university of groningen / faculty of science and engineering / bernoulli institute

Tools for Gossip

Bachelor's Project Thesis

R.A. Meffert, s2702207, r.a.meffert@student.rug.nl
Supervisors: Dr. B.R.M. Gattinger

Abstract: The field of gossip theory – concerned with the way information spreads in networks of agents – has been the subject of research since at least 1927. However, with the introduction of new paradigms within this field, such as dynamic and distributed gossip, the field has become more complex. Information on dynamic (distributed) gossip is available in academic literature, but the protocols used – and in particular, their consequences – are not easily understood intuitively. To improve ease of understanding, this paper proposes a tool for visualising gossip graphs and working with arbitrary gossip protocols. The goal of the tool is, then, to allow students and researchers to more quickly gain an understanding of the most important aspects of gossip theory. **Add:** Something about the results of the survey, something like "To evaluate this goal, a survey was held among field experts. The survey results indicate users experience the tool as easy to use and useful in a study setting. Furthermore, feedback from the survey was used to improve the user experience and gain an understanding of which features are the most important to users."

Keywords: Dynamic Gossip, Elm

1 Introduction

Gossip protocols are protocols that describe the way rumors—or, more generally, secrets—are shared in multi-agent environments. The goal of the protocols is to communicate all secrets to all agents. A lot of research has been done in this field, starting with research on the spread of infectious diseases (Kermack & McKendrick, 1927).

The definition of the gossip problem generally used nowadays was first introduced in 1972 by Hajnal et al.* In short, agents are represented as nodes in a graph, with the edges representing a relation. This relation can be either the number relation N, representing that one agent has the "phone number" of another agent, and the secret relation S, representing that one agent knows the secret of another agent. Hajnal et al. proved that this can be done in $2n - 4$ calls, where n is the number of agents, if all agents know the numbers of all other agents—that is, the number relation N is *complete*.

The problem as formulated above requires the oversight of a central authority in order to know whether all agents know all secrets. However, there are many applications where this is not feasible or desirable (Citation Needed) †. Another problem is that it often cannot be guaranteed that all agents can contact all other agents. This has led to the sub-fields of *distributed gossip*, addressing the first issue, and *dynamic gossip*, addressing the second. The combination of these fields, where there is no overseer and not all agents can contact all other agents, is called *distributed dynamic gossip*.

This paper will describe a tool that has been developed to make it easier to explore distributed dynamic gossip. It is able to visualise the connections between agents, allows exploring the execution tree of different (semi-arbitrary) protocols and validate call sequences given a graph and/or a protocol.

1.1 Earlier work

1.1.1 Applications of gossip protocols

Gossip protocols have been in use in applications ranging from epidemics simulations (Citation Needed) to managing membership and detecting failure in distributed (database) processes (Das et al., 2002; DeCandia et al., 2007). These processes bear some resemblance to distributed dynamic gossip, since membership is managed using gossip protocols: Whenever a member joins or leaves the network, this is spread using gossip, like the way numbers are shared in dynamic gossip. The difference lies in the fact that in this application, the secrets and numbers are not shared together per se; they are exchanged independently.

1.1.2 Visualisation tools

A researcher or student interested in dynamic gossip will probably start looking for some way to gain

*TODO: Tijdeman (1971) might have been earlier, but I have not been able to find a pdf of that paper. It is referenced in Van Ditmarsch et al. (2018) though. According to Faustine Maffre at her github the problem was formulated in 1970 in an unpublished work.

†TODO: maybe find a citation for this? Or just explain it better

an intuitive understanding of the field. At the time of writing, a couple of options are available. However, most of these require either an established understanding of the field and/or proficiency using command-line tools. In this section, these tools will be discussed in order to highlight some of the aspects the tool proposed in this paper attempts to improve.

The first two tools are tools by Malvin Gattinger (the supervisor of this Bachelor's project): one works in the browser (Gattinger, n.d.), the other is a command-line tool (Gattinger, 2017). The first tool, `WebGossip`, is able to visualise gossip graphs based on user input. While the input format for that tool is different from the one used in this tool (for an explanation of the format for the tool described in this paper, see Section 2.1.1), it is similar in that it also takes text input and displays the gossip graph graphically. The tool also generates code for use in LaTeX papers. The main differences between this tool and the tool described in this paper can be seen in Table 1.1

Finding call sequences and graph visualisation, command line (Gattinger, 2017)

Gossip spreading visualisation, web (Moelker, 2016)

Gossip graph visualisation, web (Gattinger, n.d.)

(relevant?) Generating Planning Domain Definition Language code for gossip protocols, command line (Maffre, 2016)

1.2 Notation

This paper takes its notation from Van Ditmarsch et al. (2018) and Van Ditmarsch et al. (2019). The notation used is explained below.

Definition 1 (Gossip graph)**.** Let A be a set of agents $\{a, b, \dots\}$. Two directed binary relations on A are defined: $N, S \subseteq A^2$. The first denotes the *number* relation, the second the *secret* relation. A gossip graph G is then defined as the a triple (A, N, S).

Definition 2 (Relations on gossip graphs)**.**

Definition 2.1 (Binary relation)**.** Pxy says that agent x has relation P to agent y.

Formally: $(x, y) \in P$

Definition 2.2 (Identity relation)**.** I_A is the set of relations where all agents have a relation to themselves.

Formally: $\{(x, x) \mid x \in A\}$

Definition 2.3 (Converse relation)**.** P^{-1} is the set of relations in P, but with their direction switched.

Formally: $P^{-1} = \{(x, y) \mid Pyx\}$

Definition 2.4 (Composition relation)**.** The composition of the relations P and Q is a new relation such that the tuple (x, z) is in said new relation if and only if there exists another agent y such that $(x, y) \in P$ and $(y, z) \in Q$.

Formally: $P \circ Q = \{(x, z) \mid \exists y((x, y) \in P \wedge (y, z) \in Q)\}$

Definition 2.5. P_x represents the agents x has relation P with.

Formally: $P_x = \{y \in A \mid Pxy\}$

Definition 2.6. P^i represents the ith composition of relation P with itself.

Formally:

$$P^i = \begin{cases} P & \text{for } i = 1 \\ P^{i-1} \circ P & \text{for } i > 1 \end{cases}$$

Definition 2.7. P^* represents the set of binary relations that are obtained through repeated relation composition of P with itself.

Formally:

$$\bigcup_{i \geq 1} P^i$$

Definition 3 (Calls)**.** A call from an agent x to an agent y is denoted xy.

Definition 3.1 (Call sequence)**.** Call sequences are denoted using lowercase greek letters, starting with σ. ϵ denotes the empty sequence. $\sigma ; \tau$ denotes the concatenation of two sequences.

Definition 3.2 (Subsequence)**.** σ_x denotes all calls in the sequence σ containing x.

Formally:

$$\sigma_x = \begin{cases} \epsilon & \text{if } \sigma = \epsilon \\ \tau_x ; uv & \text{if } \sigma = \tau ; uv \wedge (x = u \vee x = v) \\ \tau_x & \text{if } \sigma = \tau ; uv \wedge \neg(x = u \vee x = v) \end{cases}$$

Definition 3.3 (Effect of a call on a relation)**.** P^{xy} denotes the state of relation P after call xy.

Formally: $P^{xy} = P \cup (\{(x, y), (y, x)\} \circ P)$

Definition 3.4 (Effect of a call on a gossip graph)**.** G^{xy} denotes the state of gossip graph G after call xy.

Formally: $G^{xy} = (A, N^{xy}, S^{xy})$

Definition 3.5 (Effect of a call sequence on a gossip graph)**.** G^σ denotes the state of gossip graph G after call sequence σ.

Formally:

$$G^\sigma = \begin{cases} G & \text{if } \sigma = \epsilon \\ (G^{xy})^\tau & \text{if } \sigma = xy ; \tau \end{cases}$$

Definition 3.6 (Call sequence possibility)**.** A call sequence σ is possible on a gossip graph G iff for all calls $c_i \in \sigma$ such that $\sigma = c_i ; \tau$, the call is possible, i.e. Nc_i

Table 1.1: Comparison between the tool described in this paper and other tools available

Feature	GoMoChe	WebGossip	This paper
Gossip graph visualisation	Yes	Yes	Yes
LaTeX output	No	Yes	Planned
Call execution	No	No	Yes
Call sequence execution	No	No	Yes
List allowed calls	No	No	Yes
List allowed sequences	Yes	No	No
Protocols	LNS, CO, CMOWLOG, pig, ANY, NO	N/A	LNS, CO, wCO, SPI, ANY, TOK, custom

Definition 3.7 (Call permissibility)**.** A call xy is permitted on a gossip graph G^τ with a (possibly empty) history τ iff τ is possible on G, $x \neq y$, $N^\tau xy$ and $G^\tau \models \pi(x, y)$.

Definition 3.8 (Call sequence permissibility)**.** A call sequence $\sigma\,;xy$ is permitted on a gossip graph G^τ with a (possibly empty) history τ iff xy is permitted on $(G^\tau)^\sigma$ and σ is permitted on G^τ.

Definition 4 (Gossip Protocol)**.** A gossip protocol P is a combination of a protocol condition $\pi(x, y)$ and a non-deterministic algorithm (for examples see van Ditmarsch et al., 2018, p. 708).

Definition 4.1 (Protocol condition)**.** $\pi(x, y)$ represents a protocol condition, that is, a condition that needs to be satisfied before a call can be made. This report takes this condition, like van Ditmarsch et al. (2018), as a boolean combination of the following constituents: $S^\sigma xy$, $xy \in \sigma_x$, $yx \in \sigma_x$, $\sigma_x = \epsilon$, $\sigma_x = t\,;xz$ and $\sigma_x = t\,;zx$.

2 Method

To provide as much useful information as possible, the tool needs to provide information on a number of aspects of gossip. The first aspect that will be discussed is gossip graph visualisation, followed by protocol execution, call sequence validation and call (sequence) execution. First, a reasoning will be given behind the programming language of choice. Then, these aspects are discussed in terms of their meaning in both the context of gossip theory and of their implementation. Lastly, some time will be spent to explain how these aspects are integrated into the application.

2.1 Implementation

This project uses Elm, a statically typed functional programming language for web development with its roots in Functional Reactive Programming (Czaplicki & Chong, 2013). This has several advantages:

1. Implementing mathematical functions is more natural in functional languages because functions in Elm are pure;
2. Elm is compiled to standard Javascript. Since all modern browsers support Javascript, this ensures the application is cross-platform;
3. Elm does static type checking while compiling, ensuring type safety and thus reducing runtime errors;

Another advantage of using a functional language is that much of the notation introduced in section 1.2 can be translated fairly directly. For example, to evaluate protocol conditions, the last call in a call sequence needs to be checked. In mathematical notation this is represented as $\sigma_x = \tau\,;xy$ ("the last call in the sequence of calls containing x was a call made by x to another agent y"). This can be represented in Elm quite naturally, as can be seen in Listing 1.

Listing 1: $\sigma_x = \tau\,;xy$ in Elm.

```
lastFrom agent sigma_x =
    case reverse sigma_x of
        [] ->
            False

        (x, y) :: tau ->
            x == agent
```

This states that if $\sigma_x = \epsilon$, the condition is false. Otherwise, it checks to see if the last call was made by x, and returns true if that is the case.

2.1.1 Gossip graph visualisation

One of the easiest ways to visualise connections in a network is by using a graph. Therefore, the application is able to visualise gossip graphs based on user input. The input is provided by users in text, and uses a format based on the one used in the appendix of van Ditmarsch et al. (2019). However, since no complete grammar of that format is given,

the format has been adapted to be a bit more permitting than could be expected based purely on its description.

The basic idea behind the format is that every agent is represented by some letters, which in turn represent agent names. Uppercase letters represent the Secret relation S and lowercase letters represent the Number relation N. Because every agent knows their own secret ($I_A \subseteq S$), it is possible to find names for all agents if and only if the input represents a valid gossip graph. That is, every agent is represented by some letter segment such that the number of segments is equal to the number of unique agent names, and every letter segment contains a secret relation that allows it to be uniquely identified. This procedure is described in more detail in Algorithm 1.

Once the agent names have been found, it is possible to parse the relations from the input string. Algorithm 1 assumes the first agent name it finds represents the first agent. Thus, the first segment represents the relations of that agent.

This interpretation of the format is likely a bit more permissive than the original format. For example, in the original format, the names of agents always follow alphabetic order – agent 1 is A, agent 2 is B, et cetera. Furthermore, the text segments in the original format are always ordered alphabetically. These requirements are not present in the implementation, so an input like `Xyz Yzx Zxy` is allowed. However, the tool does generate a so-called "canonical input" which does follow these rules.

2.1.2 Execution calls

Single calls When a gossip graph is given by the user, the tool allows them to execute calls and call sequences. Because the possible calls depend on the active protocol, a list of possible calls is given for the currently active protocol. The possible calls are found by checking for call permissibility as defined in Definition 3.7. When the user clicks one of these calls, the call is executed. The call then happens according to Definition 3.4.

Call sequences It is also possible to execute entire call sequences. The user is able to enter a call sequence using a text input according to the format given in Definition 3.1. This sequence is then checked for possibility (Definition 3.6) and permissibility (Definition 3.8). It should be noted that these checks are done in the order mentioned here. This means that the user receives feedback on their input when they enter a call sequence that is not possible on the current gossip graph. This has the added advantage that the system need not check for permissibility if the user enters a call sequence that is impossible. Additionally, because possibility is a requirement for permissibility and its check is executed before the permissibility check, this check does not need to occur in the check for permissibility; if the system reaches the permissibility check, the call sequence is guaranteed to be possible.

Custom protocols To allow more flexibility in using protocols, the tool described in this paper allows the user to create their own gossip protocols. This is done through a drag-and-drop interface, in which users can arrange boolean constituents or groups of boolean constituents. These constituents are then interspersed with either logical conjunction ($\wedge$) or logical disjunction ($\vee$) (see also Figure 2.1).

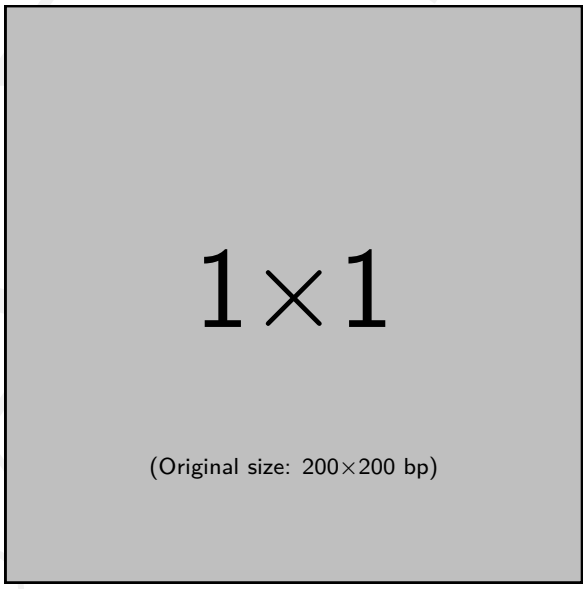

Figure 2.1: The interface for creating custom gossip protocols (placeholder)

The creation of custom protocols is based on the language $\mathcal{L}_B(V)$ of boolean formulas, given in the Backus-Naur form: $\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi \wedge \varphi$, where $p \in V$. This is extended with the logical disjunction by virtue of the formula $\varphi \vee \psi := \neg(\neg\varphi \wedge \neg\psi)$. For this tool, the vocabulary V is the set of constituents of protocol conditions as defined in Definition 4.1.

2.1.3 Execution tree and call history

Calls are recorded and it is possible to time travel between states. Something about multiple branches

2.2 Evaluation

Survey blah blah blah

3 Discussion

3.1 Further research

At the time of publication of this report, several planned features – some planned some time ago, some arising from the results of the survey – have not been implemented yet. These features are being worked on, but are included for completeness in this version of the paper in case any of them end up not being completed by the time this report is submitted.

- The custom protocol creation (Section 2.1.2) is not finished yet, and still lives in a separate git repository;
- It is not possible to create execution trees yet, though a linear call history is generated after executing calls;
- Clicking calls (Section 2.1.2) does not execute them yet;
- The rendered graph sometimes jumps when changing it;
- It is not possible to use the input format used in `WebGossip` (Gattinger, n.d.);
- Impossible secret distributions (i.e. gossip graphs that cannot be generated from an initial gossip graph with just $S = N = I_A$) are allowed;
- User input is not persisted; when the page is refreshed, all input is gone.

3.1.1 Possible extensions of the tool

The current version of the tool allows user to model dynamic gossip protocols as defined in van Ditmarsch et al. (2018). However, there is more to gossip than what is described in that paper. In this section, other papers on (dynamic) gossip are presented which introduce an additional concept. To make the tool more broadly usable, it could be interesting to consider adding some or all of these aspects to the tool.

The first possibility considered here is that of meta-knowledge or epistemic information, such as in Herzig and Maffre (2017). That means that agents can not only know secrets and numbers, but also whether other agents know a secret or number. This allows for more complex protocols.

Another idea is to include a temporal aspect, such as in Cooper et al. (2019) (note that this paper also includes epistemic information). This means that agents cannot always be reached. The paper gives as examples for this problem communication between geostationary satellites that might not always be able to 'see' each other, or a phone with a battery that sometimes does not have enough charge to communicate. This might be interesting, because this means that instead of calls being either always possible or always impossible, they might *sometimes* be possible or impossible as well.

A related problem is introduced by van den Berg (2018): normally, it is assumed that agents in dynamic gossip are reliable and tell the truth about the secrets they hold. However, if agents lie, many known properties of dynamic gossip suddenly no longer hold. By introducing unreliable agents, a new goal is added to the gossip system: *identify the unreliable agents*.

Another modification to secrets could be taken from Demers et al. (1988), who propose a system in which all secrets have a 'hotness'. This 'hotness' indicates how often the secret has been spread. This can then be used as a heuristic to determine whether it is useful to communicate the secret. This might be interesting because in the current system, an agents communicates all their secrets in a call. However, in a real-life application, this might not be desired: In large networks, communicating all secrets all the time might lead to large amounts of data being sent even though this might be data the agents receiving the secrets already have.

- Meta-knowledge (Herzig & Maffre, 2017)
- A temporal aspect (Cooper et al., 2019, also includes meta-knowledge)
- A decay factor ('hotness') to secrets (Demers et al., 1988)
- Asynchronous calls (Citation Needed)
- Unreliable agents (van den Berg, 2018)
- UI adaptation for mobile devices
- Add successfulness functionality, such as generating successful call sequences if available

References

Cooper, M. C., Herzig, A., Maris, F., & Vianey, J. (2019). Temporal epistemic gossip problems. In M. Slavkovik (Ed.), *Multi-agent systems* (pp. 1–14). Cham, Springer International Publishing. https://doi.org/10.1007/978-3-030-14174-5_1

Czaplicki, E., & Chong, S. (2013). Asynchronous functional reactive programming for GUIs. *SIGPLAN Not.*, *48*(6), 411–422. https://doi.org/10/f45mkb

Das, A., Gupta, I., & Motivala, A. (2002). SWIM: Scalable weakly-consistent infection-style process group membership protocol, In *Proceedings international conference on dependable systems and networks*. International Conference on Dependable Systems and Networks, Washington, DC, USA, IEEE Comput. Soc. https://doi.org/10/dfvrrz

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., & Vogels, W. (2007). Dynamo: Amazon's highly available key-value store, In *Proceedings of twenty-first ACM SIGOPS symposium on operating systems principles*, Association for Computing Machinery. https://doi.org/10/bj8wqc

Demers, A., Greene, D., Houser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., & Terry, D. (1988). Epidemic algorithms for replicated database maintenance [8]. *ACM SIGOPS Operating Systems Review*, *22*(1), 8–32. https://doi.org/10/fmhpvg

Gattinger, M. (n.d.). *WebGossip*. Retrieved December 24, 2020, from https://w4eg.de/malvin/illc/webgossip

Gattinger, M. (2017). *Explicit epistemic model checking for dynamic gossip*. Retrieved December 24, 2020, from https://github.com/m4lvin/GoMoChe

Hajnal, A., Milner, E. C., & Szemerédi, E. (1972). A cure for the telephone disease. *Canadian Mathematical Bulletin*, *15*(3), 447–450. https://doi.org/10/cpr4cv

Herzig, A., & Maffre, F. (2017). How to share knowledge by gossiping. *AI Communications*, *30*(1), 1–17. https://doi.org/10/f94qxh

Kermack, W. O., & McKendrick, A. G. (1927). A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society of London*, *115*(772), 700–721. https://doi.org/10.2307/94815

Maffre, F. (2016). *Gossip problem PDDL-generator*. Retrieved December 24, 2020, from https://github.com/FaustineMaffre/GossipProblem-PDDL-generator

Moelker, R. (2016). *Visualization of gossip protocol simulation*. Retrieved December 24, 2020, from https://github.com/RRMoelker/gossip-visualization

van den Berg, L. (2018, January 12). *Unreliable gossip* (Master Thesis). Universiteit van Amsterdam. https://eprints.illc.uva.nl/1597/1/MoL-2018-01.text.pdf

van Ditmarsch, H., Gattinger, M., Kuijer, L. B., & Pardo, P. (2019). Strengthening gossip protocols using protocol-dependent knowledge. *Journal of Applied Logics*, *6*(1), 157–203.

van Ditmarsch, H., van Eijck, J., Pardo, P., Ramezanian, R., & Schwarzentruber, F. (2018). Dynamic gossip. *Bulletin of the Iranian Mathematical Society*, *45*(3), 701–728. https://doi.org/10/cvpm

Algorithm 1: Finding agent names

```
   input  : A list of letter sequences called segments
   output : A list of agent names
 1 names ← {}
 2 foreach segment ∈ segments do
 3 |   possibleNames ← GetUppercaseLetters(segment)
 4 |   UniqueNameFound ← false
 5 |   foreach name ∈ possibleNames do
 6 |   |   if name ∉ names then
 7 |   |   |   names ← {name } ∪ names
 8 |   |   |   UniqueNameFound ← true
 9 |   |   end
10 |   end
11 |   if ¬UniqueNameFound then
12 |   |   ShowError()
13 |   end
14 end
15 return names
```

A Appendix