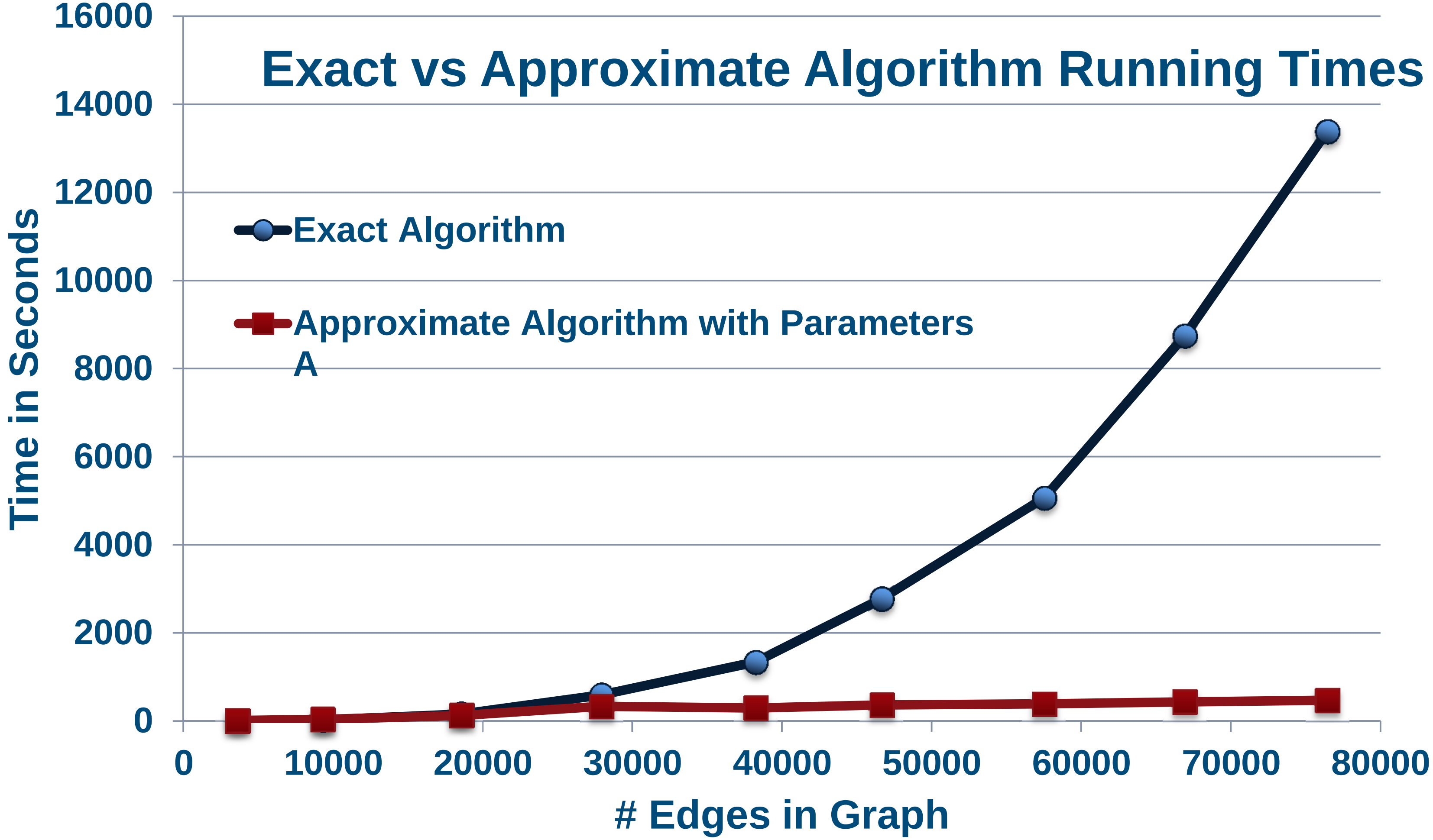
1. Ground out dependency graph as needed with fixpoint operator
2. Partition dependency graph using a modularity maximizing clustering alg
   - Inspired by Blondel et al [06]
   - Aggregate rule weights
3. Compute MPI on each cluster fixing confidence values of outside atoms
4. Go to 1 until change in I < Θ



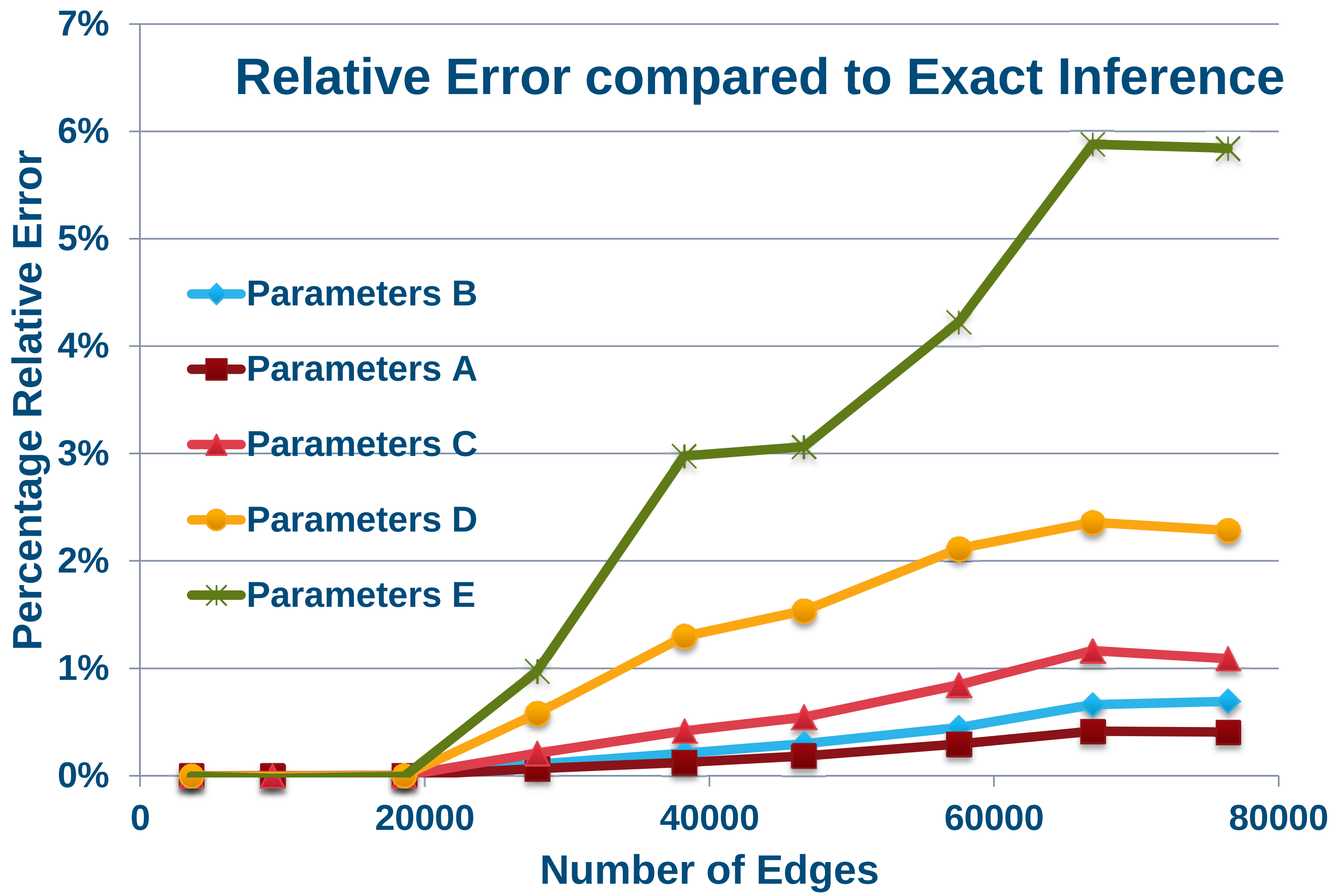
Experiments

- Synthetically generated scale free, labeled social networks
- 6 edge types, 7 rules
- Used different parameter settings for convergence condition
- Executed on single 16 core machine with 256 GB of memory.

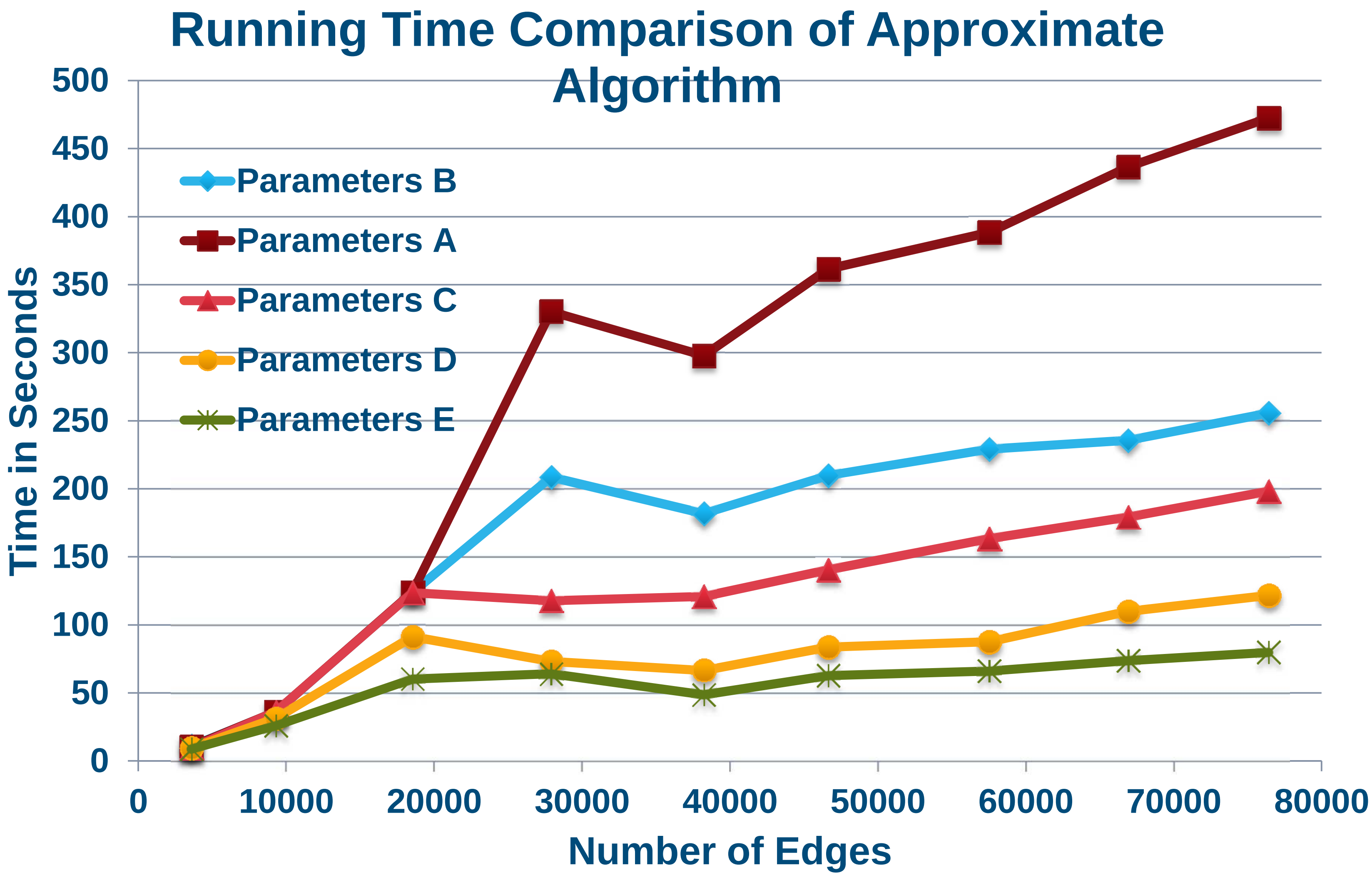
Scalability



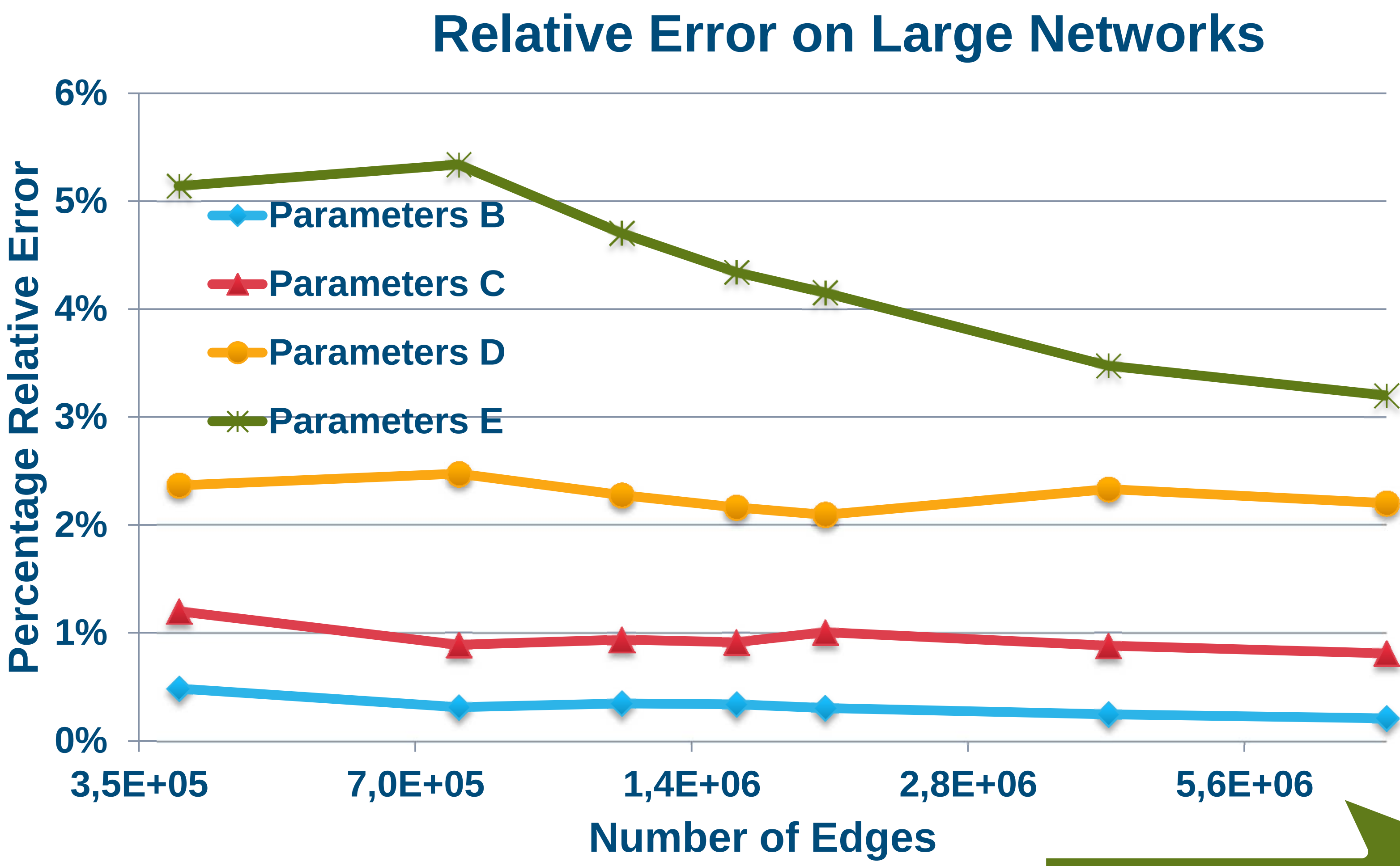
Accuracy


Relative Error compared to Exact Inference
Percentage Relative Error
7%
6%
5%
4%
3%
2%
1%
0%
0
20000
40000
60000
80000
Number of Edges
Parameters B
Parameters A
Parameters C
Parameters D
Parameters E

Runtime

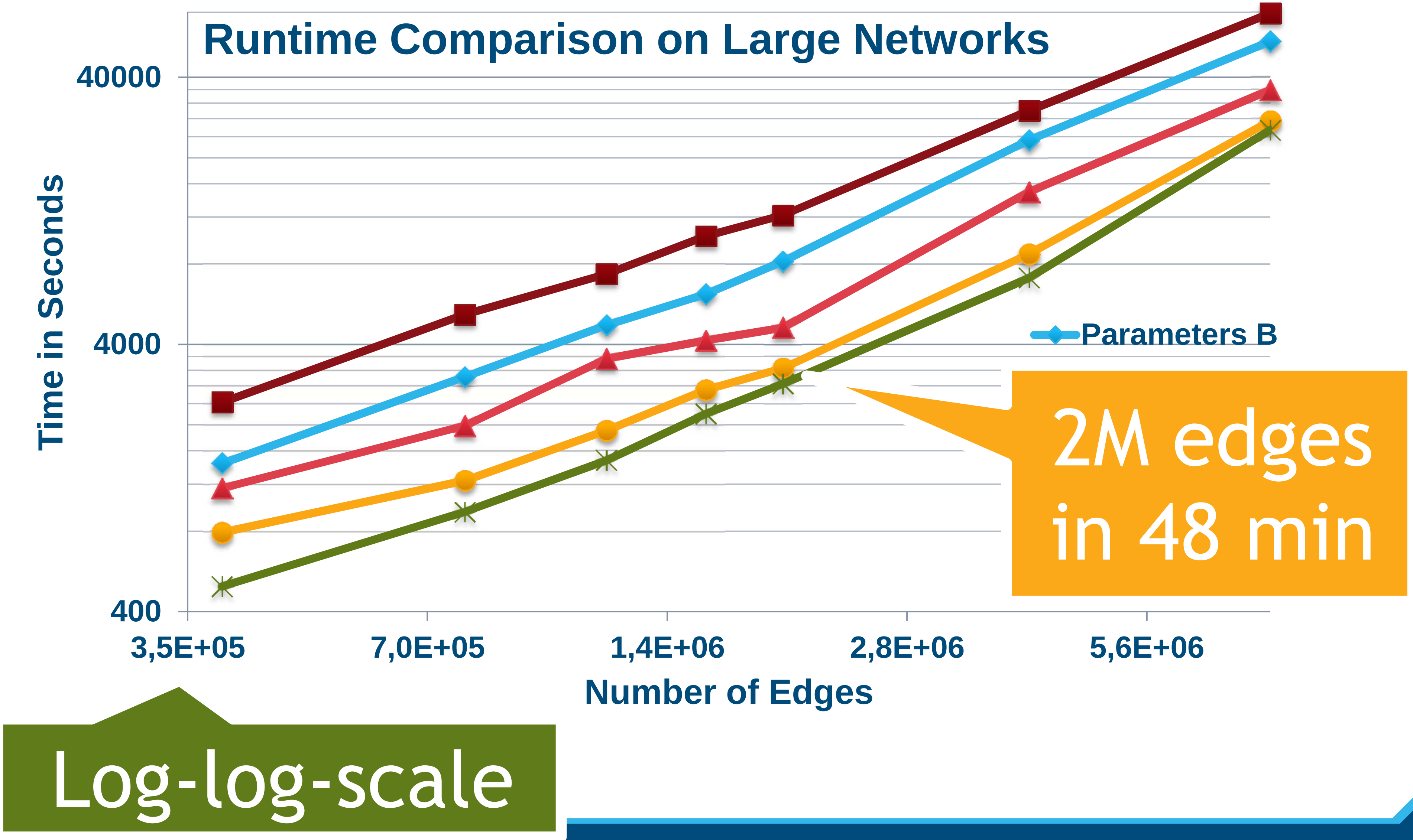

Running Time Comparison of Approximate Algorithm
Time in Seconds
Number of Edges
Parameters B
Parameters A
Parameters C
Parameters D
Parameters E
0
50
100
150
200
250
300
350
400
450
500
0
10000
20000
30000
40000
50000
60000
70000
80000

Accuracy on large SN



Log-scale

Runtime on large SN



Conclusions

- Solving optimization problems on very large graphs is hard.
- First steps have been taken.
- Future steps need to focus on scalability. Developing
  - cloud-based heuristic algorithms plus
  - Smart partitioning/hierarchical clustering approaches.

References & Bibliography

Further Information

- *COSI: Cloud Oriented Subgraph Identification in Massive Social Networks*
  Matthias Bröcheler, Andrea Pugliese and V.S. Subrahmanian, The 2010 International Conference on Advances in Social Networks Analysis and Mining
- *A Scalable Framework for Modeling Competitive Diffusion in Social Networks*
  Matthias Broecheler, Paulo Shakarian, and V.S. Subrahmanian, Proceedings of the 2010 IEEE International Conference on Social Computing, Symposium Section
- *DOGMA: A Disk-Oriented Graph Matching Algorithm*
  Matthias Broecheler, Andrea Pugliese, V.S. Subrahmanian, Proceedings of the 8th International Semantic Web Conference 2009
- *Using Generalized Annotated Programs to Solve Social Network Optimization Problems*
  Paulo Shakarian, V.S. Subrahmanian, Maria Luisa Sapino, Proceedings of the 26th International Conference on Logic Programming – July 2010, full version submitted to a journal in Jan 2011.