Growing a Graph matching from a Handful of Seeds, experimental results

Ildar Nurgaliev, DS Master Student
E-mail: *nurildar9@gmail.com*

I. Abstract

Graph isomorphism problem is considered very hard in general [1]. Graph matching is a generalization of the classic graph isomorphism problem. By using only their structures a graph-matching algorithm finds a map between the vertex sets of two similar graphs. For this task there was implemented 3 algorithms that in common has heuristic approach: Percolation Graph Matching (PGM), ExpandOnce [4] with NoisySeed and the last one and more efficient ExpandWhenStuck [3]. Afterwards we conducted experiments under 4 kinds of random graphs and else 1 experiment on real graph in order to find relation between degree distribution and efficiency of these algorithms. The main algorithm that is observed is Expand-WhenStuck - the distinguishing feature of this algorithm is that it requires a dramatically smaller number of seeds and it is more stable than the other heuristic algorithms. For every algorithm there were conducted experiments on 4 types of random graph and one real. The final part is critics section.

II. Algorithms and implementation

The most scalable graph–matching approaches use ideas from percolation theory, where a matched node pair "infects" neighboring pairs as additional potential matches. This class of matching algorithm requires an initial seed set of known matches to start the percolation. The first algorithm that was implemented is Percolation Graph Matching (PGM). It is not such as powerful otherwise it generates only correct matching seeds. In this implementation PGM algorithm was slightly modified: the threshold r was reduced from 4 to 2. That leads to significantly more matches (wrong matches also accepted). In PGM algorithms, initial seeds play an important role because it does not generate them as the other two algorithms.

The next algorithm is ExpandOnce matching algorithm, which performs one round of expansion of the initial seed set. This helps the percolation process overcome the bottleneck due to a small seed set size. Afterwards it executes NoisySeed algorithm that does percolation in a graph. The main part that we should know about this algorithm is that it takes candidate seed for percolation randomly without any strategy and smart heuristic. That's why this algorithm could be considered as not efficient one.

The last one and more efficient algorithm is ExpandWhen-Stuck that is based on the robustness ideas developed [2]: means that it dynamically generates huge number of seeds for percolation with no big affection of wrong matching. Whenever there are no further pairs with score at least two, we add all the unused and unmatched neighboring pairs of all the matched pairs to the candidate pairs. In comparison with ExpandOnce, this algorithm has better performance for both real and random graphs, and its computational complexity is lower. Else on significant difference with respect to ExpandOnce is that ExpandWhenStuck uses smart choice of candidate seed for percolation: namely saying it takes a seed with max score and min difference in vertex degree of that seed.

III. Experiments

For experiments there were generated 4 random graphs (Erdős–Rényi, Watts-Strogatz, Barabási–Albert and Chung-Lu) and was chosen one real graph (Face-book graph). In order to observe behavior of the algorithms on random graphs, power-law graph and high clustered graph. The algorithms were written in c++ language with usage just stl library. All the experiments were conducted on the server with 125 GB RAM and processor Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz. For experiment we take one graph and do matching with itself with given some initial correct matches whose count depends on the algorithm.

A. Erdős–Rényi model (ER)

A graph was generated as a random choose from the collection of all graphs which have 2 nodes and 1 edges.
Parameters for generation:

1) $k = 2$ (links / per node)
2) $V = 10^6$
3) $deg_{avg} = 20$
4) $clique_{count} = \dfrac{V * deg_{avg}}{(k-1) * k}$

Parameters of generated graph:

- $V = 10^6$
- $E = 9\ 999\ 896$
- $dense = 0.00002$
- $deg_{avg} = 19.9998$

B. Barabási–Albert model

Parameters of generated graph:

- $V = 10^6$
- $E = 2\ 999\ 991$
- $dense = 0.000006$
- $deg_{avg} = 5.999$
- avg path length: $\ell \sim \dfrac{ln(N)}{lnln(N)} = 5.261$
- empirical cluster coef: $C \sim N^{-0.75} = 0.000032$

C. Watts and Strogatz model

Distribution properties of generated graph:

1) small-world properties
2) including short average path lengths
3) high clustering

Parameters of generated graph:

- $V = 10^6$
- $E = 20 * 10^6$
- $\beta = 0.4$
- $dense = 0.00004$
- $deg_{avg} = 40$

D. Chung-Lu model

Parameters for generation:

- $V = 200\ 000$
- $\beta = 0.991$

Chung-Lu graph was generated with algorithm below:

```
int x0 = 4
int x1 = (int)√V
for randomly choosen y:
```

$degrees[i] = (int)((x_1^{1-\beta} - x_0^{1-\beta}) * y + x_0^{1-\beta})^{1/(1-\beta)}$

Parameters of generated graph:

- $V = 100\ 000$
- $E = 3\ 585\ 8631$
- $dense = 0.000717$
- $deg_{avg} = 71.717$

E. Social circles: Face-book

- $V = 4\ 039$
- $E = 88\ 234$
- $dense = 0.01082$
- $deg_{avg} = 43.691013$
- Average clustering coefficient = 0.6055

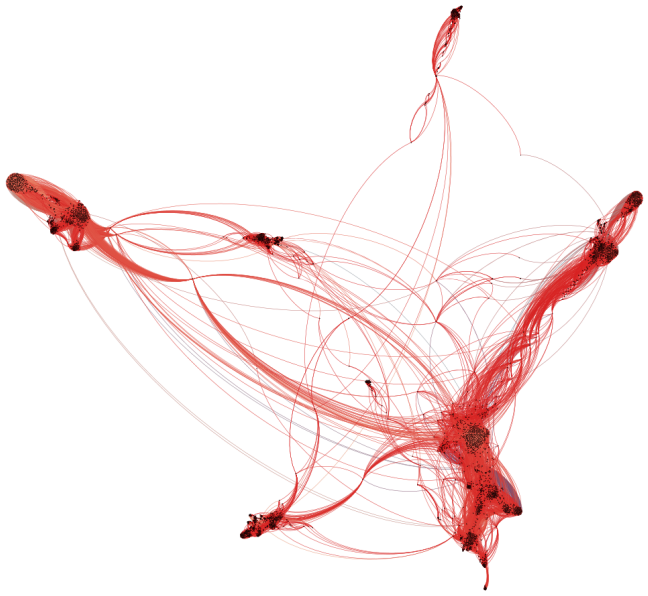

Fig. 1. Face-book circles graph ranked by degree. Contains a lot of sparse nodes

IV. Graphics

For every random graphs there were generated 3 graphics:

1) total matches - count of matches within two graphs. This metrics shows how an algorithm was expanded over a graph.
2) precision $= \dfrac{\wedge(\mathcal{M}^*)}{\wedge(\mathcal{M}^*) + \Psi(\mathcal{M}^*)}$
   - $\wedge(\mathcal{M}^*)$ - count of correct matches
   - $\Psi(\mathcal{M}^*)$ - count of incorrect.
3) recall $= \dfrac{\wedge(\mathcal{M}^*)}{n_{indent}}$
   - n_{indent} - number of nodes that are present in both graphs $G_{1,2}$ with degrees at least two.
4) there is no reason for F1-score (given precision and recall) and for count of correct matches (given recall).

A. Erdos-Renyi

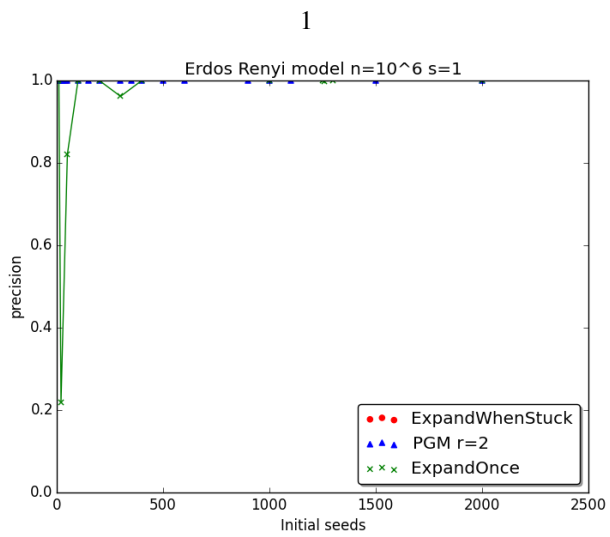

Fig. 2. **Precision**: ExpandWhenStuck shows good precision as PGM that is working without seed generation. ExpandOnce has bad precision at some points because of its random strategy of seed choose for expanding operation.
Time Execution:
* ExpandOnce - 45 minutes
* ExpandWhenStuck - 36 minutes
* PGM - 42 minutes

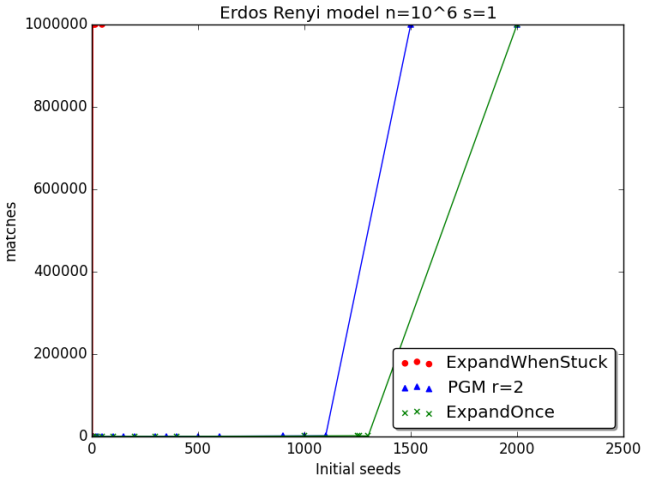

Fig. 3. **Total matches**: ExpandWhenStuck and PGM (r=2) have fully expanded the graph; ExpandOnce expands somehow with 2000 initial seeds.

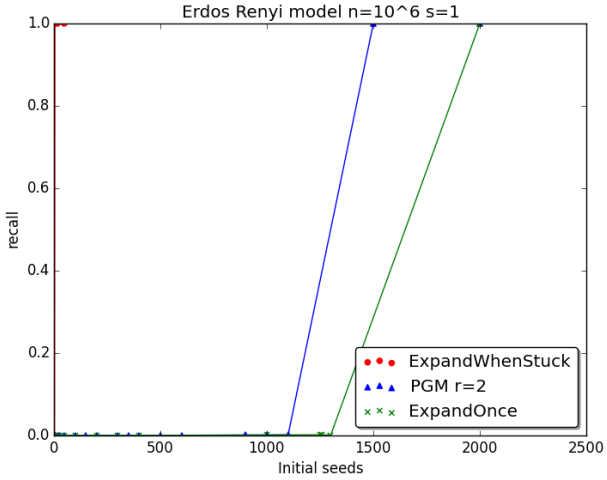

Fig. 4. **Recall**: The overall result of ExpandWhenStcuk is good while ExpandOnce has bad performance on random graph. PGM waits percolation moment that was found in 1200 initial seeds.

B. Barabasi

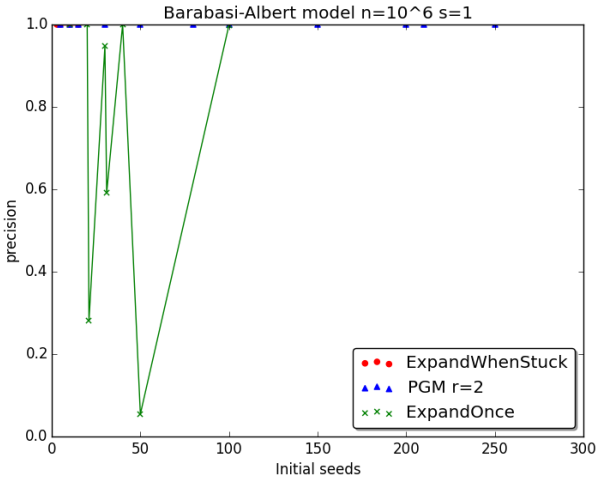

Fig. 1. **Precision**: ExpandWhenStuck and PGM have good performance; in most cases ExpandOnce has good results while it has bad exploration percentage.
Time Execution:
* ExpandOnce - from 22 minutes to 2 hours
* ExpandWhenStuck - 22 minutes in average
* PGM - 19 minutes in average.

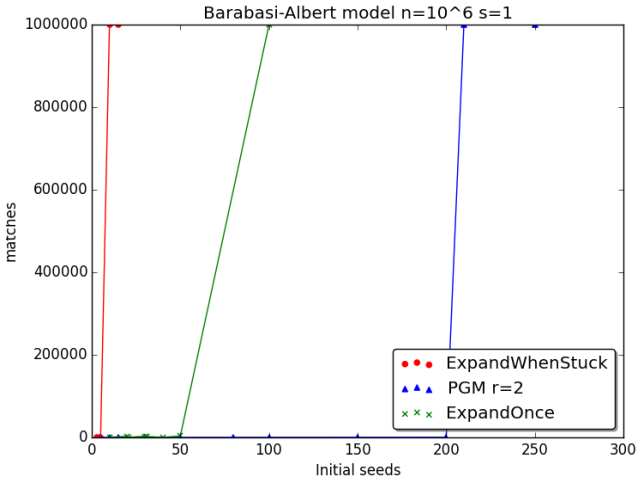

Fig. 2. **Total matches**: for ExpandOnce it is hard to guess appropriate number of seed and initial correct matches for good expansion. PGM starts expansion near 250 initial correct seeds while ExpandWhenStuck has good performance in this case starting from 10 initial seeds.

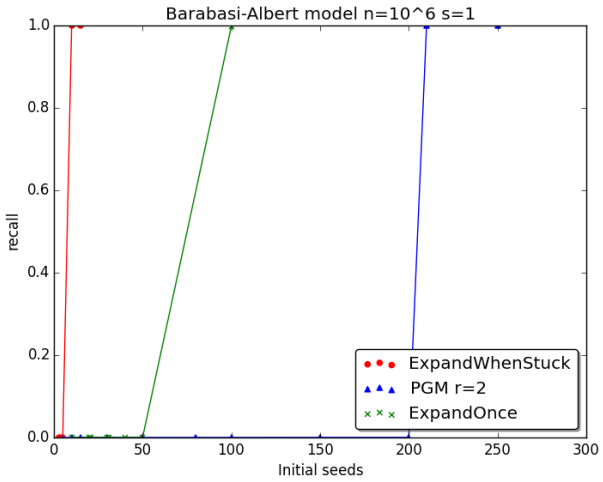

Fig. 3. **Recall**: PGM has good results when it starts good expansion through the graph. ExpandWhenStuck has good performance. ExpandOnce is unpredictable but in most cases is bad.

C. *Wattz-Strogatz*

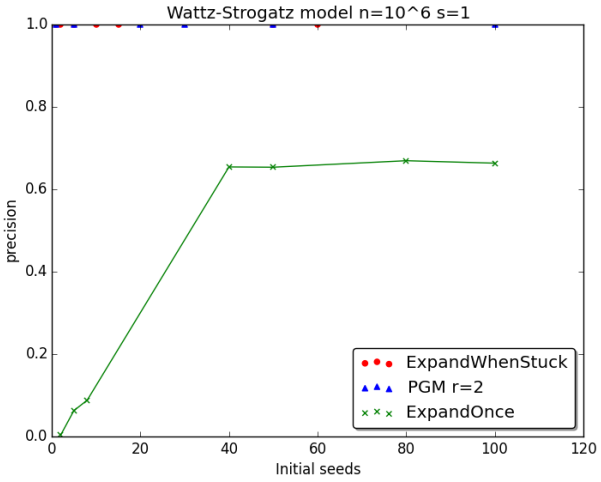

Fig. 1. **Precision**: Wattz-Strogatz graph has good dense and high average degree that's why the results of ExpandWhenStuck is very nice just with 1 initial correct seed. The results of PGM from 40 initial seeds are also significant.

Time Execution:
* ExpandOnce - 4 hours and 25 minutes for experiment with full exploration
* ExpandWhenStuck - 1 hour and 28 minutes for one experiment
* PGM - 1 hour and 20 minutes (50 seeds)

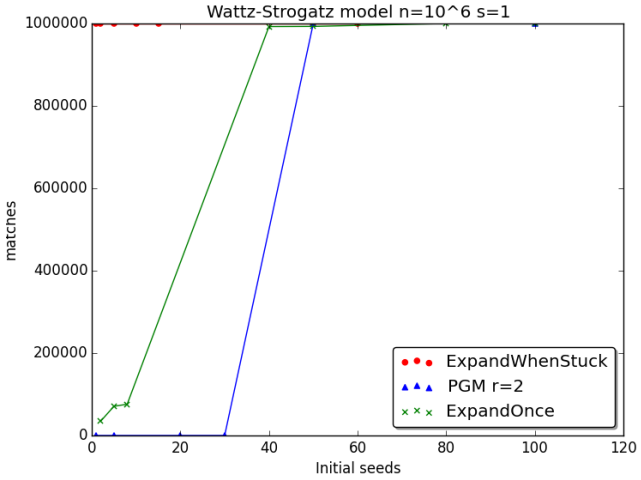

Fig. 2. **Total matches**: ExpandWhenStuck hasn't any problem for expansion even with one correct seed given. The triangle transitive helps a lot even for ExpandOnce to explore the whole graph having a little bit less number of initial seed than it is needed for PGM.

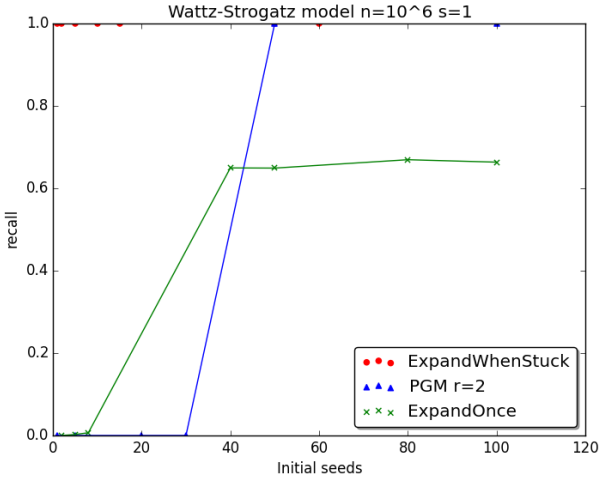

Fig. 3. **Recall**: ExpandWhenStack shows incredible results just with one initial seed, it says that high clustering coefficient and especially big amount of triangle transitivities helps a lot while percolation process.

D. Chung-Lu

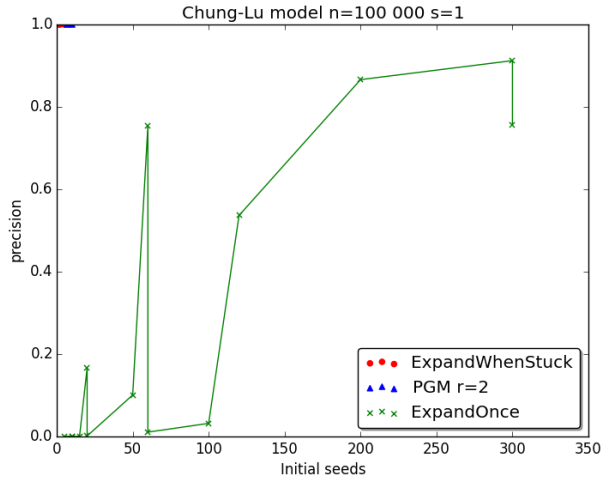

Fig. 1. **Precision**: Chung-Lu graph has significantly bigger dense and high average degree than the Wattz-Strogatz graph that's why the results of ExpandWhenStuck is perfect just with 1 initial correct seed. The results of PGM also need just handful count of initial seeds in order to get significant results. In other hand ExpandOnce with random strategy shows worse performance while executing.

Time Execution:
* ExpandOnce - the whole day (24-30 hours)
* ExpandWhenStuck - the whole day (24-30 hours)
* PGM - the whole day (24-30 hours)

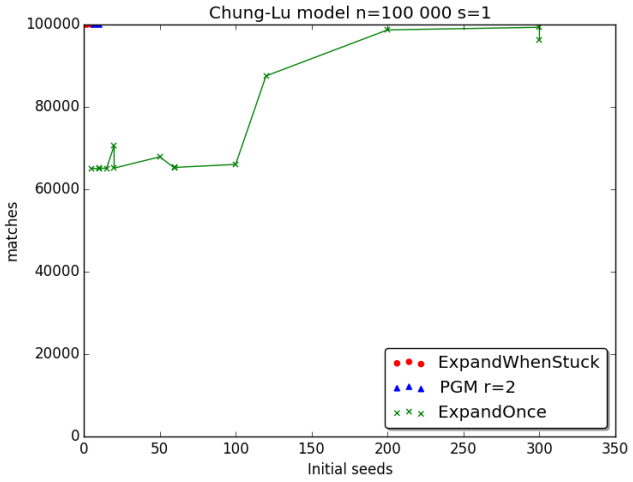

Fig. 2. **Total matches**: The dense and average degree allow PGM to fully percolate without any generated seed, ExpandWhenStuck fully explore the graphs just with one initial correct seed. ExpandOnce started good exploration just after 200 initial seed.

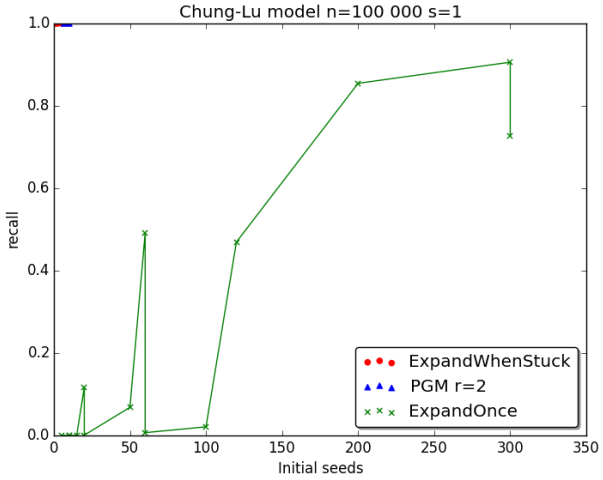

Fig. 3. **Recall**: For ExapndOnce it is hard to get stable good results while varying with a_c - number of generated seed from initial correct seed, that's why the recall for this algorithm looks like unpredictable. PGM and ExpandWhenStuck have nice results having good precision while the whole graph exploration.

E. Facebook circles

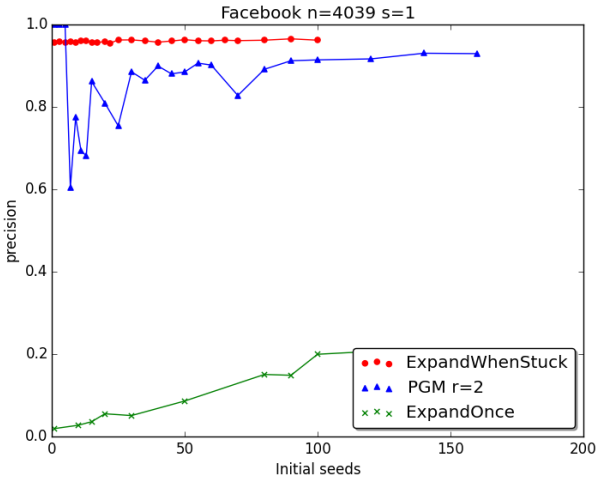

Fig. 1. **Precision**: the face-book graph has a lot of standalone vertexes with degree equal to one or two, that leads to stable not perfect result of ExpandWhenStuck, while ExpandOnce can't reach any remarkable results with high number of initial seeds. PGM varies from 80% to 90% after 20 initial seeds.

Time Execution:
* ExpandOnce - have not any good results, nothing to say about time
* ExpandWhenStuck - 1 minute
* PGM - 1 minute for best result of PGM presented in the graphic above.

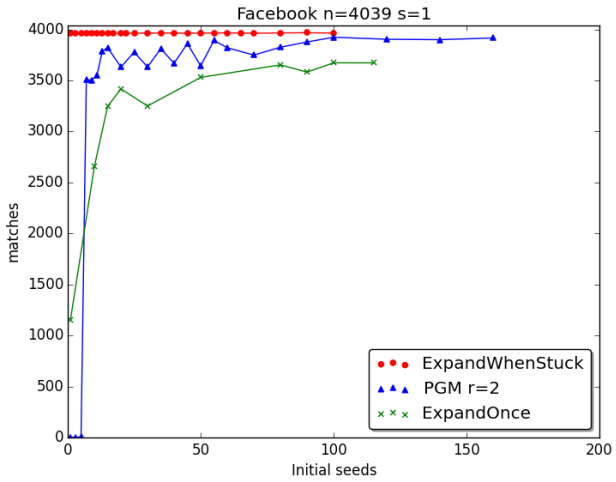

Fig. 2. **Total matches**: ExpandWhenStuck has stable results of exploration: 3800-3850 matched pairs. PGM exploration after 20 initial seed varies from 3300 - 3700 matches. For ExpandOnce it is hard to say about stability and to find good relation between initial seed and a_c - number of noisy seeds, there should be empirically chosen good relation between initial correct seeds and noisy seeds.

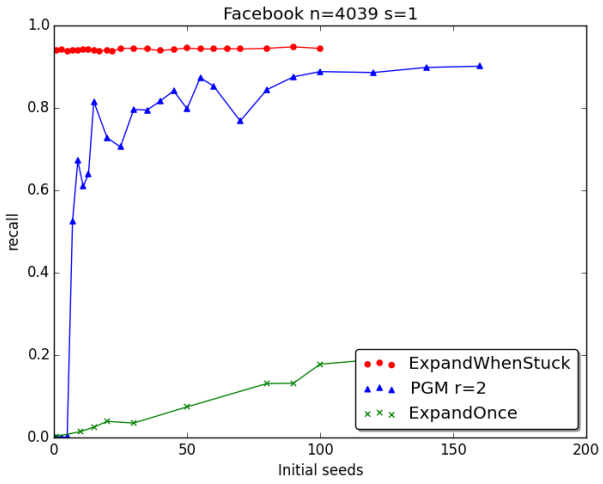

Fig. 3. **Recall**: there is no 100% result even for ExpandWhenStcuk. ExpandOnce can't percolate on this graph. The stability of PGM is hurt by no noisy seed generation.

V. Critics and proposition

These algorithms have some weak points, though the most obvious is impossibility to match a leave with one edge because of unaccepted two marks from at least two different edges. It could be avoided by reusing matched seed as candidate seed. For example every vertex that has a unique neighbor leaf would have special marking that it could be reused while being in matched set.

The worst case is when between two clusters is just one vertex that connects them just by two edges. In case of initial seeds in one cluster the second cluster would not be achieved and matched because of unmatched bridge vertex. In this case ExpandOnce has benefit with respect to ExpandWhenStuck and PGM, ExpandOnce firstly generates noisy seed by noisy seeds not through matched pairs as in ExpandWhenStuck. The resulting break-throughly received noisy seeds could percolate both clusters.

The given Theorem 1 (Robustness of NoisySeeds) in [3] is applicable, but it is very rare when we could guaranty good expansion of NoisySeed algorithm because we always unsatisfy condition given in this theorem as the graphs from conducted experiments do not satisfy the condition of the theorem. That's why the new algorithm for solving this task should use percolation theory with not random strategy for matching candidate seed, as it was done with success in ExpandWhenStuck in order to get a little bit more guarantee for good percolation.

ExpandWhenStuck has good **Time Execution** with respect to other two algorithms because of trade-off with space, ExpandWhenStuck generates a lot of candidates in every iteration. In case of big dense and deg_{avg} of a graph the space complexity will become huge as it was with Chung-Lu random graph.

REFERENCES

[1] M. R. Garey and D. S. Johnson. Computers and intractability: a guide to the theory of np-completeness. 1990.

[2] Svante Janson, Tomasz Luczak, Tatyana Turova, and Thomas Vallier. Bootstrap percolation on the random graph Gn,P. *Annals of Applied Probability*, pages 1989–2047, 2012.

[3] Ehsan Kazemi, S Hamed Hassani, and Matthias Grossglauser. Growing a Graph Matching from a Handful of Seeds. *Vldb 2015*, (ii):1010–1021.

[4] Lyudmila Yartseva and Matthias Grossglauser. On the performance of percolation graph matching. *Proceedings of the first ACM conference on Online social networks - COSN '13*, pages 119–130, 2013.