Variable Length Relationship Pattern Extensions

Teon Banek

March 4, 2019
About Me

Teon Banek

- Graduated from University of Zagreb, Faculty of Electrical Engineering and Computing
- Lead query engine developer at Memgraph
- Loves fencing, lasagne and black tea
- teon.banek@memgraph.com
About Us

Memgraph Ltd.

- Startup, founded in 2016
- Building a graph database
  - In-memory
  - High-performance
  - Distributed
- [https://memgraph.com](https://memgraph.com)
Contents

1. About
2. Filtering
3. Traversal Strategies
4. Conclusion
Relationship Pattern Syntax

- MATCH ()-[var? types? variableLength? properties?]-()
- types = '::' name ( '|' types )?
- variableLength = '*' min_bound? ( '..' max_bound )?
- properties = '{' ( key_name '::' value )* '}'
MATCH ()-[rs:FriendOf *2..3 {years: 3}]->()
MATCH ()-[rs:FriendOf *2..3 {years: 3}]->()
Standard Filtering

- We can filter on relationship type & property equality

What if want arbitrary expression predicate? We can traverse the obtained list.

For example:

```
ALL(r IN rs WHERE r.years > 2)
```

What if we want to filter on traversed nodes? Still possible, we have access to whole path.

But, the query can get complicated.
Standard Filtering

- We can filter on relationship type & property equality
- What if want arbitrary expression predicate?

What if we want to filter on traversed nodes? Still possible, we have access to whole path. But, the query can get complicated.
Standard Filtering

- We can filter on relationship type & property equality
- What if want arbitrary expression predicate?
  - We can traverse the obtained list.
  - For example: `ALL(r IN rs WHERE r.years > 2)`
We can filter on relationship type & property *equality*

What if want arbitrary expression predicate?
  - We can traverse the obtained list.
  - For example: `ALL(r IN rs WHERE r.years > 2)`

What if we want to filter on traversed nodes?
  - Still possible, we have access to whole path.
  - But, the query can get complicated.
Filter Lambda

- For example
  
  ```
  MATCH ()-[rs * (next_r, next_n | next_r.years > 2)]-()
  ```

- Essentially **ALL** embedded in pattern syntax
  
  ```
  MATCH ()-[rs *]-() WHERE ALL(next_r IN rs WHERE next_r.years > 2)
  ```
Filter Lambda

- For example
  ```plaintext
  MATCH ()-[rs * (next_r, next_n | next_r.years > 2)]-()
  ```
- Essentially **ALL** embedded in pattern syntax
  ```plaintext
  MATCH ()-[rs *]-() WHERE ALL(next_r IN rs WHERE next_r.years > 2)
  ```
- Lambda syntax `(' rel_var ',' node_var '|' expr ')'
  - `rel_var` — next relationship that we are about to traverse
  - `node_var` — next node that we are about to reach
  - `expr` — arbitrary expression which when evaluated produces **true** if we want to continue traversing
MATCH (:Alice)-[rs:FriendOf * (r, n | r.years + n.age > 34)]->()
MATCH (:Alice)-[rs:FriendOf * (r, n | r.years + n.age > 34)]->()
MATCH (:Alice)-[rs:FriendOf * (r, n | r.years + n.age > 34)]->()
Filter Lambda Conclusions

- We can match using arbitrary expression on each upcoming node and relationship.
- Covers the most common use case of regular expression like patterns.
- There’s still the issue of filtering based on remote parts of the traversed paths.
Depth-First Search

- Variable length expansion essentially performs a depth-first search.
- Each result is a list of relationships forming the current step of the algorithm.
MATCH (:Alice)-[*]->(:Daniel)
MATCH (:Alice)-[*]->(:Daniel)
MATCH (:Alice)-[*]->(:Daniel)
MATCH (:Alice)-[*]->(:Daniel)
Breadth-First Search

- One of the more common graph use cases is finding shortest paths.
- Breadth-first search seems like a logical addition.
- Syntax remains the same, but to enable the algorithm just append `bfs` to `*`.
- `MATCH ()-[var? types? '*bfs' properties? filterLambda?]-()`
MATCH (:Alice)-[*bfs]->(:Daniel)
MATCH (:Alice)-[*bfs]->(:Daniel)
BFS Conclusion

- BFS is nice and simple.
- What about path cost determined by something other than a relationship traversal?
Weighted Shortest Path Search

- Natural extension is assigning *weights* to relationships.
- Algorithm is enabled by appending `wShortest` to `*`.
- We also need syntax for tracking the weight.
Weighted Shortest Path Search

• Natural extension is assigning \textit{weights} to relationships.
• Algorithm is enabled by appending \texttt{wShortest} to \texttt{*}.
• We also need syntax for tracking the weight.
• \texttt{MATCH ()-[\texttt{var}? \texttt{types}? \texttt{'*wShortest'} \texttt{properties}? \texttt{weightLambda \texttt{weightVar filterLambda}?}]-()-}
• \texttt{weightLambda} — just like \texttt{filterLambda} but needs to produce a positive number as the current weight.
• \texttt{weightVar} — stores the final weight of the path.
MATCH (:Alice)-[*wShortest (r, n | r.years) total_weight]-(:Daniel)
MATCH (:Alice)-[*wShortest (r, n | r.years) total_weight]-(:Daniel)
Conclusion

- Syntactic additions are tied with pattern matching.
- No special functions doing pattern matching outside of patterns.
- This avoids surprising users.
Conclusion

- Syntactic additions are tied with pattern matching.
- No special functions doing pattern matching outside of patterns.
- This avoids surprising users.
- Implementation maps naturally to how we do pattern matching.
Thank You

• Thank you for your attention!
• Any questions?