General Graph Refinement with Polynomial Delay

Jan Ramon & Siegfried Nijssen

K.U.Leuven

MLG, August 2007

Outline

Introduction
- Graph enumeration
- Earlier work

Dense refinement schemas

Results

Conclusions

Outline

Introduction - Enumeration

Pattern Mining:

- Given a language $\mathcal{L}$ and an (often 'anti-monotonic') predicate *interesting*, list all interesting patterns.

Many pattern mining algorithms perform essentially two tasks:

- Generating candidate patterns
- Checking interestingness of patterns (e.g. counting frequency)

Introduction - Enumeration

This paper deals with the first step:

- Given a (graph) language $\mathcal{L}$
- Enumerate all (graph) patterns $p \in \mathcal{L}$ from small to large
- ... allowing for suitable pruning
- ... and do not generate duplicates

In other words, we assume the predicate “interesting” to be evaluable efficiently.

Avoiding duplicates is non-trivial: for graphs, isomorphism is not known to be polynomial.

Introduction - Enumeration

Complexity measures:

- Polynomial time – total running time bounded by polynomial in input
- Polynomial delay – time needed for generating next solution is bounded by polynomial in size of input
- Incremental polynomial time – time needed for generating next solution is bounded by polynomial in size of input and output so far
- Output polynomial time – total running time bounded by polynomial in input + output.

Introduction - Enumeration

Applications of graph enumerating:

- Pattern mining (e.g. candidate generation)
- Combinatorics (e.g. counting)
- Search

Introduction - Earlier work

Earlier work

- Data mining and existing enumeration algorithms
- 'Simple' cases
- Graph listing (L.A. Goldberg)

Introduction - Earlier work

Literature:

- Systems:
  - gSpan, AGM, . . .
  - McKay's Nauty
- All devote much attention to efficient candidate generation
- Methods: Canonical forms, Joining operators, . . .
- But none of them proves polynomial delay.

Introduction - Earlier work

Special cases:

- Itemsets: Apriori runs with polynomial delay
- Free trees (Wright'86)
- Outerplaner graphs: (Horvath & Ramon'06)
- ...

Introduction - Earlier work

L.A.Goldberg's graph listing work

- One can enumerate all graphs of size at most n without duplicates with delay $O(n^6)$.
  - Relies on "most graphs are easy"
- If p is an allmost sure property, then one can enumerate all graphs that satisfy p (in the original paper including duplicates) with polynomial delay. Every FOL property is either allmost always true or allmost always false.
- For pattern mining, typically not all patterns are interesting. In particular, the interesting case is when there are only few interesting patterns.

Contribution

Given:

- a language $\mathcal{L}$ of graphs
- a dense refinement schema for $\mathcal{L}$
- an anti-monotonic constraint *interesting* (that can be evaluated efficiently)

we enumerate

- all *interesting* patterns
- with polynomial delay

Better?

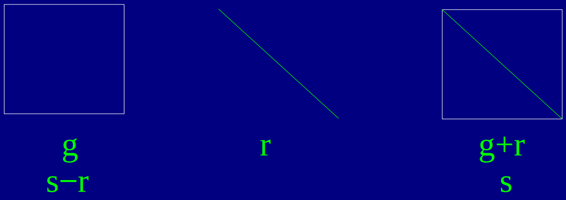
- More general than L.A.Goldberg ($\mathcal{L}$ or *interesting* can produce only 'difficult' graphs)
- More general than specialized methods
- More efficient (better proven assymptotic complexity) than gSpan, AGM, . . .

Outline

Refinement description

Definition

A refinement description is a pair $(V(r), E(r))$ of vertices and edges.



Refinement schema

Definition

A refinement schema is a pair (ρ^+, ρ^-) of functions from graphs onto sets of refinement descriptions. Then,

- $r \in \rho^+(g)$ are the downward refinement descriptions of g and $g + r$ are (downward refinement / specialisation / supergraph)
- $r \in \rho^-(g)$ are the upward refinement descriptions of g and $g - r$ are (upward refinement / generalisation / subgraph)
- Consistent: $r \in \rho^+(g)$ iff $r \in \rho^-(g + r)$
- Isomorphism-invariant= If $r \in \rho^+(g)$ and $g \equiv_\varphi h$, then $\varphi(r) \in \rho^+(h)$.

Refinement schema (ex)

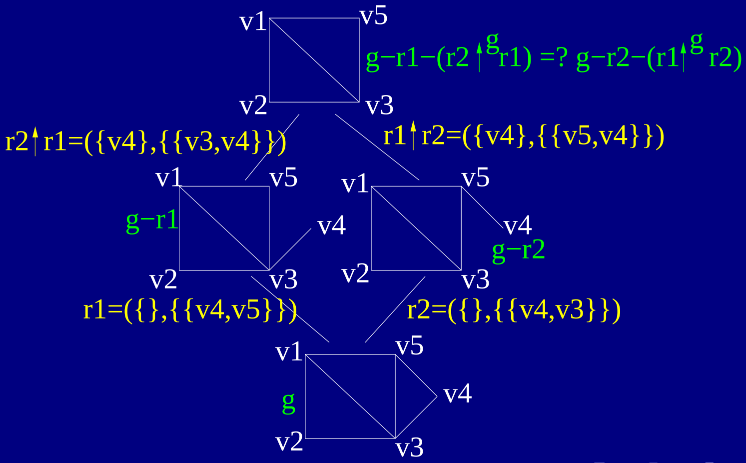
Connected graphs:

- $\rho^{+}(g)$: adding edge between two existing vertices or adding new vertex and connecting it to existing one.
- $\rho^{-}(g)$: removing vertex with degree 1 and its adjacent edge, or removing en edge belonging to a cycle.

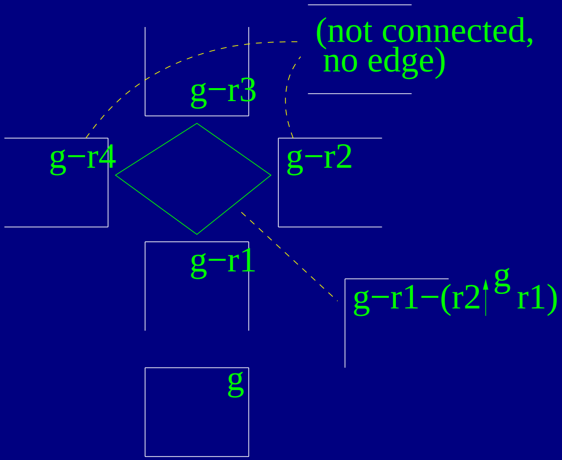
Combining refinements

$\uparrow$ is an operator 'lifting' an upward refinement. In particular, if $r_1, r_2 \in \rho^-(g)$ and removing r_2 from $g - r_1$ makes sense, $r_2 \uparrow^g r_1$ is the 'lifted' version of r_2 which can be applied to $g - r_1$.

Combining refinements



Graph of parents $GoP_{\rho^-,\uparrow}(g)$



Dense refinement schema

Definition

A dense refinement schema is a triple $(\rho^+, \rho^-, \uparrow)$ s.t.

- (ρ^+, ρ^-) is a refinement schema
- The graph $GoP_{\rho^-,\uparrow}(g)$ is connected.

This requirement has far-reaching implications, but still allows for dense refinement schemas for a wide range of graph classes.

Lemma

For every dense refinement schema, one can define a size function $|\cdot|$ such that for every graph g and $r \in \rho^+(g)$, $|g + r| = |g| + 1$.

Outline

- Introduction
  - Graph enumeration
  - Earlier work
- Dense refinement schemas
- Results
- Conclusions

Complexity theorem

Theorem

Consider a dense refinement schema $(\rho^+, \rho^-, \uparrow)$. If the following conditions hold:

- $|\rho^+(g)|$ *and* $|\rho^-(g)|$ *are* $O(|g|)$
- *For every* $r \in \rho^+(g)$, $|r \cap g|$ *is bounded by a constant*
- *The refinements* r *have an easy structure or* $|r|$ *is bounded by a constant.*

Then, our algorithm runs with polynomial delay

Data structure

- Our algorithm builds a data structure storing one representative for every isomorphism class.
- This may take exponential space in the input size (if the number of solutions is exponential), but for mining purposes this is not to be expected
- Given any graph, the data structure can be used to search the representative in polynomial time.

Key idea of algorithm

- We construct a candidate pattern from a parent.
- Since the refinement schema is dense, we can hop from one parent to the next one through grandparents.
- In this way we can avoid to generate children from different parents that are isomorphic.
- Finally, we can avoid isomorphic children from one single parent by (incrementally) computing automorphism groups.

What can we enumerate?

- 'Monotone' classes
- Hereditary classes with bounded degree
- Classes restricted to connected graphs

'Monotone'[1] classes

interesting is monotone iff for every graph G such that *interesting*(G), and for every subgraph $S \preceq G$, *interesting*(S).

- Minimal (efficient) frequency constraints under subgraph isomorphism.
- Maximal vertex & edge counts
- Maximal degree, treewidth, ...
- Forbidden subgraphs and minors
- ...

[1] : in different communities, monotone and anti-monotone are reversed

Complexity

Corollary

Let interesting be an antimonotonic predicate on the set of all (connected) graphs. Then, we can list all interesting graphs g with delay $O(|V(g)|^5)$.

Compare with L.A.Goldberg: enumerates all graphs with delay $O(|V(g)|^6)$

Hereditary classes with bounded degree

interesting is hereditary iff for every graph G such that *interesting*(G), and for every **induced** subgraph $S \preceq_i G$, *interesting*(S).

- Minimal (efficient) frequency constraints under induced subgraph isomorphism.
- Maximal degree, treewidth, edge count, vertex count
- Forbidden induced subgraphs, e.g. claw-free graphs (graphs not containing an induced claw $K_{1,3}$)
- ...

Connected graphs

We can combine both previous examples with the constraint that the graphs should be connected. (even though e.g. connectedness is not closed under taking subgraphs).

Outline

Conclusions

We proposed:

- A (more) general method
- to list all *interesting* graphs
- of a wide range of graph classes (dense refinement schema needed)
- with polynomial delay

Open problems

Theory:

- Enumerate graphs under induced subgraph isomorphism (e.g. claw-free graphs, no degree bound).
- How about homomorphism (aka theta-subsumption in ILP) ? (some negative results are already known).
- Larger refinement steps ? (e.g. for closed pattern mining)

Practice:

- Can a canonical form help?
- Experiments?

Questions or comments?