Chapter 7: Data Matching

PRINCIPLES OF

DATA INTEGRATION

ANHAI DOAN ALON HALEVY ZACHARY IVES

MORGAN KAUFMANN

Introduction

- Data matching: find structured data items that refer to the same real-world entity
  - entities may be represented by tuples, XML elements, or RDF triples, not by strings as in string matching
  - e.g., (David Smith, 608-245-4367, Madison WI)
    vs (D. M. Smith, 245-4367, Madison WI)
- Data matching arises in many integration scenarios
  - merging multiple databases with the same schema
  - joining rows from sources with different schemas
  - matching a user query to a data item
- One of the most fundamental problems in data integration

Outline

- Problem definition
- Rule-based matching
- Learning- based matching
- Matching by clustering
- Probabilistic approaches to matching
- Collective matching
- Scaling up data matching

Problem Definition

- Given two relational tables X and Y with identical schemas
  - assume each tuple in X and Y describes an entity (e.g., person)
- We say tuple $x \in X$ matches tuple $y \in Y$ if they refer to the same real-world entity
  - (x,y) is called a match
- Goal: find all matches between X and Y

Example

Table X

	Name	Phone	City	State
X_1	Dave Smith	(608) 395 9462	Madison	WI
X_2	Joe Wilson	(408) 123 4265	San Jose	CA
X_3	Dan Smith	(608) 256 1212	Middleton	WI

(a)

Table Y

	Name	Phone	City	State
y_1	David D. Smith	395 9426	Madison	WI
y_2	Daniel W. Smith	256 1212	Madison	WI

(b)

Matches
(x_1, y_1)
(x_3, y_2)

(c)

- Other variations
  - Tables X and Y have different schemas
  - Match tuples within a single table X
  - The data is not relational, but XML or RDF
- These are not considered in this chapter (see bib notes)

Why is This Different than String Matching?

- In theory, can treat each tuple as a string by concatenating the fields, then apply string matching techniques
- But doing so makes it hard to apply sophisticated techniques and domain-specific knowledge
- E.g., consider matching tuples that describe persons
  - suppose we know that in this domain two tuples match if the names and phone match exactly
  - this knowledge is hard to encode if we use string matching
  - so it is better to keep the fields apart

Challenges

- Same as in string matching
- How to match accurately?
  - difficult due to variations in formatting conventions, use of abbreviations, shortening, different naming conventions, omissions, nicknames, and errors in data
  - several common approaches: rule-based, learning-based, clustering, probabilistic, collective
- How to scale up to large data sets
  - again many approaches have been developed, as we will discuss

Outline

- Problem definition
- Rule-based matching
- Learning- based matching
- Matching by clustering
- Probabilistic approaches to matching
- Collective matching
- Scaling up data matching

Rule-based Matching

- The developer writes rules that specify when two tuples match
  - typically after examining many matching and non-matching tuple pairs, using a development set of tuple pairs
  - rules are then tested and refined, using the same development set or a test set
- Many types of rules exist, we will consider
  - linearly weighted combination of individual similarity scores
  - logistic regression combination
  - more complex rules

Linearly Weighted Combination Rules

- Compute the sim score between tuples x and y as a linearly weighted combination of individual sim scores
  - sim(x,y) = $\sum_{i=1}^{n} \alpha_i * sim_i(x, y)$
  - n is number of attributes in each table
  - **s_i(x,y)** is a sim score between the i-th attributes of x and y
  - $\alpha_i \in [0,1]$ is a pre-specified weight that indicates the important of the i-th attribute to sim(x,y), such that $\sum_{i=1}^{n} \alpha_i = 1$
- We declare x and y matched if sim(x,y) $\geq \beta$ for a pre-specified β, and not matched otherwise
  - in another variation: declare x and y matched if sim(x,y) $\geq \beta$, not matched if sim(x,y) $< \gamma$, and subject to human review

Example

Table X

	Name	Phone	City	State
X_1	Dave Smith	(608) 395 9462	Madison	WI
X_2	Joe Wilson	(408) 123 4265	San Jose	CA
X_3	Dan Smith	(608) 256 1212	Middleton	WI

(a)

Table Y

	Name	Phone	City	State
y_1	David D. Smith	395 9426	Madison	WI
y_2	Daniel W. Smith	256 1212	Madison	WI

(b)

Matches

(x_1, y_1)
(x_3, y_2)

(c)

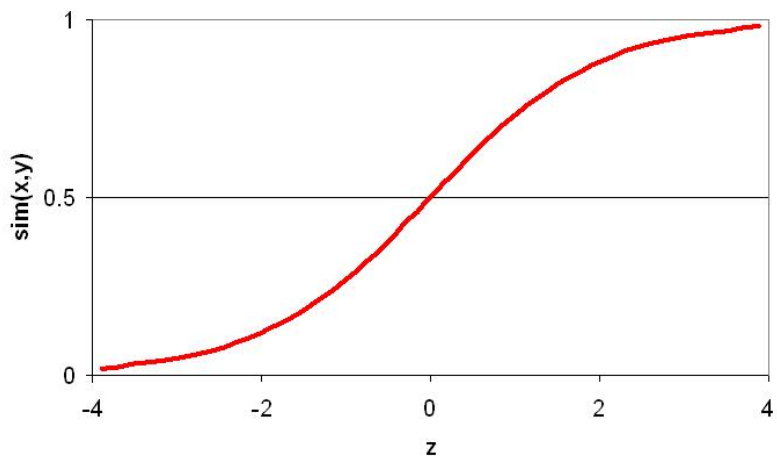
- sim(x,y) =
  $$0.3s_{name}(x,y) + 0.3s_{phone}(x,y) + 0.1s_{city}(x,y) + 0.3s_{state}(x,y)$$
  - $s_{name}(x,y)$: based on Jaro-Winkler
  - $s_{phone}(x,y)$: based on edit distance between x's phone (after removing area code) and y's phone
  - $s_{city}(x,y)$: based on edit distance
  - $s_{state}(x,y)$: based on exact match; yes ➔ 1, no ➔ 0

Pros and Cons

- Pros
  - conceptually simple, easy to implement
  - can learn weights α_i from training data
- Cons
  - an increase ± in the value of any $\mathbf{s}_i$ will cause a linear increase α_i * ± in the value of s
  - in certain scenarios this is not desirable, there after a certain threshold an increase in $\mathbf{s}_i$ should count less (i.e., "diminishing returns" should kick in)
  - e.g., if $\mathbf{s}_{name}(\mathbf{x},\mathbf{y})$ is already 0.95 then the two names already very closely match
    - so any increase in $\mathbf{s}_{name}(\mathbf{x},\mathbf{y})$ should contribute only minimally

Logistic Regression Rules

- address the diminishing-returns problem
- sim(x,y) = 1 / (1 + e^{-z}), where z = $\sum_{i=1}^{n} \alpha_i * sim_i(x, y)$
  - here \alpha_i are not constrained to be in [0,1] and sum to 1
  - so z goes from -∞ to +∞, in which case sim(x,y) gradually increases, but minimally so after z has exceeded a certain value ➔ ensuring diminishing returns



Logistic Regression Rules

- Are also very useful in situations where
  - there are many "signals" (e.g., 10-20) that can contribute to whether two tuples match
  - we don't need all of these signals to "fire" in order to conclude that the tuples match
  - as long as a reasonable number of them fire, we have sufficient confidence
- Logistic regression is a natural fit for such cases
- Hence is quite popular as a first matching method to try

More Complex Rules

- Appropriate when we want to encode more complex matching knowledge
  - e.g., two persons match if names match approximately and either phones match exactly or addresses match exactly

1. If $\mathbf{s}_{name}(\mathbf{x},\mathbf{y}) < 0.8$ then return "not matched"
2. Otherwise if $\mathbf{e}_{phone}(\mathbf{x},\mathbf{y})$ = true then return "matched"
3. Otherwise if $\mathbf{e}_{city}(\mathbf{x},\mathbf{y})$ = true and $\mathbf{e}_{state}(\mathbf{x},\mathbf{y})$ = true then return "matched"
4. Otherwise return "not matched"

Pros and Cons of Rule-Based Approaches

- Pros
  - easy to start, conceptually relatively easy to understand, implement, debug
  - typically run fast
  - can encode complex matching knowledge
- Cons
  - can be labor intensive, it takes a lot of time to write good rules
  - can be difficult to set appropriate weights
  - in certain cases it is not even clear how to write rules
  - learning-based approaches address these issues

Outline

- Problem definition
- Rule-based matching
- Learning- based matching
- Matching by clustering
- Probabilistic approaches to matching
- Collective matching
- Scaling up data matching

Learning-based Matching

- Here we consider supervised learning
  - learn a matching model M from training data, then apply M to match new tuple pairs
  - will consider unsupervised learning later
- Learning a matching model M (the training phase)
  - start with training data: $T = \{(\mathbf{x_1},\mathbf{y_1},\mathbf{l}_1), \ldots (\mathbf{x}_n,\mathbf{y}_n,\mathbf{l}_n)\}$, where each $(\mathbf{x}_i,\mathbf{y}_i)$ is a tuple pair and $\mathbf{l}_i$ is a label: "yes" if $\mathbf{x}_i$ matches $\mathbf{y}_i$ and "no" otherwise
  - define a set of features $\mathbf{f}_1, \ldots, \mathbf{f}_m$, each quantifying one aspect of the domain judged possibly relevant to matching the tuples

Learning-based Matching

- Learning a matching model M (continued)
  - convert each training example $(\mathbf{x}_i,\mathbf{y}_i,\mathbf{l}_i)$ in T to a pair $(<\mathbf{f_1}(\mathbf{x}_i,\mathbf{y}_i), \ldots, \mathbf{f}_m(\mathbf{x}_i,\mathbf{y}_i)>, \mathbf{c}_i)$
    - $\mathbf{v}_i = <\mathbf{f_1}(\mathbf{x}_i,\mathbf{y}_i), \ldots, \mathbf{f}_m(\mathbf{x}_i,\mathbf{y}_i)>$ is a feature vector that encodes $(\mathbf{x}_i,\mathbf{y}_i)$ in terms of the features
    - $\mathbf{c}_i$ is an appropriately transformed version of label l_i (e.g., yes/no or 1/0, depending on what matching model we want to learn)
  - thus T is transformed into T' = $\{(\mathbf{v_1},\mathbf{c}_1), \ldots, (\mathbf{v}_n,\mathbf{c}_n)\}$
  - apply a learning algorithm (e.g. decision trees, SVMs) to T' to learn a matching model M

Learning-based Matching

- Applying model M to match new tuple pairs
  - given pair (x,y), transform it into a feature vector
    - $v = \langle \mathbf{f}_1(\mathbf{x},\mathbf{y}), \ldots, \mathbf{f}_m(\mathbf{x},\mathbf{y}) \rangle$
  - apply M to v to predict whether x matches y

Example: Learning a Linearly Weighted Rule

<a_1 = (Mike Williams, (425) 247 4893, Seattle, WA), b_1 = (M. Williams, 247 4893, Redmond, WA), yes>
<a_2 = (Richard Pike, (414) 256 1257, Milwaukee, WI), b_2 = (R. Pike, 256 1237, Milwaukee, WI), yes>
<a_3 = (Jane McCain, (206) 111 4215, Renton, WA), b_3 = (J. M. McCain, 112 5200, Renton, WA), no>

match names match phones match cities match states check area code against city

$v_1 = <[s_1(a_1,b_1), s_2(a_1,b_1), s_3(a_1,b_1), s_4(a_1,b_1), s_5(a_1,b_1), s_6(a_1,b_1)], 1>$

$v_2 = <[s_1(a_2,b_2), s_2(a_2,b_2), s_3(a_2,b_2), s_4(a_2,b_2), s_5(a_2,b_2), s_6(a_2,b_2)], 1>$

$v_3 = <[s_1(a_3,b_3), s_2(a_3,b_3), s_3(a_3,b_3), s_4(a_3,b_3), s_5(a_3,b_3), s_6(a_3,b_3)], 0>$

- $\mathbf{s}_1$ and $\mathbf{s}_2$ use Jaro-Winkler and edit distance
- $\mathbf{s}_3$ uses edit distance (ignoring area code of a)
- $\mathbf{s}_4$ and $\mathbf{s}_5$ return 1 if exact match, 0 otherwise
- $\mathbf{s}_6$ encodes a heuristic constraint

Example:
Learing a Linearly Weighted Rule

- Goal: learn rule s(a,b) = $\sum_{i=1}^{6} \alpha_i s_i(a,b)$
- Perform a least-squares linear regression on training data

v_1 = <[$s_1(a_1,b_1)$, $s_2(a_1,b_1)$, $s_3(a_1,b_1)$, $s_4(a_1,b_1)$, $s_5(a_1,b_1)$, $s_6(a_1,b_1)$], 1>

v_2 = <[$s_1(a_2,b_2)$, $s_2(a_2,b_2)$, $s_3(a_2,b_2)$, $s_4(a_2,b_2)$, $s_5(a_2,b_2)$, $s_6(a_2,b_2)$], 1>

v_3 = <[$s_1(a_3,b_3)$, $s_2(a_3,b_3)$, $s_3(a_3,b_3)$, $s_4(a_3,b_3)$, $s_5(a_3,b_3)$, $s_6(a_3,b_3)$], 0>

to find weights α_i that minimizes the squared error

$$\sum_{i=1}^{3} (c_i - \sum_{j=1}^{6} \alpha_j s_j(v_i))^2$$

- c_i is the label associated with v_i

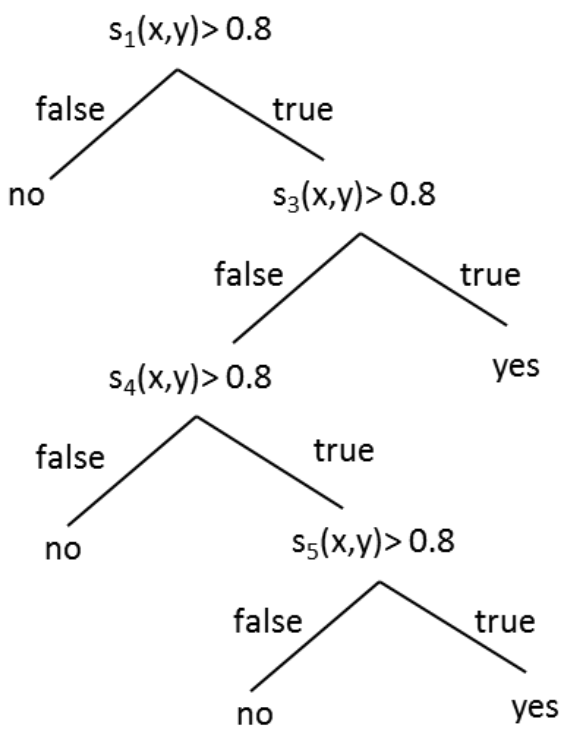
Example: Learning a Decision Tree

match names match phones match cities match states check area code against city

$v_1 = <[s_1(a_1,b_1), s_2(a_1,b_1), s_3(a_1,b_1), s_4(a_1,b_1), s_5(a_1,b_1), s_6(a_1,b_1)], \text{yes}>$

$v_2 = <[s_1(a_2,b_2), s_2(a_2,b_2), s_3(a_2,b_2), s_4(a_2,b_2), s_5(a_2,b_2), s_6(a_2,b_2)], \text{yes}>$

$v_3 = <[s_1(a_3,b_3), s_2(a_3,b_3), s_3(a_3,b_3), s_4(a_3,b_3), s_5(a_3,b_3), s_6(a_3,b_3)], \text{no}>$



Now the labels are yes/no, not 1/0

The Pros and Cons of Learning-based Approach

- Pros compared to rule-based approaches
  - in rule-based approaches must manually decide if a particular feature is useful ➔ labor intensive and limit the number of features we can consider
  - learning-based ones can automatically examine a large number of features
  - learning-based approaches can construct very complex “rules”
- Cons
  - still require training examples, in many cases a large number of them, which can be hard to obtain
  - clustering addresses this problem

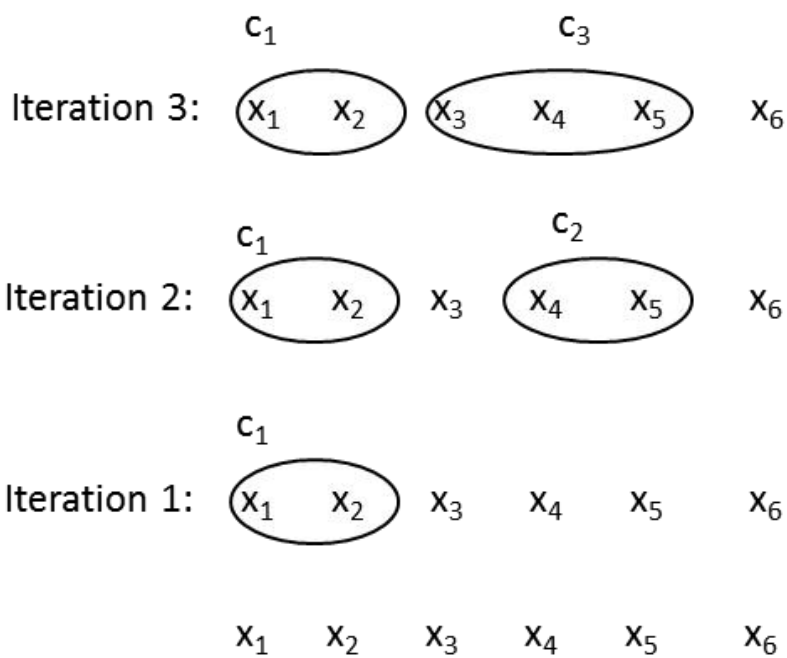
Outline

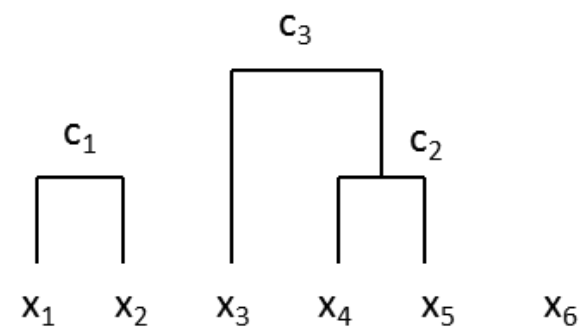
- Problem definition
- Rule-based matching
- Learning- based matching
- Matching by clustering
- Probabilistic approaches to matching
- Collective matching
- Scaling up data matching

Matching by Clustering

- Many common clustering techniques have been used
  - agglomerative hierarchical clustering (AHC), k-means, graph-theoretic, …
  - here we focus on AHC, a simple yet very commonly used one
- AHC
  - partitions a given set of tuples D into a set of clusters
    - all tuples in a cluster refer to the same real-world entity, tuples in different clusters refer to different entities
  - begins by putting each tuple in D into a single cluster
  - iteratively merges the two most similar clusters
  - stops when a desired number of clusters has been reached, or until the similarity between two closest clusters falls below a pre-specified threshold

Example





- sim(x,y) =
  $0.3s_{name}(x,y) + 0.3s_{phone}(x,y) + 0.1s_{city}(x,y) + 0.3s_{state}(x,y)$

Computing a Similarity Score between Two Clusters

- Let c and d be two clusters
- Single link: $s(c,d) = \mathbf{min}_{x_i \in c,\, y_j \in d}\ \mathbf{sim}(x_i, y_j)$
- Complete link: $s(c,d) = \mathbf{max}_{x_i \in c,\, y_j \in d}\ \mathbf{sim}(x_i, y_j)$
- Average link: $s(c,d) = [\sum_{x_i \in c,\, y_j \in d} \mathbf{sim}(x_i, y_j)] / [\#\text{ of } (x_i, y_j) \text{ pairs}]$
- Canonical tuple
  - create a canonical tuple that represents each cluster
  - sim between c and d is the sim between their canonical tuples
  - canonical tuple is created from attribute values of the tuples
    - e.g., "Mike Williams" and "M. J. Williams" ➔ "Mike J. Williams"
    - (425) 247 4893 and 247 4893 ➔ (425) 247 4893

Key Ideas underlying the Clustering Approach

- View matching tuples as the problem of constructing entities (i.e., clusters)
- The process is iterative
  - leverage what we have known so far to build "better" entities
- In each iteration merge all matching tuples within a cluster to build an "entity profile", then use it to match other tuples ➔ merging then exploiting the merged information to help matching
- These same ideas appear in subsequent approaches that we will cover

Outline

- Problem definition
- Rule-based matching
- Learning- based matching
- Matching by clustering
- Probabilistic approaches to matching
- Collective matching
- Scaling up data matching

Probabilistic Approaches to Matching

- Model matching domain using a probability distribution
- Reason with the distribution to make matching decisions
- Key benefits
  - provide a principled framework that can naturally incorporate a variety of domain knowledge
  - can leverage the wealth of prob representation and reasoning techniques already developed in the AI and DB communities
  - provide a frame of reference for comparing and explaining other matching approaches
- Disadvantages
  - computationally expensive
  - often hard to understand and debug matching decisions

What We Discuss Next

- Most current probabilistic approaches employ generative models
  - these encode full prob distributions and describe how to generate data that fit the distributions
- Some newer approaches employ discriminative models (e.g., conditional random fields)
  - these encode only the probabilities necessary for matching (e.g., the probability of a label given a tuple pair)
- Here we focus on generative model based approaches
  - first we explain Bayesian networks, a simple type of generative models
  - then we use them to explain more complex ones

Bayesian Networks: Motivation

- Let $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ be a set of variables
  - e.g., X = {Cloud, Sprinkler}
- A state = an assignment of values to all variables in X
  - e.g., s = {Cloud = true, Sprinkler = on}
- A probability distribution P assigns to each state $\mathbf{s}_i$ a value $\mathbf{P}(\mathbf{s}_i)$ such that $\sum_{\mathbf{s}i \in S} \mathbf{P}(\mathbf{s}_i) = 1$
  - S is the set of all states
  - $\mathbf{P}(\mathbf{s}_i)$ is called the probability of $\mathbf{s}_i$

States		Probabilities
Cloud	Sprinkler	
t	on	0.3
t	off	0.3
f	on	0.3
f	off	0.1

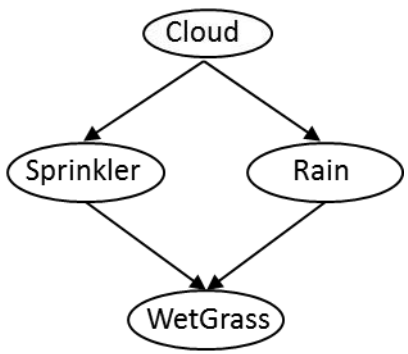
Bayesian Networks: Motivation

- Reasoning with prob models: to answer queries such as
  - P(A = a)? P(A = a|B = b) = ? where A and B are subsets of vars
- Examples
  - P(Cloud = t) = 0.6
    (by summing over first two rows)
  - P(Cloud = t | Sprinkler = off)
    = 0.75

States		Probabilities
Cloud	Sprinkler	
t	on	0.3
t	off	0.3
f	on	0.3
f	off	0.1

- Problems: can't enumerate all states, too many of them
  - real-world apps often use hundreds or thousands of variables
- Bayesian networks solve this by providing a compact representation of a probability distribution

Baysian Networks: Representation



Cloud	
t	f
0.3	0.7

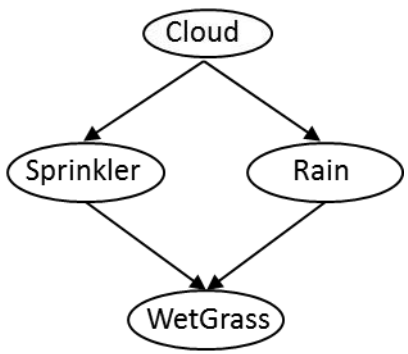
Cloud	Sprinkler	
	on	off
t	0.2	0.8
f	0.8	0.2

Cloud	Rain	
	t	f
t	0.6	0.4
f	0.3	0.7

Sprinkler	Rain	WetGrass	
		t	f
on	t	1	0
on	f	1	0
off	t	1	0
off	f	0.1	0.9

- nodes = variables, edges = probabilistic dependencies
- Key assertion: each node is probabilistically independent of its non-descendants given the values of its parents
  - e.g., WetGrass is independent of Cloud given Sprinkler & Rain
  - Sprinkler is independent of Rain given Cloud

Baysian Networks: Representation



Cloud	
t	f
0.3	0.7

Cloud	Sprinkler	
	on	off
t	0.2	0.8
f	0.8	0.2

Cloud	Rain	
	t	f
t	0.6	0.4
f	0.3	0.7

Sprinkler	Rain	WetGrass	
		t	f
on	t	1	0
on	f	1	0
off	t	1	0
off	f	0.1	0.9

- The key assertation allows us to write
  - P(C,S,R,W) = P(C).P(S|C).P(R|C).P(W|R)
  - Thus, to compute P(C,S,R,W), need to know only four local probability distributions, also called conditional probability tables (CPTs)
  - use only 9 statements to specify the full PD, instead of 16

Bayesian Networks: Reasoning

- Also called performing inference
  - computing P(A) or P(A|B), where A and B are subsets of vars
- Performing exact inference is NP-hard
  - taking time exponential in number of variables in worst case
- Data matching approaches address this in three ways
  - for certain classes of BNs there are polynomial-time algorithms or closed-form equations that return exact answers
  - use standard approximate inference algorithms for BNs
  - develop approximate algorithms tailored to the domain at hand

Learning Bayesian Networks

- To use a BN, current data matching approaches
  - typically require a domain expert to create the graph
  - then learn the CPTs from training data
- Training data: set of states we have observed
  - e.g., $\mathbf{d}_1$ = (Cloud=t, Sprinkler=off, Rain=t, WetGrass=t)
    $\mathbf{d}_2$ = (Cloud=t, Sprinkler=off, Rain=f, WetGrass=f)
    $\mathbf{d}_3$ = (Cloud=f, Sprinkler=on, Rain=f, WetGrass=t)
- Two cases
  - training data has no missing values
  - training dta has some missing values
    - greatly complicates learning, must use EM algorithm
  - we now consider them in turn

Learning with No Missing Values

Training data D

$d_1 = (1,0)$
$d_2 = (1,0)$
$d_3 = (1,1)$
$d_4 = (0,1)$

CPTs to be learned

A	
1	0
?	?

A	B	
	1	0
1	?	?
0	?	?

CPTs learned from training data

A	
1	0
0.75	0.25

A	B	
	1	0
1	0.33	0.67
0	1	0

- $\mathbf{d}_1 = (1,0)$ means A = 1 and B = 0

Learning with No Missing Values

- Let θ be the probabilities to be learned. Want to find θ^* that maximizes the prob of observing the training data D
  - $\theta^* = \arg\max_\theta P(D|\theta)$
- θ^* can be obtained by simple counting over D
- E.g., to compute P(A = 1): count # of examples where A = 1, divide by total # of examples
- To compute P(B = 1 | A = 1): divide # of examples where B = 1 and A =1 by # of examples where A = 1
- What if not having sufficient data for certain states?
  - e.g., need to compute P(B=1|A=1), but # states where A = 1 is 0
  - need smoothing of the probabilities (see notes)

Learning with Missing Values

- Training examples may have missing values
  - d = (Cloud=?, Sprinkler=off, Rain=?, WetGrass=t)
- Why?
  - we failed to observe a variable
    - e.g., slept and did not observe whether it rained
  - the variable by its nature is unobservable
    - e.g., werewolves who only get out during dark moonless night
      ➔ can't never tell if the sky is cloudy
- Can't use counting as before to learn (e.g., infer CPTs)
- Use EM algorithm

The Expectation-Maximization (EM) Algorithm

- Key idea:
  - two unknown quantities: \theta and missing values in D
  - iteratively estimates these two, by assigning initial values, then using one to predict the other and vice versa, until convergence

Training data D

$d_1 = (?,0)$
$d_2 = (?,0)$
$d_3 = (?,1)$

The EM algorithm

1. Initialize θ and let it be θ^0. Set $n = 0$.
2. (Expectation) Use θ^n to estimate missing values of D. Let the resulting set be D^n.
3. (Maximization) Compute θ^{n+1} using counting over D^n.
4. Exit and return θ^n if no increase is achieved for $P(\theta^n | D^n)$. Otherwise repeat Steps 2-3 with $n = n + 1$.

An Example

Training data D

$d_1 = (?,0)$
$d_2 = (?,0)$
$d_3 = (?,1)$

θ^0

A	
1	0
0.5	0.5

A	B	
	1	0
1	0.6	0.4
0	0.5	0.5

D^0

$d_1 = \begin{bmatrix} P(A=1) = 0.44 \\ P(A=0) = 0.56 \end{bmatrix}, B=0$

$d_2 = \begin{bmatrix} P(A=1) = 0.44 \\ P(A=0) = 0.56 \end{bmatrix}, B=0$

$d_3 = \begin{bmatrix} P(A=1) = 0.54 \\ P(A=0) = 0.46 \end{bmatrix}, B=1$

θ^1

A	
1	0
0.47	0.53

A	B	
	1	0
1	0.38	0.62
0	0.29	0.71

- EM also aims to find θ that maximizes $P(D|\theta)$
  - just like the counting approach in case of no missing values
- It may not find the globally maximal θ^*
  - converging instead to a local maximum

Bayesian Networks as Generative Models

- Generative models
  - encode full probability distributions
  - specify how to generate data that fit such distributions
- Bayesian networks: well-known examples of such models
- A perspective on how the data is generated helps
  - guide the construction of the Bayesian network
  - discover what kinds of domain knowledge to be naturally incorporated into the network structure
  - explain the network to users
- We now examine three prob approaches to matching that employ increasingly complex generative models

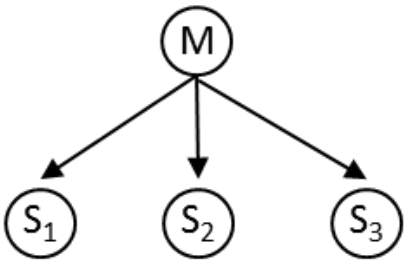
Data Matching with Naïve Bayes

- Define variable M that represents whether a and b match
- Our goal is to compute P(M|a,b)
  - declare a and b matched if P(M=t|a,b) > P(M=f|a,b)
- Assume P(M|a,b) depends only on $\mathbf{S}_1, \ldots, \mathbf{S}_n$, features that are functions that take as input a and b
  - e.g., whether two last names match, edit distance between soc sec numbers, whether the first initials match, etc.
- P(M|a,b) = $\mathbf{P(M|S_1, \ldots, S_n)}$, using Bayes Rule, we have
  - $\mathbf{P(M|S_1, \ldots, S_n) = P(S_1, \ldots, S_n|M)P(M)/P(S_1, \ldots, S_n)}$

Data Matching with Naïve Bayes

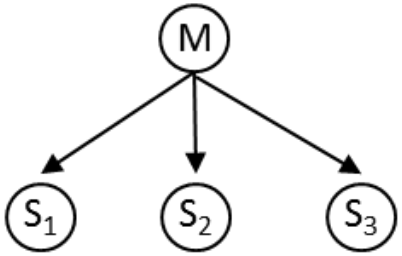
- $\mathbf{P(M|S_1, \ldots, S_n) = P(S_1, \ldots, S_n|M)P(M)/P(S_1, \ldots, S_n)}$
- Assume $\mathbf{S_1, \ldots, S_n}$ are independent given M
  - $\mathbf{P(S_1, \ldots, S_n|M)} = \prod_{i=1}^{n} P(Si|M)$
- We also have
  - $\mathbf{P(S_1, \ldots, S_n) = P(S_1, \ldots, S_n|M{=}t)P(M{=}t) + P(S_1, \ldots, S_n|M{=}f)P(M{=}f)}$
- So, to compute $\mathbf{P(M|S_1, \ldots, S_n)}$, i.e., P(M|a,b), we only need to know $\mathbf{P(S_1|M)}$, …, $\mathbf{P(S_n|M)}$, and P(M)
- The above model is captured in a Bayesian network called a Naïve Bayes model

The Naïve Bayes Model



- The assumption that $\mathbf{S}_1, \ldots, \mathbf{S}_n$ are independent of one another given M is called the Naïve Bayes assumption
  - which often does not hold in practice
- Computing $\mathbf{P}(\mathbf{M} | \mathbf{S}_1, \ldots, \mathbf{S}_n)$ is performing an inference on the above Bayesian network
- Given the simple form of the network, this inference can be performed easily, if we know the CPTs

Learning the CPTs Given Training Data



$(a_1, b_1, \text{ yes})$ $\quad$ $<t, s_1(a_1,b_1), s_2(a_1,b_1), s_3(a_1,b_1)>$

$(a_2, b_2, \text{ yes})$ $\quad$ $<t, s_1(a_2,b_2), s_2(a_2,b_2), s_3(a_2,b_2)>$

$(a_3, b_3, \text{ no})$ $\quad$ $<f, s_1(a_3,b_3), s_2(a_3,b_3), s_3(a_3,b_3)>$

- Convert training examples into feature vectors
- Then apply learning with no missing values as described earlier (i.e., counting) to feature vectors to learn the CPTs
- Once learned, can apply the BN to match a new pair of tuples (a,b) by comparing P(M=t|a,b) and P(M=f|a,b)
  - this reduces to comparing $\prod_{i=1}^{n} P(Si|M = t)P(M = t)$ and $\prod_{i=1}^{n} P(Si|M = f)P(M = f)$
  - no need to compute $\mathbf{P(S_1, \ldots, S_n)}$

Learning the CPTs Given No Training Data

(a_4, b_4)	$<?, s_1(a_4,b_4), s_2(a_4,b_4), s_3(a_4,b_4)>$
(a_5, b_5)	$<?, s_1(a_5,b_5), s_2(a_5,b_5), s_3(a_5,b_5)>$
(a_6, b_6)	$<?, s_1(a_6,b_6), s_2(a_6,b_6), s_3(a_6,b_6)>$

- Assume $(\mathbf{a_4},\mathbf{b_4})$, …, $(\mathbf{a_6},\mathbf{b_6})$ are tuple pairs to be matched
- Convert these pairs into training data with missing values
  - the missing value is the correct label for each pair (i.e., the value for variable M: "matched", "not matched")
- Now apply EM algorithm to learn both the CPTs and the missing values at the same time
  - once learned, the missing values are the labels (i.e., "matched", "not matched") that we want to see

Summary

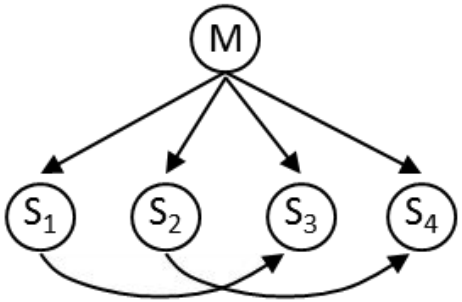
- The developer specifies the network structure, i.e., the directed acyclic graph
  - which is a Naïve Bayesian network structure in this case
- If given training data in form of tuple pairs together with their correct labels (matched, not matched), we can learn the CPTs of the Naïve Bayes network using counting
  - then we use the trained network to match new tuple pairs (which means performing exact inferences to compute P(M|a,b))
- People also refer to the Naïve Bayesian network as a Naïve Bayesian classifier

Summary (cont.)

- If no training data is given, but we are given a set of tuple pairs to be matched, then we can use these tuple pairs to construct training data with missing values
  - we then apply EM to learn the missing values and the CPTs
  - the missing values are the match predictions that we want
- The above procedures (for both cases of having and not having training data) can be generalized in a straightforward fashion to arbitrary Bayesian network cases, not just Naïve Bayesian ones

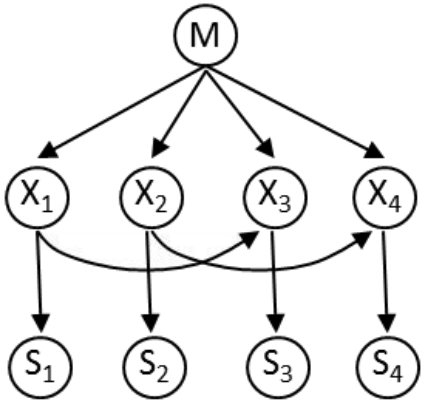
Modeling Feature Correlations

- Naïve Bayes assumes no correlations among $\mathbf{S}_1, \ldots, \mathbf{S}_n$
- We may want to model such correlations
  - e.g., if $\mathbf{S}_1$ models whether soc sec numbers match, and $\mathbf{S}_3$ models whether last names match, then there exists a correlation between the two
- We can then train and apply this more expressive BN to match data
- Problem: "blow up" the number of probs in the CPTs
  - assume n is # of features, q is the # of parents per node, and d is the # of values per node ➔ $\mathbf{O(nd^q)}$ vs. 2dn for the comparable Naïve Bayesian



Modeling Feature Correlations

- A possible solution
  - assume each tuple has k attributes
  - consider only k features $\mathbf{S}_1, \ldots, \mathbf{S}_k$, the i-th feature compares only values of the i-th attributes
  - introduce binary variables $\mathbf{X}_i$, $\mathbf{X}_i$ models whether the i-th attributes should match, given that the tuples match
  - then model correlation only at the $\mathbf{X}_i$ level, not at $\mathbf{S}_i$ level
- This requires far fewer probs in CPTs
  - assume each node has q parents, and each S_i has d vallues, then we need $\mathbf{O}(\mathbf{k2}^q + 2kd)$ probs



Key Lesson

- Constructing a BN for a matching problem is an art that must consider the trade-offs among many factors
  - how much domain knowledge to be captured
  - how accurately we can learn the network
  - how efficiently we can do so
- The notes present an even more complex example about matching mentions of entities in text

Outline

- Problem definition
- Rule-based matching
- Learning- based matching
- Matching by clustering
- Probabilistic approaches to matching
- Collective matching
- Scaling up data matching

Collective Matching

- Matching approaches discussed so far make independent matching decisions
  - decide whether a and b match independently of whether any two other tuples c and d match
- Matching decisions hower are often correlated
  - exploiting such correlations can improve matching accuracy

An Example

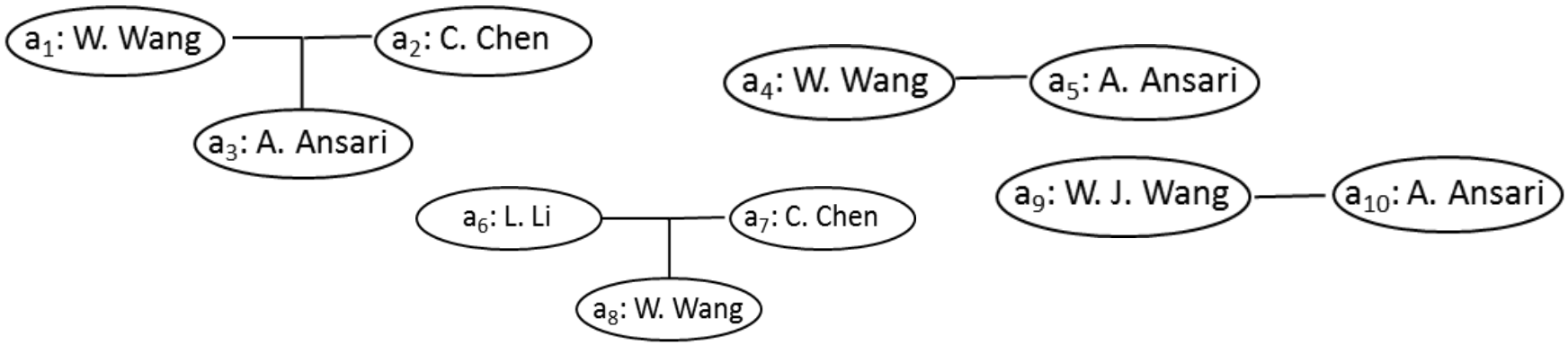
W. Wang, C. Chen, A. Ansari, A mouse immunity model
W. Wang, A. Ansari, Evaluating immunity models
L. Li, C. Chen, W. Wang, Measuring protein-bound fluxetine
W. J. Wang, A. Ansari, Autoimmunity in biliary cirrhosis

	First initial	Middle inititial	Last name
a_1	W		Wang
a_2	C		Chen
	...	...	...
a_9	W	J	Wang
a_{10}	A		Ansari

- Goal: match authors of the four papers listed above
- Solution
  - extract their names to create the table above
  - apply current approaches to match tuples in table
- This fails to exploit co-author relationships in the data

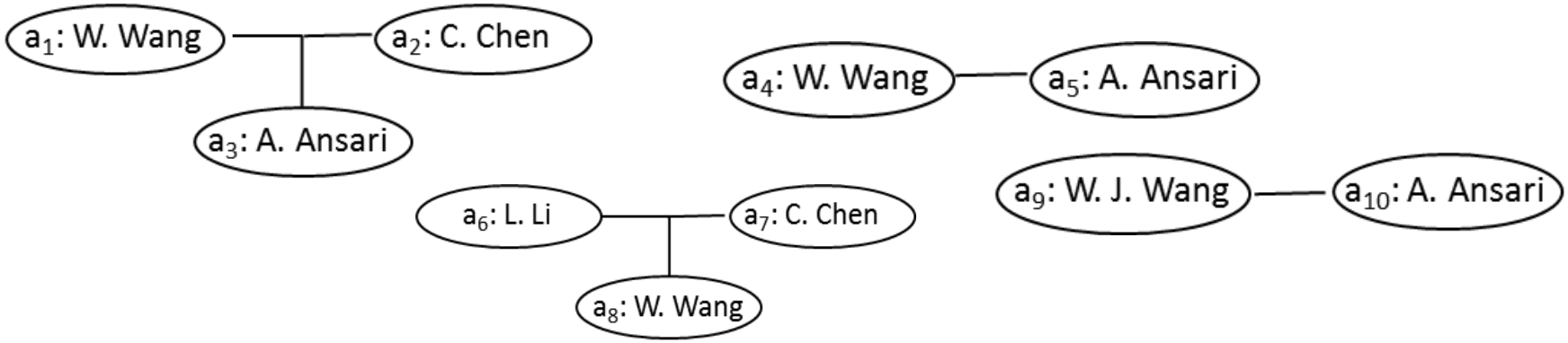
An Example (cont.)

W. Wang, C. Chen, A. Ansari, A mouse immunity model
W. Wang, A. Ansari, Evaluating immunity models
L. Li, C. Chen, W. Wang, Measuring protein-bound fluxetine
W. J. Wang, A. Ansari, Autoimmunity in biliary cirrhosis



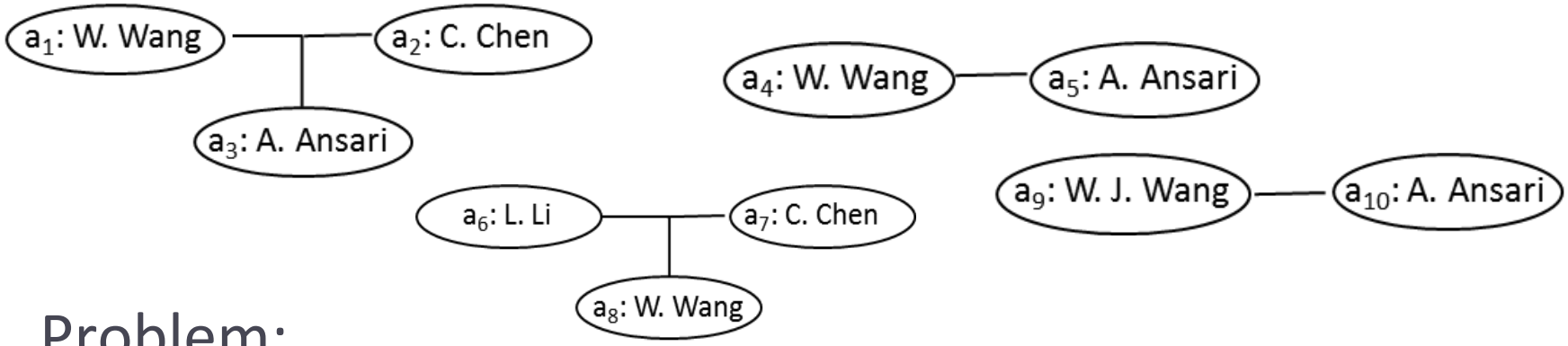
- nodes = authors, hyperedges connect co-authors
- Suppose we have matched $\mathbf{a}_3$ and $\mathbf{a}_5$
  - then intuitively $\mathbf{a}_1$ and $\mathbf{a}_4$ should be more likely to match
  - they share the same name and same co-author relationship to the same author

An Example (cont.)



- First solution:
  - add coAuthors attribute to the tuples
    - e.g., tuple a_1 has coAuthors = {C. Chen, A. Ansari}
    - tuple a_4 has coAuthors = {A. Ansari}
  - apply current methods, use say Jaccard measure for coAuthors

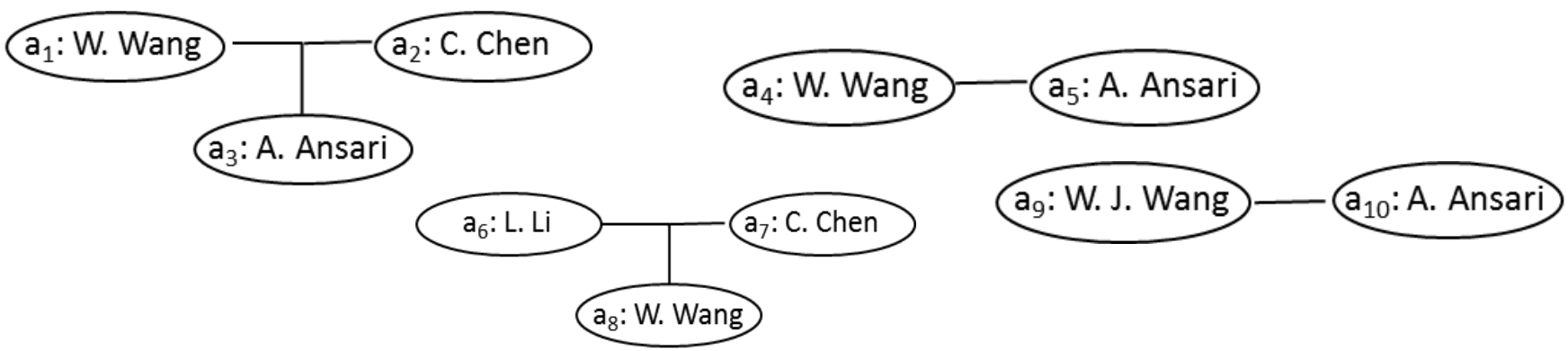
An Example (cont.)



- Problem:
  - suppose $\mathbf{a}_3$: A. Ansari and $\mathbf{a}_5$: A. Ansari share same name but do not match
  - we would match them, and incorrectly boost score of $\mathbf{a}_1$ and $\mathbf{a}_4$
- How to fix this?
  - want to match $\mathbf{a}_3$ and $\mathbf{a}_5$, then use that info to help match $\mathbf{a}_1$ and $\mathbf{a}_4$; also want to do the opposite
  - so should match tuples collectively, all at once and iteratively

Collective Matching using Clustering

- Many collective matching approaches exist
  - clustering-based, probabilistic, etc.
- Here we consider clustering-based (see notes for more)
- Assume input is graph
  - nodes = tuples to be matched
  - edges = relationships among tuples



Collective Matching using Clustering

- To match, perform agglomerative hierarchical clustering
  - but modify sim measure to consider correlations among tuples
- Let A and B be two clusters of nodes, define
  - sim(A,B) = α * $\mathbf{sim}_{\text{attributes}}(\mathbf{A,B})$ + (1- α) * $\mathbf{sim}_{\text{neighbors}}(\mathbf{A,B})$
  - α is pre-defined weight
  - $\mathbf{sim}_{\text{attributes}}(\mathbf{A,B})$ uses only attributes of A and B, examples of such scores are single link, complete link, average link, etc.
- $\mathbf{sim}_{\text{neighbors}}(\mathbf{A,B})$ considers correlations
  - we discuss it next

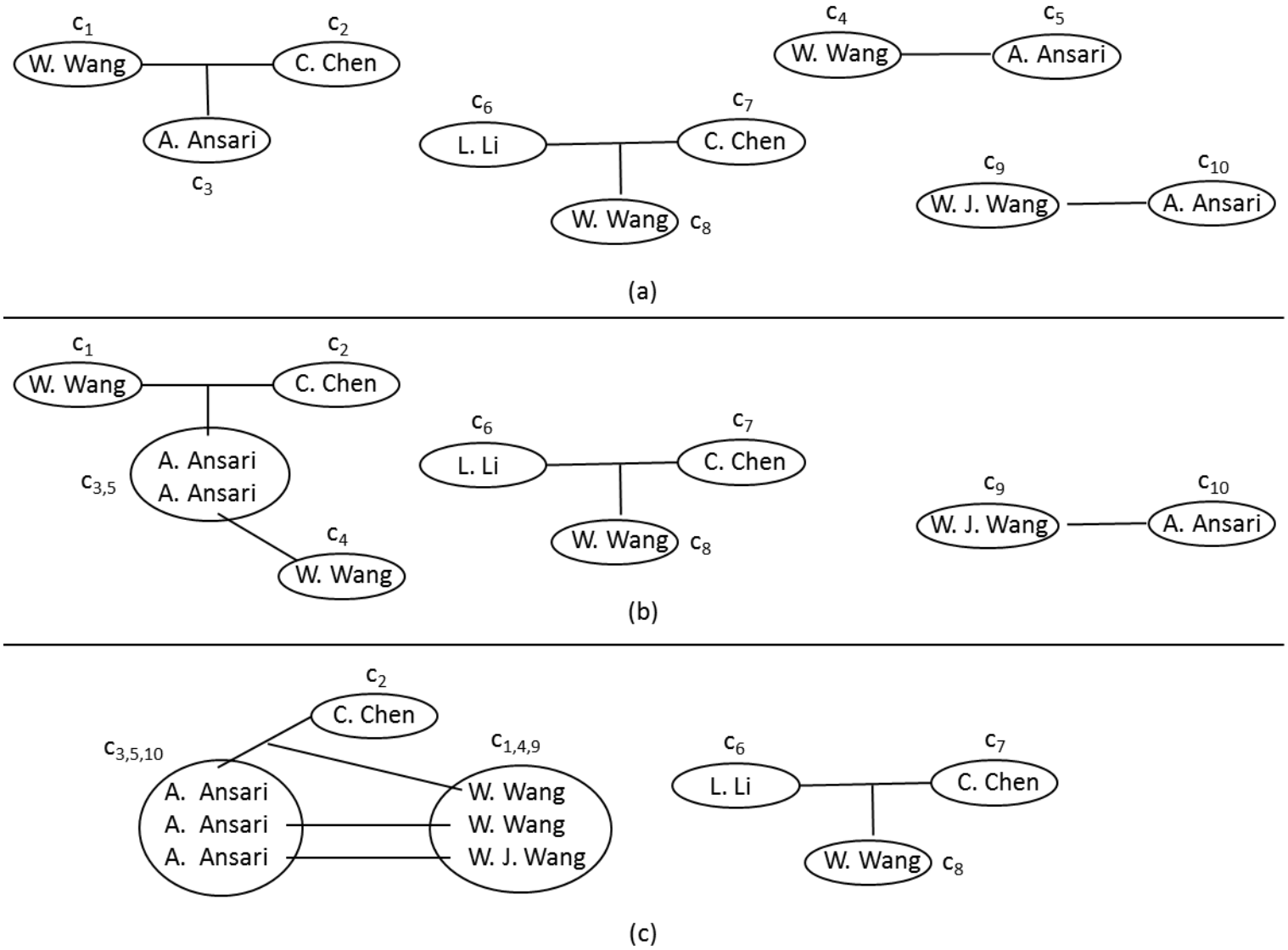
An Example of $sim_{neighbors}(A,B)$

- Assume a single relationship R on the graph edges
  - can generalize to the case of multiple relationships
- Let N(A) be the bags of the cluster IDs of all nodes that are in relationship R with some node in A
  - e.g., cluster A has two nodes a and a',
    a is in relationship R with node b with cluster ID 3, and
    a' is in relationship R with node b' with cluster ID 3
    and another node b'' with cluster ID 5
    ➔ N(A) = {3, 3, 5}
- Define $\mathbf{sim}_{neighbors}(\mathbf{A,B})$ =
  $$\text{Jaccard}(N(A),N(B)) = |N(A) \cap N(B)| \,/\, |N(A) \cup N(B)|$$

An Example of $sim_{neighbors}(A,B)$

- Recall that earlier we also define a Jaccard measure as
  - $\mathbf{JaccardSim}_{coAuthors}(\mathbf{a,b})$ = $|coAuthors(a) \cap coAuthors(b)| \,/\, |coAuthors(a) \cup coAuthors(b)|$
- Contrast that to
  - $\mathbf{sim}_{neighbors}(\mathbf{A,B})$ = $Jaccard(N(A),N(B)) = |N(A) \cap N(B)| \,/\, |N(A) \cup N(B)|$
- In the former, we assume two co-authors match if their "strings" match
- In the latter, two co-authors match only if they have the same cluster ID

An Example to Illustrate the Working of Agglomerative Hierarchical Clustering



Outline

- Problem definition
- Rule-based matching
- Learning- based matching
- Matching by clustering
- Probabilistic approaches to matching
- Collective matching
- Scaling up data matching

Scaling up Rule-based Matching

- Two goals: minimize # of tuple pairs to be matched and minimize time it takes to match each pair
- For the first goal:
  - hashing
  - sorting
  - indexing
  - canopies
  - using representatives
  - combining the techniques
- Hashing
  - hash tuples into buckets, match only tuples within each bucket
  - e.g., hash house listings by zipcode, then match within each zip

Scaling up Rule-based Matching

- Sorting
  - use a key to sort tuples, then scan the sorted list and match each tuple with only the previous (w-1) tuples, where w is a pre-specified window size
  - key should be strongly "discriminative": brings together tuples that are likely to match, and pushes apart tuples that are not
    - example keys: soc sec, student ID, last name, soundex value of last name
  - employs a stronger heuristic than hashing: also requires that tuples likely to match be within a window of size w
    - but is often faster than hashing because it would match fewer pairs

Scaling up Rule-based Matching

- Indexing
  - index tuples such that given any tuple a, can use the index to quickly locate a relatively small set of tuples that are likely to match a
    - e.g., inverted index on names
- Canopies
  - use a computationally cheap sim measure to quickly group tuples into overlapping clusters called canopines (or umbrella sets)
  - use a different (far more expensive) sim measure to match tuples within each canopy
  - e.g., use TF/IDF to create canopies

Scaling up Rule-based Matching

- Using representatives
  - applied during the matching process
  - assigns tuples that have been matched into groups such that those within a group match and those across groups do not
  - create a representative for each group by selecting a tuple in the group or by merging tuples in the group
  - when considering a new tuple, only match it with the representatives
- Combining the techniques
  - e.g., hash houses into buckets using zip codes, then sort houses within each bucket using street names, then match them using a sliding window

Scaling up Rule-based Matching

- For the second goal of minimizing time it takes to match each pair
  - no well-established technique as yet
  - tailor depending on the application and the matching approach
  - e.g., if using a simple rule-based approach that matches individual attributes then combines their scores using weights
    - can use short circuiting: stop the computation of the sim score if it is already so high that the tuple pair will match even if the remaining attributes do not match

Scaling up Other Matching Methods

- Learning, clustering, probabilistic, and collective approaches often face similar scalability challenges, and can benefit from the same solutions
- Probabilistic approaches raise additional challenges
  - if model has too many parameters ➔ difficult to learn efficiently, need a large # of training data to learn accurately
  - make independence assumptions to reduce # of parameters
- Once learned, inference with model is also time costly
  - use approximate inference algorithms
  - simplify model so that closed form equations exist
- EM algorithm can be expensive
  - truncate EM, or initializing it as accurately as possible

Scaling up Using Parallel Processing

- Commonly done in practice
- Examples
  - hash tuples into buckets, then match each bucket in parallel
  - match tuples against a taxonomy of entities (e.g., a product or Wikipedia-like concept taxonomy) in parallel
    - two tuples are declared matched if they match into the same taxonomic node
    - a variant of using representatives to scale up, discussed earlier

Summary

- Critical problem in data integration
- Huge amount of work in academia and industry
  - Rule-based matching
  - Learning- based matching
  - Matching by clustering
  - Probabilistic approaches to matching
  - Collective matching
- This chapter has covered only the most common and basic approaches
- The bibliography discusses much more