Several experiments in elementary self-modifying protocol games, such as Nomic

Gerard A.W. Vreeswijk

University of Limburg, Department of Computer Science (FdAW),
P.O. Box 616, NL-6200 MD Maastricht, The Netherlands
Email address: vreeswijk@cs.rulimburg.nl
Phone: +31-43 88 2021 (work) +31-1720 40953 (home)

ABSTRACT

There is the idea that network protocols can be changed by the computers that use them. Such protocols are called open protocols. At this moment, open protocols do not exist. To better understand the principles of open protocols, researchers build (and play with) simplified models, referred to as self-modifying protocol games. Nomic, for example, is such a game. Most of the activities in the area are informal, and ‘fail’ to materialize into results that can be applied to the construction of open protocols. This paper is the 3rd of a sequence of papers that tries to approach the field more formally. It contains a number of experiments in elementary self-modifying protocol games, of which the results are meant to provide additional insight in the development of more complicated such games.

1. Introduction

It is expected that, in the near future, computers will use communication protocols that allow them to change the protocol that they are using.

Loosely speaking, such protocols already exist right now. For example, the current generation of high-speed modems negotiate their data-exchange protocol before establishing a definite connection. It goes without saying that the negotiation has it own protocol, too. If the protocol for negotiation and the data-exchange protocol are considered one, we have a first example of a device that uses a protocol to define the protocol that it will be using.¹

This paper is part of a larger research project, that aims at a theory of open communication protocols in distributed systems (Vreeswijk, 1995a; Wan et al., 1995).²,³,⁴ An open communication protocol is a collection of rules of communication, that is continuously altered and maintained through communication. The work is important for the next generation of distributed knowledge-based systems, distributed problem-solving systems (DPS), distributed intelligent information retrieval systems, and cooperating multi-agent systems (MAS). The idea is that

¹Another example is the Point-to-Point Protocol (PPP), proposed by (Simpson, 1992).⁵ The PPP provides a standard method of encapsulating Network Layer protocol information over point-to-point links. It also defines an extensible Link Control Protocol, which allows negotiation of an Authentication Protocol for authenticating its peer before allowing Network Layer protocols to transmit over the link.⁶ Further details of APs, such as PAPs and CHAPs, can be found in (Simpson et al., 1992).
distributed computer systems of the future will use their communication protocols in much the same way as we do in ordinary meetings. Just like us, intelligent components of distributed systems will attempt to define or alter the protocol with which they are communicating. The reasons to do so are the same reasons that participants of a meeting use when they try to change the standing order, and the same reasons that members of parliament use in their attempts to change the constitution. Members of parliament try to change the constitution to govern the decision making process in parliament; participants of a meeting raise points of order to control, direct, or strongly influence the actions and conduct of what follows in the rest of the meeting. The motives that lie in turn behind these activities are considerations of time, process and, sometimes, policy. For instance, it is policy to adopt a law that enables to railroad several controversial laws through. Moreover, it is to the benefit of due process to adopt a rule that each participant speaks at least once, and it saves time to adopt a rule that restricts the speaking time of each participant. In all these cases, the protocol is changed to exercise continuous influence on the rules of interaction, so as to ensure that the flow of the messages between participants is mediated in a fair, effective, and economic way.

Thus,

1. All important human forms of interaction proceed by means of protocols that allow modifications on the run.

2. People do this for reasons that will be equally important for the next generation of distributed computer systems, namely, to ensure a smooth operation "for the good of the individual and the whole".

3. Since such protocols protocols for computers do not yet exist, we will have to develop them.

Writing communication software that is able to alter and adapt itself on the run is a difficult undertaking. There are several problems. Some of these problems are fundamental, and arise as a result of the paradoxes of self-amendment and reflexivity. Other problems are of a more practical nature.

- **Fundamental problems.** How do agents (or distributed knowledge-based systems) communicate if there is no communication standard to begin with? How is it possible to write software that is intended partially to rewrite itself? How do we separate the different layers of a protocol, without losing the idea of self-amendment?

- **Problems pertaining to specification and verification.** By definition, an open protocol is subject to change, and therefore undergoes a number of transformations in the course of its existence. How do we ensure the correctness of a changing protocol? How do we ensure that a dynamic protocol does not lapse into dead states?

- **Problems pertaining to 'computational legislation'.** There may be loopholes in the current state of an open protocol. In civil legislation, loopholes in the law are sometimes exploited, not only to daily matters, but also to legislation itself. For example, loopholes in civil legislation are sometimes used to railroad a bunch of harmful proposals through. This may happen with open protocols, too, either accidentally or on purpose. How do we prevent such misuse of the flexibility of open protocols?

As long as these problems are not solved, it is almost impossible to start with the

---

2I refer to the constitution instead of the law in general, because members of parliament try to change the law (e.g., the civil law) to govern things that happen outside the House of Parliament. Thus, revisions of the civil law do not have immediate repercussions to the law-making process in the parliament, and therefore do not contribute to a discussion on self-governing protocols.
implementation of open protocols for distributed computer systems. Current research has not much delivered. At this moment, there are not many protocols that allow modification by the computers that use them. There are no protocols that allow a substantial modification by their users (which, again, are the computers and not the designers), and there are no certainly no protocols that allow their users to carry through radical and irreversible changes in the order of interaction. Existing inter-agent languages and protocols, like KQML (Neches et al., 1992; Finin et al., 1994; and Labrou et al., 1994), AKL (Haridi et al., 1992; Carlson et al., 1994; and Janson et al., 1994), and COOL (Barbuceanu et al., 1994), do not pay much attention to the autonomous change of protocol performatives. Finin et al. claim that KQML pays attention to how agents structure conversations. This is referred to as agent policies. Example agent policies in KQML include honesty, gullibility, helpfulness, responsiveness, empathy, pertinence, and identity. Unfortunately, the policies as indicated by Finin et al. are not ways to structure conversations, but merely motives that lie behind structuring conversations. In KQML, it is left unspecified how to structure or re-structure a multi-agent conversation. COOL comes closer to the realization of an open protocol, since Barbuceanu et al. propose to extend KQML with conversation rules and conversation classes. The conversation rules can in theory be modified by the agents that use them, and might thus, in theory, function as a prototypical arsenal of service primitives for open protocols.

Contemporary research on intelligent network protocols and inter-agent languages does not materialize into results that can be applied to the construction of open protocols. The reason for this ‘failure’ is that even the elementary concepts for autonomous communication, such as self-modifying dialogue protocols, bootstrap communication and autonomous law-making, are not very well understood. That is unfortunate, because a great deal of understanding of these sub-problems is required to start developing communication software for autonomous agents.

One small area, however, shows remarkable progression. The activity revolves around a game, called Nomic.

1.1. What is Nomic?

Nomic [from the Greek νόμος (nómos), meaning ‘law’) is an abstract game of rule-making and legislation, that recently gains popularity among a selected group of dedicated enthusiasts with access to international computer networks.

Nomic is conceived and designed by Peter Suber (1980), who presented it as a self-modifying rule-making and legislation language. Nomic is a game in which the rules are created and modified by the players. The game is based on the idea of evolving a set of rules that govern the game, with the players having the ability to modify these rules as the game progresses. This creates a dynamic and interactive environment where the rules of the game are not fixed but can be changed and adapted by the players.

Throughout this paper, I use ‘distributed computer systems’ as a convenient shorthand for distributed knowledge-based systems, cooperating multi-agent systems (MAS), intelligent information retrieval systems, distributed control systems, and distributed problem solving systems (DPS).

At the beginning of this introduction it was argued that the protocols used by modern high-speed modems, and the PPP proposed by (Simpson et al., 1992) are two examples of protocols that, loosely speaking, allow autonomous modification on the run. Of course, there are more such protocols, but they all fall in the same category of ‘openness’.

KQML (Knowledge Query and Manipulation Language) is a language and protocol for exchanging information and knowledge. It is both a message format and a message-handling protocol to support run-time knowledge sharing among agents. AKL (Agent Kernel Language) is a concurrent constraint programming language, developed at the Swedish Institute of Computer Science (SICS). In AKL, computation is performed by agents interacting through stores of constraints. COOL (Coordination language) relies on speech-act based communication, but integrates it in a structured conversation framework that captures the coordination mechanisms agents use when working together.

Accordingly, much of the information about Nomic presented here is fetched from the internet and similar information services. A Nomic FAQ list can be FTP’d from ftp.cse.unsw.edu.au in the directory /pub/users/s2119737/nomic/FAQ. An HTML version also exists at: http://www.cse.unsw.edu.au/~s2119737/nomic-faq.html. News and other things worth knowing about Nomic are to be found regularly in the newsgroup rec.games.abstract. (As of May 10, 1995.)
game, based on reflexivity in law. The game was first published in Douglas Hofstadter’s column “Metamagical THEMES” (Hofstadter, 1982), and later in Hofstadter’s book, by the same name (Hofstadter, 1985). Suber revised the rules and published them in his own book (Suber, 1990).

Here are a few initial rules of Nomic:

Rule 201. Players shall alternate in clockwise order, taking one whole turn apiece. Turns may not be skipped or passed, and parts of turns may not be omitted. All players begin with zero points.

Rule 202. One turn consists of two parts, in this order:

1. Proposing one rule change and having it voted on;
2. Throwing one die once and adding the number of its points on its face to one’s score.

Rule 203. A rule change is adopted if and only if the vote is unanimous among the eligible voters.

The idea behind Nomic is to change the rules of Nomic. The game can be completely different at the end than it was at the start. The basic play is explained in rule 202: a player proposes a rule change, all the players vote on it, and if the vote succeeds, the change is immediately incorporated into the game. An interesting point is that rule 202 itself can be changed. If a player changes this rule successfully, then the way you play Nomic changes, and the game proceeds from there. Nomic is completely self-reflexive. Every rule of Nomic can be changed, including the rule that says you can change rules. In principle, Nomic can become any other game.

The importance of Nomic for the theory of amendable protocols is that it offers an excellent ‘playground’, on which new ideas become prosperous.

1.2. The problem which this paper deals with

It was already remarked that the idea of open communication protocols provokes a host of problems. In this paper, I have tried to answer the following basic question:

Just what happens if participants in a voting, vote about modifications of the rules of the voting procedure?

For example, what happens if we let participants vote about altering the number of votes required to pass a proposal? What happens if such a voting is repeated, say, a 10,000 times? These, and similar questions are considered in this paper.

I choose elementary voting games as object of my study for two reasons. The first reason is that elementary voting games resemble Nomic, in which participants also vote about protocol changes (at least initially). Nomic is generally considered to be the prototype of a self-modifying protocol game. The second, and more substantial reason, is that elementary voting games can easily be simulated on a computer. Simulating a vote is, basically, nothing more than tossing a coin where heads means ‘yea’ and tails means ‘nay’. More complicated self-modifying protocol games involve finding rational support for proposals, dispute mediation, and other ramifications of parliamentary procedure. These aspects are too difficult to deal with at the present stage of the research.

Finding an answer to the question I formulated above is important for the following reasons:

1. Construction. Since the experiments suggested must be performed on a computer, this
necessarily involves the construction of a computer program that performs a number of elementary self-modifying protocol games. Although the program will be far from ideal, it is a first step towards running self-modifying games on the computer.

2. **Observation.** The experiments below literally enable us to watch at the trail of several interesting, albeit impoverished, versions of self-modifying protocol games.

3. **Reporting our findings.** We write down our observations, and compare them with our expectations. The findings thus reported yield insight about the development and behaviour of protocol games.

These activities serve two goals. The first goal is to show what happens if a group of participants alter a shared protocol by using it. This is done in Sections 4, 5, and 6. The second goal is to indicate in which way research on this subject must be continued and extended, which is done in Sections 7, and 8.

**Contribution**

This paper contains the results of several experiments in elementary self-modifying protocol games, such as Nomic. To my knowledge, this article is the first report that shows how ‘impoverished’ versions of such games develop.

**Contents**

1. Introduction
2. The setup
3. Simulation on a computer
4. First experiments: voting about quotum
5. More experiments: voting about quotum + quorum
6. More experiments (continued): voting about quotum + preference
7. Future experiments
8. Future research (in general)
9. Conclusions
   - References
   I. Voting about quotum
   II. Voting about quotum + quorum
   III. Voting about quotum + preference
   IV. Program that was used to perform the experiments

Sections I, II and III contain the results of the experiments.

2. **The setup**

My aim is to define an impoverished version of Nomic, that can run on a computer. Since Nomic is a voting game, at least initially, this means that my aim is to define an elementary voting game in which players vote about rule changes. The next aim is to simulate this elementary voting game on a computer.
2.1. Definition of an elementary voting game

We assume a group of $N$ players. At each stage of the game, some of these $N$ players are entitled to cast their vote. This group is called the quorum. The quorum consists of at least 1, and of at most $N$ players. The players that do not belong to the quorum in a certain stage of the game, do not participate in a voting.

An elementary voting game is played in rounds. A round consists of three stages:

1. The submission of a proposal that pertains to a rule change.
2. Voting on that proposal.
3. If the vote succeeds, the proposed rule change is immediately incorporated in the game.

Submission of proposals

The proposal is submitted by a third party. This means that participants do not submit proposals, and do not exercise influence on the selection of proposals to be submitted.

Proposals come from a so-called proposal space. The proposal space is a collection of rule changes that may be proposed in a game. An example of a possible proposal space is the following.

$$P = \{\text{quorum} := k \mid 1 \leq k \leq N\}$$ (1)

where $\text{quorum} := k$ is a proposal to set the quorum to $k$ active voters in the next round. Thus, a game with proposal space (1) is bound to rule changes in which the quorum is re-set incessantly to ever different sizes. Of course, there are many other possible proposal spaces.

Voting on proposals

At each stage in the game, there is a non-negative number called the quotum. The quotum determines the number of votes required to pass a proposal, in the sense that the proposal passes if the number of yea’s is greater than or equal to $\text{quotum} \times \text{quorum}$. Thus, if the quorum consists of 4 participants and the quotum is 0.68, the proposal passes if the number of yea’s is greater than or equal to $0.68 \times 4 = 2.72$. Since the number of votes is always integer-valued, this means in this case that 3 participants must vote in favour of the proposal to let it pass.

Incorporation of accepted proposals

If a proposal is accepted, it is incorporated in the rules of the game at the end of the current round. For example, if the quotum is equal to 0.78, and 8 out of 10 participants vote in favour of the proposal to set the quotum to 0.98, the proposal in question passes, and the quotum is set to the new value of 0.98. (Note the order.)

2.2. Proposal spaces

Recall that a proposal space is a collection of rule changes that may be proposed in a game.

The nature of a voting game is determined by its proposal space. For example, if the proposal space is large and contains many different proposals for rule changes, the game may develop into many different, often unpredictable, directions. Nomic is a typical example of such a game. On the other hand, if the proposal space contains only a few rule changes, or if the proposal space contains many similar elements (cf. equation 1), the game is often predictable. Therefore, the selection of a suitable proposal space largely determines the practicability of the experiments that we want to perform.
<table>
<thead>
<tr>
<th>Rule</th>
<th>Possible amendment</th>
</tr>
</thead>
</table>
| 1. The game must be continued | • the game must be closed  
• the game must be closed at the end of round \( <n> \) |
| 2. Quorum is equal to \( <quorum> \) | • \( \text{quorum} := <new\ quorum> \)  
• \( \text{quorum} := \text{quorum} + \frac{1}{N} \)  
• \( \text{quorum} := \text{quorum} - \frac{1}{N} \) |
| 3. Quorum is equal to \( <quorum> \) participants | • \( \text{quorum} := <new\ quorum> \)  
• \( \text{quorum} := \text{quorum} + 1 \)  
• \( \text{quorum} := \text{quorum} - 1 \) |
| 4. Preference of an individual voter towards adoption of new proposals is equal to \( <preference> \) | • \( <preference> := <new\ preference> \)  
• \( <preference> := <preference> + 0.01 \)  
• \( <preference> := <preference> - 0.01 \) |
| 5. The next proposal to be submitted is drawn from the proposal space \( \text{ad\ random} \) | • the proposal space is linearly ordered, and submission is periodical, by order  
• the quorum votes about which proposal will be voted on next  
• the quorum votes about whether to vote about which proposal will be voted on next |
| 6. Voting is done synchronously, in rounds. | • voting is done asynchronously, by using a voting noticeboard |
| 7. Members of the quorum are obliged to mark their vote with a definite yes or no. | • members of the quorum are allowed to abstain  
• participants not in the quorum are allowed to cast their vote by authorization to a member of the quorum; a member may be authorized by at most two participants |
| 8. The proposal space remains fixed throughout the game. | • the proposal space is emptied  
• the proposal space may be extended by proposing an extension, and having it voted on  
• the proposal space is set equal to the set of all possible proposals |
| 9. All players must always abide by all the rules then in effect, in the form in which they are then in effect, and interpreted in accordance with currently existing game custom. | • let the hell break loose |

Table 1: Summary of possible rules together with possible amendments.

To see which rules are suitable for use, let us consider Table 1. This table presents a varied summary of possible rules with possible modifications of these rules. On the one hand, I have included rules that easily lend themselves for translation into computer code (rule 1, rule 2, rule 3, rule 4, rule 5, rule 6, and rule 7); on the other hand, I have included rules for which it is almost impossible to translate them into working computer code (rule 8, and rule 9). I have done so to show that one has to be careful with the selection of modifiable rules for simulating self-modifying games on the computer.
In what follows, I will tell which rules are selected for the experiments, and which are not. Every choice is briefly motivated.

1. A rule concerning the continuation of a game can easily be translated into computer code. However, this rule is not made modifiable in the experiments, because I want to keep control over the length of the simulations.

2. A modifiable rule defining the quotum can be represented by a register that stores a numerical value, i.e. the quotum. Since computers are good at such things, a modifiable rule defining the quotum will therefore be used in the forthcoming experiments. There are several possible modifications to this rule. I have chosen to use the following.
   - increase of quotum
   - decrease of quotum

   I have chosen not to incorporate discrete rule changes, because the resulting graphs are difficult to read, with the consequence that I am unable to produce satisfactory results.

3. The same holds for a modifiable rule defining the quorum. The quorum is a number and can easily be represented and manipulated by a computer. There are several possible modifications to this rule. I have chosen to use the following.
   - increase of quorum with one participant
   - decrease of quorum with one participant

   I have chosen not to incorporate discrete rule changes, for the same reason as above.

4. A modifiable rule defining the preference of an individual voter towards adoption of new proposals is incorporated in future experiments for similar reasons. I have chosen to incorporate only one-step rule changes for the same reasons as above.

   It must be admitted that a rule defining the preference of an individual voter seems artificial. This is true, if we assume that a vote is the result of an intelligent deliberation on occasion of a proposed rule change. But we have not assumed this. Instead, it was stipulated that a vote is obtained by performing a two-valued random experiment. For example, we might imagine that the participants maintain a certain political attitude towards changes in the protocol, ranging from extremely conservative (‘no new proposal gets ever passed!’) to extremely progressive (‘let it flow!’). Furthermore, we might imagine that every proposal is equally important to every participant, so that each participant has to distribute its political attitude over the sequence of proposed rule changes in an even way, without preference. Since the participant is indifferent about the relative importance of the proposals, we might imagine that it tosses a biased coin to determine its vote.

5. The decision which proposal is submitted next is made ad random by the computer. It is possible to write computer code that let participants vote about which proposal is submitted next. However, this is not a straightforward job, and distracts from our original intention to perform elementary experiments.

6. It is possible to represent rules that let computers alternate between synchronous and asynchronous voting. Here, a blackboard architecture (voting noticeboard architecture, if you wish) is appropriate to implement asynchronous voting. (Cf. Morgan et al., 1988; Craig, 1992; Velthuijsen, 1992.) However, this is a non-trivial task, that falls beyond the scope of our present objective.

7. For a rule concerning abstention holds the same as above: in principle, such a rule can be implemented on the computer, but I have chosen not to do so.

8. Rules defining or altering the proposal space are of another category. If one wishes to
make such rules modifiable, it is important to be aware of the fact that modifications of the proposal space do not come ‘out of the blue’ but must, on their turn, come from a ‘secondary proposal space’. This ‘secondary proposal space’ might be considered as a stock (or inventory) of possible modifications and extensions of the ‘primary proposal space’. Clearly, such ideas fall beyond the scope of this paper as well.

9. The rule to obey the rules is included in Table 1 merely for the sake of contrast. It shows that there exist rules for which it is hard to imagine that they can ever be modified in the course of self-modifying computer games. At least to me, a rule that says to obey the rules is unassailable, i.e. hardwired into the protocol. If this rule is substantiated in the form of working computer code, it will be code that says: ‘don’t follow the code’. This is not entirely absurd (cf. Hofstadter, 1985), but somewhere we will have to draw a line between ‘possibly modifiable’ and ‘definitely unassailable’.

Yet more complicated rules and proposals for rule changes include the enactment, or repeal, of rules; the enactment, repeal, or amendment of amendments, and the incorporation of rules that were formerly understood to be unchangeable. For now, we leave the more complicated rule changes for future research.

3. Simulation on a computer

Throughout this entire paper it is assumed that voting is random. We assume that a vote is generated by an elementary two-valued random process, such as tossing a coin. In this paper, it is assumed that a vote is not the outcome of an intelligent deliberation on occasion of a proposed rule change.7

3.1. Simulation of votes

A vote with possible outcomes \{yea, nay\} can be simulated by tossing a coin, where heads means ‘yea’ and tails means ‘nay’. More properly, a vote with possible outcomes \{yea, nay\} can be simulated by a two-valued random variable8

\[ \chi: \{\text{yea, nay}\} \rightarrow [0,1]. \]

The value \(\chi(\text{yea})\) expresses the preference of an individual participant towards the adoption of an arbitrary proposal. For instance, if \(\chi(\text{yea}) = 0.98\), the probability that an individual participant votes in favour of the proposal is 0.98, and the probability that an individual participant votes against the proposal is 0.02.

As a result, the number of yea’s that are issued by a quorum of size \(n\) is a binomial random variable.9 That is, the probability that exactly \(k\) of \(n\) participants vote in favour of a proposed rule change is given by the formula

\[ P(\#\text{yea}'s = k) = P(\text{Bin}[n, p] = k) = \binom{n}{k} p^k (1-p)^{n-k}. \]  

where \(p = \chi(\text{yea})\).

---

7 This more complicated approach is followed in (Vreeswijk, 1995a). The price to be paid is the difficulty (if not impossibility) of performing experiments.

8 In probability theory, a two-valued random variable is called a Bernoulli random variable.

9 A binomial random variable Bin\((n, p)\) with parameters \(n\) and \(p\) is obtained if we toss a coin \(n\) times, and count the number of heads that occur; \(p\) is the probability that heads occurs when the coin is tossed once. See literature on probability theory (e.g., Ross, 1994).
Figure 2 shows the situation for \(n = 25\) and \(p = 0.50\).

![Figure 2: binomial distribution of yea’s, \(p = 0.5\).](image)

Referring to Figure 2, we may for instance conclude that the probability that exactly 15 of 25 participants vote in favour of a proposal is approximately equal to 0.1.

With the knowledge that the number of votes is binomially distributed, we are able to infer the probability that a proposal passes, given the number of votes required to do so. The probability that a proposal passes if \(k\) votes are needed in the presence of \(n\) participants, is given by the formula

\[
P(\text{\# yea’s } \geq k) = P(Bin[n,p] \geq k) = 1 - P(Bin[n,p] < k) = 1 - P(Bin[n,p] \leq k-1) = 1 - \left[ \sum_{i=0}^{k-1} \begin{pmatrix} n \\ i \end{pmatrix} p^i (1-p)^{n-i} \right].
\]

Figure 3 shows the situation for \(n = 25\) and \(p = 0.50\):

![Figure 3: expected speed of the game, \(p = 0.50\).](image)

Figure 3 tells us, for instance, that, when the number of votes required to pass a proposal is equal to 15, the probability that a proposal passes is approximately equal to 0.25.

From the elementary exercises in probability above we may conclude some interesting things,
that help us to understand the forthcoming experiments.

1. **Speed of the experiments.** The quotum (or, equivalently, the number of votes required to pass a proposal) strongly determines the speed of the experiments. For example, if the quotum approaches 0.0, the probability that a proposal passes will tend to 1. If a session is defined to be, say, a block of 100 rounds, this means that the expected number of rule changes per session, *i.e.* the expected speed of the game, will tend to 100 rule changes per session. Conversely, if the quotum approaches 1.0, we may expect that the game gets slower, and slower, and slower. From Figure 3 we may expect that the experiments proceed slowly as soon as the number of votes required to pass a proposal is greater than or equal to 18.

2. **The change in the speed of the experiments.** The slope of the points in Figure 3 is non-linear. As the number of votes required tends to 25, the points soon nestle tightly against the x-axis. This means that the experiments will almost ‘get stuck’ when the number of votes required to pass a proposal is greater than or equal to 18.

To see whether these statements really do hold, we will have to perform the experiments.

4. **First experiments: voting about quotum**

In the first set of experiments, 15 participants vote about proposals to change the quotient. It is assumed that each participant votes either for or against a proposal, with equal likelyhood.

The following rule table is used:

<table>
<thead>
<tr>
<th>Rule</th>
<th>Possible amendment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Quotum is equal to &lt;quotum&gt;</td>
<td>• quotum:= quotum + 1/15</td>
</tr>
<tr>
<td></td>
<td>• quotum:= quotum − 1/15</td>
</tr>
</tbody>
</table>

Table 4: possible rules and amendments of 1st set of experiments.

Several experiments were performed, with a length varying from 10 to 10,000 rounds. It was decided to include only two experiments of 1,000 rounds each, because they summarize the situation most adequately.

The results are displayed in the appendix:

From the first set of experiments we learn that, if the quotient is near 15, the chance that a next proposal passes is low. This complies with our expectations that were expressed in Section 3. Further, we observe that voting about the number of votes required to pass a proposal seems to be a process that has the tendency to direct itself into high quota.

**Markov chain**

Intuitively, it will be clear that, eventually, the quotient will reach the value 1+1/15, and will stay there. This intuition turns out to be true, but a rigorous proof of it requires no less than an introduction to the theory of Markov chains (cf. Freedman, 1976; Manjunath, 1984). This surely is beyond the scope of the present paper.1,10
5. More experiments: voting about quotum + quorum

The first set of experiments were performed to offer a first acquainittance with the basic statistics of elementary voting games. A characteristic aspect of the first set of experiments is the number of rules that is subject to change, viz. one: only one rule can be modified during an experiment.

The logical next step is to increase the number of rules that can be modified, not in the last place because Nomic initially has 13 such rules. In this way, we obtain games that move along \( n \) different dimensions, \( 1 \leq n \). (Thus speaking, Nomic initially moves along 13 dimensions; maybe Robert’s rules of order initially moves along 130 different dimensions?)

To create a game that moves along 2 different dimensions, we make one more rule modifiable. Let us take the rule that defines the quorum. This gives us two modifiable rules: one rule defining the quotum, and one rule defining the quorum. Table 5 further shows which admissible modifications I have chosen for these rules.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Possible amendment</th>
</tr>
</thead>
</table>
| 1. Quotum is equal to \(<quotum>\) | \(\text{quotum}:= \text{quotum} + 1/25\)  
\(\text{quotum}:= \text{quotum} - 1/25\) |
| 2. Quorum is equal to \(<quorum>\) participants | \(\text{quorum}:= \text{quorum} + 1\)  
\(\text{quorum}:= \text{quorum} - 1\) |

Table 5: possible rules and amendments of 2nd set of experiments.

The proposal space is equal to
\[
P = \{v+, v-\} \cup \{q+, q-\}
\]
where
\[
v+ = \text{`quotum:= quotum + 1/25'},
v- = \text{`quotum:= quotum - 1/25'},
q+ = \text{`quorum:= quorum + 1'}, \text{and}
q- = \text{`quorum:= quorum - 1'}.
\]

It is assumed that new proposals are submitted by a third party, in this case the computer. Furthermore, it is assumed that the computer selects the proposals ad random from the proposal space.

The development of the experiments will be displayed as in Figure 6. Every game ‘walks’ through a x-y coordinate-system, in such a way that the points in the grid correspond to states of the game. For example, the point \((0.2, 5)\) corresponds to a state of the game in which the quotum is 0.2, and the quorum is equal to 5 participants.

Before running the experiments, it is helpful to calculate the expected speed, for each point in the grid. (Recall that the speed of a game is defined as the number of rule changes per 100 rounds.) The expected speed tells us at which points the experiments tend to run fast, and at which points the experiments tend to run slow.

\(^{10}\)Rules 201-213, the so-called mutable rules.
By definition, the expected speed of a game at point \((x, y)\) is directly related to the probability of a rule change at \((x, y)\). We are therefore going to compute the latter. From equation (3) we know that the probability of a rule change is equal to

\[
P(\#\text{yea}'s \geq k)
\]

where \(k\) is the number of votes required to pass a proposal. Since \(k = \lceil x*y \rceil^{11}\) at \((x, y)\), we have

\[
P(\#\text{yea}'s \geq \lceil x*y \rceil)
\]

which is equal to

\[
P(\text{Bin}[y, p] \geq \lceil x*y \rceil).
\]

Hence, the expected speed of the game at point \((x, y)\) is given by the formula

\[
E(x, y) = 100 \times P(\text{Bin}[y, p] \geq x*y)
= 100 \times P(\text{Bin}[y, p] \geq \lceil x*y \rceil)
= 100 \times [1 - P(\text{Bin}[y, p] < \lceil x*y \rceil)]
= 100 \times [1 - P(\text{Bin}[y, p] \leq \lceil x*y \rceil - 1)]
= 100 \times [1 - \sum_{i=0}^{\lceil x*y \rceil - 1} \binom{y}{i} p^i (1-p)^{y-i}].
\]

For each possible \((x, y)\) pair, i.e. each possible quorum-quorum pair, the expected speed for \(p = 0.5\) is calculated and displayed in Figure 7.

---

\(^{11}\)The number \(\lceil x \rceil\) is the ceiling of \(x\) and is equal to the smallest integer \(n\) such that \(x \leq n\). E.g., \(\lceil 4.8 \rceil = 5, \lceil 4.0 \rceil = 4, \lceil -4.8 \rceil = -4\), and so forth.
The bottom line of Figure 7, for instance, gives information about the course of an experiment when the quorum consists of precisely one participant:

- If $quotum = 0.0$, there are no votes required to pass a proposal, which means that every proposal gets passed. It follows that the expected speed of the game is equal to 100.
- If $0.0 < quotum \leq 1.0$, the acceptance of the proposal in question is in the hands of a single participant. There is a 0.50 chance that this participant votes in favour of the proposal, in which case ‘everybody’ votes in favour of the proposal. The expected speed of the game is equal to 50.
- If $1.0 < quotum$, the number of votes required to pass a proposal is greater than the maximal number of positive votes (i.e. 1), so that no proposal ever gets passed. Hence, the expected speed is equal to 0.0.

More generally, we see that a small number of active participants leads to fast developments in the game. Conversely, a large number of active participants, combined with a high threshold, tends to block many proposed rule changes. This prediction is confirmed by the experiments (cf. Set II of the appendix). In order to accelerate the experiments, I performed a number of
Simulations in which every participant almost always votes in favour of a proposed rule change. More specifically, I looked at what happened when \( \chi(yea) = 0.98 \). Referring to Figure 8, we may expect that, in the new scenario, when \( p = 0.98 \), there is at each point a fair chance that the next proposal gets passed. (Except in case \( 1.0 < \text{quotum} \).)

![Figure 8: expected speed of the game (in rule changes/100 rounds), \( p = 0.98 \).](image)

The last six runs (of 750 rounds + high preference towards adoption) are random walks that eventually get stuck at the right edge of the square. (At the end of Section 4 it was argued that this always happens.) Furthermore, there is neither a mutual influence between the two variables (such as mutual reinforcement), nor a strange-attracting or self-organizing behaviour.\(^{12}\)

\(^{12}\)The term ‘self-organizing system’ is first defined by Farley and Clark of Lincoln Laboratory in 1954 (cf. Jacobi et al., 1962): “a self-organizing system is a system that changes its basic structure as a function of its experience and environment.” This definition clearly relates to today’s hot topics of adaptive control, unsupervised learning, emergent behaviour, and genetic algorithms.

A strange attractor is the limit set of a chaotic trajectory. It is distinct from a periodic orbit or a limit cycle. For example, consider a rule set of a self-modifying game defined by all the initial conditions such a game may have. For a dissipative game, such as Nomic, the conceptual space occupied by this rule set will shrink as the game evolves in time (legal variant of Liouville’s Theorem). If the game is sensitive to initial conditions, the trajectories of the rules defining initial conditions will move apart in some directions, closer in others, but there will be a net shrinking of the rule set.
This is due to the simplicity of the experiments.

6. More experiments (continued): voting about quotum + preference

To create another game that moves along 2 different dimensions, we make another rule modifiable. This time, let us take the rule that defines the preference of an individual participants towards adopting a proposed rule change. As in with the previous set of experiments, this gives us two modifiable rules: one rule defining the quotum, and one rule defining the individual preference. Table 9 further shows which admissible modifications I have chosen for these rules.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Possible amendment</th>
</tr>
</thead>
</table>
| 1. Quotum is equal to \(<quotum>\) | * quotum := quotum + 1/25  
* quotum := quotum − 1/25 |
| 2. Preference of an individual voter towards adoption of new proposals is equal to \(<preference>\) | * \(<preference> := <preference> + 0.04\)  
* \(<preference> := <preference> − 0.04\) |

Table 9: possible rules and amendments of 3rd set of experiments.

The proposal space is equal to

\[ P = \{v+, v−\} \cup \{p+, p−\} \]

where

\[ v+ = \text{`quotum} := \text{quotum} + \frac{1}{25} \]  
\[ v− = \text{`quotum} := \text{quotum} − \frac{1}{25} \]  
\[ q+ = \chi(\text{yea}) := \chi(\text{yea}) + .04, \text{ and} \]  
\[ q− = \chi(\text{yea}) := \chi(\text{yea}) − .04 \].

Each round, an arbitrary element of \(P\) is chosen, and submitted to a fixed quorum of 12 participants.

Before running the experiments, it is again helpful to calculate the expected speed, for each point in the grid. From equation (3) we know that the probability of a rule change at point \((x, y)\) is

\[ P(\text{number of yea's} \geq \lceil x*12 \rceil) \]

which is equal to

\[ P(\text{Bin}[12, y] \geq \lceil x*12 \rceil). \]

It follows that, the expected speed of the game at point \((x, y)\) is given by the formula

\[ E(x, y) = 100 * P(\text{Bin}[12, y] \geq x*12) \]

Ultimately, all rules and regulations lie along a fine line of zero volume. This is the strange attractor. All initial rules which ultimately land on the attractor form a Basin of Attraction. A strange attractor results if a game is sensitive to initial conditions and is not conservative. [Nomic is dissipative, because the game may change beyond recognition.]11
\begin{equation}
= 100 \times P(Bin[12, y] \geq [x^*12])
= 100 \times [1 - P(Bin[12, y] < [x^*12])] 
= 100 \times [1 - P(Bin[12, y] \leq [x^*12] - 1)]
= 100 \times [1 - \sum_{i=0}^{[x^*12] - 1} \binom{12}{i} y^i(1-y)^{12-i}].
\end{equation}

For each possible \((x, y)\) pair, i.e. each possible quotum-quorum pair, the expected speed is calculated and displayed in Figure 10.\textsuperscript{13}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure10.png}
\caption{expected speed of the game (in rule changes/100 rounds).}
\end{figure}

The difference between Figure 7 and Figure 10 shows that the experiments described in Section II and in Section III are based on really different conditions.

\textsuperscript{13} From equation (5) it follows that \(E(0.0, 0.0) = 100\) and \(E(1.0, 1.0) = 000.\)
7. Future experiments

Future experiments involve more modifiable rules, and larger proposal spaces. For example, such experiments might operate on a collection of 20 modifiable rules, plus 3 possible modifications per rule (which makes a proposal space of 60 possible amendments).

The development of such experiments may look like what is depicted in Figure 11. This figure shows a hypothetical collection of 7 graphs of 7 modifiable rules, indicating the value of each of these rules during a period of 250 rounds. (For instance, the graph of rule no. 1 expresses the size of the quorum; the graph of rule no. 2 expresses the duration (in minutes) of each voting period; the graph of rule no. 3 expresses the maximum number of times a rule may be modified; and so forth.) Such a multi-graph is one way to monitor the game, and there are certainly better ways to do this. (Using a 'scatterplot vector', for instance.) Finding a fortunate representation of the results of the experiments is also a problem that must be solved.

An obligatory enhancement of future experiments is the incorporation of rules of which the modification involves more than the adjustment of a corresponding numerical parameter. (Cf. the previous experiments.) A modification of the quorum rule, for instance, is nothing more than the adjustment of the parameter defining the size of the quorum. And likewise with rules defining the quorum and the preference of participants towards adoption. For these type of rules, the state of the protocol can be represented by a 7-vector, of which the 1st entry stores the value of, e.g., the quorum, the 2nd entry stores the value of, e.g., the quorum, and so forth.

In more complicated protocol games such as Nomic, the state of the protocol cannot be represented by means of a numerical $n$-vector. An adequate representation of the state of the protocol of such complicated games is therefore an essential prerequisite for further experiments in this direction. A logical framework in which such things can be done is proposed in (Vreeswijk, 1995a). In that paper, I propose to represent the state of an open protocol as a collection of logical propositions. This set of logical propositions is called the *standing order* which thus governs a dispute (instead of a voting). The dispute, on its turn, may produce arguments that change the standing order; cf. Fig. 12. Experiments with such logical protocol games will certainly produce useful results, particularly as regards to the theory of intelligent

---

14 A scatterplot for $n$ variables is a sequence of $n-1$ graphs that shows how one variable varies as a function of the other $n-1$ variables.
(i.e. non-random) protocol modifications.

8. Future research (in general)

Above, I indicated which experiments could be performed next if we were to continue the trend of performing several experiments in self-modifying protocol games.

Performing numerical experiments, however, is not a trend that I wish to continue in further research. The reason (that, I hope, will now be clear) is that the execution of a self-modifying protocol game is evidently more than the manipulation of a numerical $n$-vector. It is true that numerical experiments are a good first acquaintance with the peculiarities of self-modifying protocol, but the real problems with self-modifying protocol games lie further ahead.

From now on, I will further occupy myself with the symbolic aspects of self-modifying protocol games. Particularly the development of a working prototype of a multi-agent meeting system will be my next objective. The system I have in mind is a computer program, in which proposals are program statements. These program statements eq. proposals are incorporated in the protocol if they succeed to survive rational dispute. Although symbolic data is harder to present than numerical data, I do hope that this new project yields sessions of which the scripts can be presented as clearly as the results of the numerical experiments currently performed.

![Diagram of order, dispute, protocol]

Figure 12: self-modification in rational protocol games

9. Conclusion

The experiments of which the results are reported in this paper were performed with two goals:

1. To show what happens if a group of participants alter a shared protocol by using it.
2. To indicate in which way research on this subject must be continued and extended.

The first goal has been achieved insofar as elementary protocol games are concerned. We were able to look at the development of several elementary voting games, in which participants exercise influence on certain aspects of the protocol. The main conclusion as regards to these experiments is the following:
Elementary voting games exhibit a certain behaviour (namely, acceleration near low quota, changing performance due to changing combinations of quorum and quorum, and so forth). This behaviour is typical, but not complex. More specifically, the behaviour of elementary voting games is simple enough to be explained with the help of existing theories (in our case probability theory and the theory of Markov chains). Elementary experiments are useful to get used to the idea of autonomous protocol modification, but show too little that could not predicted beforehand with the help of existing scientific theories. More complex experiments are more likely to surprise us, and are therefore more likely to provide additional insight in the workings of self-modifying protocol games. Therefore, more complex experiments must be fabricated.

Thus, a possible point of criticism might be that experiments were unnecessary, since the results were predictable anyway. This is true, but misses the point. The point of this paper is to show, as clearly as possible, what happens if elementary self-modifying games are performed by a group. I have shown this,—and although I admit that the results are not extremely surprising for those that are in control of probability theory, I am convinced that my simple findings are likely to encourage other researchers to improve and extend the results presented in this paper in many ways.

The following issues were left unexplored, and deserve further attention:

a. Investigating what happens if the proposal space is extended, if more rules are made modifiable, and if more fundamental rules are made modifiable (such as the last rule of Table 1.)

b. Investigating which combination of rules and modifications is likely to exhibit a strange-attracting or self-organizing behaviour (Jacobi et al., 1962). (Sub-problems: what kind of strange-attracting or self-organizing behaviour do we wish to invoke? Can the behaviour be theoretically explained? And so forth.)

c. Suggested by P. Suber, 1995: investigating how to find a way to eliminate all rule-changing power permanently. This research activity, by the way, resides in the informal area (as opposed to the ‘mathematical area’).

d. Investigating how to run symbolic (instead of numerical) self-modifying protocol games. Which language are we going to use? Which representation? How do we separate layers of ‘computational legislation’?

Investigating how ideas on autonomous protocol modification can be worked into existing inter-agent communication languages, such as KQML, AKL, and COOL. This will not be easy! Despite work of Finin et al. (KQML), Carlson et al. (AKL), and Barbuceanu et al. (COOL), little to no research has been done on the modification of performatives of communication languages ‘on the run’ by agents that use them. It is now time to find out how such ideas can be incorporated in, say, KQML.

e. Investigating how Nomic can be formalized. Furthermore, investigating how computer Nomic tournaments can be programmed an organized, in analogy with core-war tournaments (Dewdney, 1990), and computer-chess tournaments.

f. Investigating the theoretical aspects of the area. According to scientific methodology, the appropriate order is:

(1) performing experiments

(2) trying to arrest typical behaviour

(3) explaining and predicting this behaviour with (new) hypothesis and theories
(4) verifying hypothesis and theories by (1) performing experiments

At this moment, I believe that the performance of more complex experiments might yield the typical behaviour we hope for, which might then be explained with the help of theoretical means.

Acknowledgements. I would like to thank Bart Verheij for pointing out an irregularity in a former presentation of the experiments. Furthermore, I would like to thank Peter Suber for his helpful comment on my work on Nomic.
References


Set I. Voting about quotum

The first set of experiments involves a sequence of votings about changing exactly one rule, namely, the rule that defines the number of votes required to pass a proposal, or, equivalently, the rule that defines the quotum.

Every experiment starts with a quotum of 0.33. The quotum varies throughout the experiment, in such a way that the number of votes required to pass a proposal remains between 0 and 16.

The number of active voters, the *quorum*, remains constant throughout the experiments, and is equal to 15.

The proposal space is equal to

\[ P = \{v^+, v^-\} \]

where

\[ v^+ = 'quotum := quotum + 1/15', \text{ and} \]
\[ v^- = 'quotum := quotum - 1/15'. \]

(Note that the proposal space is *continuous*, which means that it consists of elements that propose gradual transitions.) Each round, an arbitrary element of \( P \) is chosen, and submitted to the panel of active voters.

The number of active voters, the *quorum*, remains constant throughout the experiments, and is equal to 15. The voting behaviour of each individual active voter is determined by the random variable \( \chi: \{\text{yea, nay}\} \rightarrow [0,1] \), with \( \chi(\text{yea}) = 0.5 \).

For short:

| Nr. of rounds: | 1,000 |
| Nr. of participants: | 15 |
| Quotum: | 0.33 (initially) |
| Size of the quorum: | 15 |
| \( \chi(\text{yea}) \): | 0.5 |

In the graph of run no. 1, we see the development of a simulation in which 15 participants collectively vote about either increasing or decreasing the number of votes required to pass a proposal.

We observe that, if the quotum is near 15, the chance that a next proposal passes is remarkably
low. This coincides with our expectations that I expressed in Section 3. We see that the game is in easy fairway near rounds number 200, 450, and 900.

Many simulations proceed as follows:

![Graph showing the number of votes required to pass a proposal over the number of rounds. The graph shows a trend of increased votes with time, settling into a calm fairway.]

Apparently, voting about the number of votes required to pass a proposal is a process that has the tendency to direct itself into calm fairways.
Set II. Voting about quotum + quorum

This set is divided into 4 different subsets:

1. Six experiments of 125 rounds.
2. Six experiments of 750 rounds.
3. Six experiments of 10,000 rounds.
4. Six experiments of 750 rounds + high preference towards adoption of new proposals.

In the last subset, the game is accelerated by the assumption that each participant almost always votes in favour of a proposed rule change.
II.1. Six experiments of 125 rounds

I have performed 6 experiments, each lasting 125 rounds. Every experiment starts with 1 active voter. The number of active voters, the *quorum*, varies throughout the experiment, but remains always between 1 and 25.

For every experiment, the number of votes required to pass a proposal is determined by the *quotum*, and is equal to *quotum * quorum*. Every experiment starts with a quotum of 0.0. The quotum varies throughout the experiment, in such a way that the number of votes required to pass a proposal remains between 0 and 26.

The proposal space is equal to

\[ P = \{v^+, v^-\} \cup \{q^+, q^-\} \]

where

\[ v^+ = 'quotum:= quotum + 1/25', \]
\[ v^- = 'quotum:= quotum - 1/25', \]
\[ q^+ = 'quorum:= quorum + 1', \text{ and} \]
\[ q^- = 'quorum:= quorum - 1'. \]

(Note that the proposal space is continuous, which means that it consists of elements that propose gradual transitions.) Each round, an arbitrary element of \( P \) is chosen, and submitted to the panel of active voters.

The voting behaviour of each individual active voter is determined by the random variable \( \chi: \{\text{yea}, \text{nay}\} \to [0, 1] \), with \( \chi(\text{yea}) = 0.5 \).

For short:

| Nr. of rounds: | 125 |
| Nr. of participants: | 25 |
| Quotum: | 0.0 (initially) |
| Size of the quorum: | 1 (initially) |
| \( \chi(\text{yea}): \) | 0.5 |

Here are the first two experiments:
Since we are at this page anyway, let us consider run nr. 3. The trail of run nr. 3 starts at the bottom left corner of the rectangle, the point where there is precisely 1 qualified voter, and a quotum of 0.0. This means that, in the first round, every proposal submitted passes unconditionally, on behalf of one participant, whatever that proposal might be. Since the trail starts moving to the right, we conclude that the first proposal is of the form $v^+$. The proposal passes unconditionally, and the new quotum is set to $1/25$.

In the second round, at least two things might have happened. This can be seen by following the trail. Either the trail has moved back to the left, or it has moved further to the right. In the first case, the proposal was $v^-$, the single participant gave a 'yea', and the proposal was accepted. If the trail moved further to the right, the proposal was $v^-$, the single participant gave a 'yea', and the proposal was accepted also. There is also a slight chance that either one of the proposals for quorum change is submitted but rejected. In that case, the trail 'pauses' during round two. Unfortunately, the trail does not show us what really has happened.

The other four runs proceed likewise:
The bullets indicate the relative frequency by which a node is visited. If the bullet is large, this means that the node, and hence the corresponding combination of quorum and quorum, is visited many times in comparison with other nodes. If there is no bullet, or if the bullet is small, the node is visited only a few times in comparison with other nodes. Since the proposal space is continuous, fat nodes are often grouped together. The distribution of the nodes gives a good impression which spots of the proposal space the group tends to be in.

Note: the plots do not show where the trail ends. This does not matter, since the outcome of an experiment, i.e. the last node of a trail, is determined by the number of rounds, rather than by the type of experiment. The outcome is of an experiment is therefore not representative for the type of experiment.

**Findings that are typical for the current type of experiments:**

Most runs (viz. 1, and 3-6) do not pass the line $x = 0.5$. This can be explained with the help of Figure 7: for each experiment the trail is slowed down as soon as it moves to the right. The length of the experiments (125 rounds) is too short to let the game walk to the right. Only in run 2, the trail coincidently ‘finds’ the fast bottom line and creeps underneath, under the slow region, to reach the right half of the square.
II.2. Six experiments of 750 rounds

Next, I performed 6 experiments, of which each experiment lasts 750 rounds. Thus, the first box contains precisely all six experiments of the first set of experiments of 125 rounds. You can check this by putting the first box of the first set of experiments over the first box of the second set of experiments, and holding the pages against the light.

For short:

- Nr. of rounds: 750
- Nr. of participants: 25
- Quotum: 0.0 (initially)
- Size of the quorum: 1 (initially)
- $\chi(yea):$ 0.5

The rest of the parameters are the same as as in the previous set of experiments.
Findings that are typical for the current type of experiments:

When the number of rounds is increased, obviously the first thing that we see is that the trail gets longer. But there is more.

Some of the experiments result in a graph that shows that, if the number of rounds is increased (in this case to 750), some runs make it to the right half of the square (viz. run 2, 5, and perhaps 4). Moreover, we see that, once the trail (i.e. the game, or the state of the game) has reached that area, it tends to stay there. Run 6 seems to try to ‘climb over’ the bell-shaped curve of Figure 7 ...
II.3. Six experiments of 10,000 rounds

I performed 6 experiments, of which each experiment lasts 10,000 rounds. Thus, the first box contains all six experiments (and more) of the previous set of experiments of 750 rounds.

For short:

<table>
<thead>
<tr>
<th>Nr. of rounds:</th>
<th>10,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nr. of participants:</td>
<td>25</td>
</tr>
<tr>
<td>Quotum:</td>
<td>0.0 (initially)</td>
</tr>
<tr>
<td>Size of the quorum:</td>
<td>1 (initially)</td>
</tr>
<tr>
<td>( \chi(\text{yea}) ):</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Here are the results:
Findings that are typical or the current type of experiments:

In all experiments (no exception), the game has reached the right half of the square. The game tend to reside there most of the time, which was already observed on occasion on the previous experiments.

Furthermore, the graphs of this set of experiments show an interesting thing, that could not be observed on previous occasions.

Consider the following cut-points:

- run 1: point (0.84, 7)
- run 2: point (0.44, 10)
- run 3: point (0.40, 9)
- run 4: point (0.80, 9)
- run 5: point (0.72, 10)
- run 6: point (0.8, 9).

These points divide the squares in two parts: a left part and a right part. Since every trail of a run is connected (i.e. does not make leaps), we conclude that every trail must have passed its cut-point at least once to reach the right half of the square. Moreover, it is extremely unlikely (but not impossible) that each trail has passed its cut-point more than once (i.e. 3, 5, 7, 9, ...). In turn, this makes it extremely likely that the game tends to reside permanently on the right half of the square. There are two other pieces of evidence that point into this direction:

1. The relative frequency by which the points in the grid are visited (symbolized by the size of the dots).
2. The number of edges that were visited by the game, in each experiment (viz. 267, 122, 136, 289, 242, 161, respectively). This number is, on the average, equal to 200, which means that each edge has been walked over $10,000/200 = 50$ times, on the average.

In total, this evidence suggests that every game ‘walks’ to the right, and tends to stay there.

Run 5 is the first case of a game that lapses into a dead state [viz. (1.04, 6)]. If the game were allowed to continue indefinitely, it would run forever, and no proposal would ever get passed.
II.4. Six experiments of 750 rounds + high preference towards adoption

To see if the average individual preference of the participating voters has influence on the course of a run, I set up six experiments of 750 rounds, such that the voting behaviour of each individual active voter is determined by the random variable

\( \chi: \{\text{yea}, \text{nay}\} \rightarrow [0,1], \text{ with } \chi(\text{yea}) = 0.98. \)

For short:

- Nr. of rounds: 750
- Nr. of participants: 25
- Quotum: 0.0 (initially)
- Size of the quorum: 1 (initially)
- \( \chi(\text{yea}): \) 0.98

Here are the results:
Findings that are typical for the current type of experiments:

There are two types of runs: runs going to the right that lapse into a dead state (2, 3, 5) and runs that do not (1, 4, 6). For both type of runs, we see that the size of the quorum is of no relevant influence—as opposed to the experiments in which $\chi(yea) = 0.50$. The runs that remain wandering on the left, show a widespread dispersal among the points visited.

(No further findings.)
Set III. Voting about quotum + preference

This set contains eight experiments of 750 rounds.
III.1. Eight experiments of 750 rounds

I have performed 8 experiments, each lasting 750 rounds. The number of active voters, the *quorum*, remains fixed throughout the experiment, and is equal to 12.

For every experiment, the number of votes required to pass a proposal is determined by the *quotum*, and is equal to *quotum* * quorum. Every experiment starts with a quotum of 0. The quotum varies throughout the experiment, but varies in such a way that the number of votes required to pass a proposal remains between 0 and 26.

The voting behaviour of each individual active voter is determined by the random variable $\chi: \{\text{yea}, \text{nay}\} \to [0,1]$. The value of $\chi(\text{yea})$ varies throughout the experiment, but is initially equal to 1.0.

The proposal space is equal to

$$ P = \{v+, v-\} \cup \{p+, p-\} $$

where

$$ v_+ = '\text{quotum}:= \text{quotum} + 1/25', $$

$$ v_- = '\text{quotum}:= \text{quotum} - 1/25', $$

$$ q_+ = '\chi(\text{yea}):= \chi(\text{yea}) + .04', \text{ and} $$

$$ q_- = '\chi(\text{yea}):= \chi(\text{yea}) - .04'. $$

Each round, an arbitrary element of $P$ is chosen, and submitted to the panel of active voters.

For short:

| Nr. of rounds: | 750 |
| Nr. of participants: | 25 |
| Quotum: | 0.0 (initially) |
| Size of the quorum: | 12 |
| $\chi(\text{yea})$: | 1.0 (initially) |

Here are the results:

![Graph](image-url)
preference of individual voter towards adoption

quotum

run 3

quotum

run 4

quotum

run 5

quotum

run 6

quotum

run 7

quotum

run 8
Findings that are typical for the current type of experiments:

Figure 10 (Section 6) shows a diagonal barrier (from the lower-left to the upper-right). This barrier has a restraining influence on the development of the experiments.

Let us tell for each run briefly what happens:

run 1: wanders in the fast area
run 2: manages to ‘walk around’ the frontier of the slow area
run 3: bogs down in the slow area
run 4: as run 2, manages to ‘walk around’ the slow area
run 5: as run 3
run 6: as run 3
run 7: as run 3
run 8: as run 1, not intercepted by slow area

Conclusion: it is a new experience to watch at these runs, but most if not all of them were predictable just by looking at Figure 10. See Section 9 of this paper for what this means for the general conclusion.
Appendix IV. Program

The program below has been used to perform the various experiments. It is available via anonymous ftp from

ftp.cs.rulimburg.nl:/pub/software/vreeswyk/vote.1.tar.Z

In Unix, you will have to make the following dialogue:

```
$ ftp ftp.cs.rulimburg.nl
Connected to bommel.cs.rulimburg.nl.
220 bommel.cs.rulimburg.nl FTP server (SunOS 4.1) ready.
Name (ftp.cs.rulimburg.nl): anonymous
331 Guest login ok, send ident as password.
Password: <your e-mail address>, e.g., zippy@cs.zoppy.nl
230 Guest login ok, access restrictions apply.
ftp> cd /pub/software/vreeswyk
250 CWD command successful.
ftp> ls
200 PORT command successful.
150 ASCII data connection for /bin/ls (137.120.13.8,3467)
vote.1.tar.Z
226 ASCII Transfer complete.
28 bytes received in 0.85 seconds (0.032 Kbytes/s)
ftp> binary
200 Type set to I.
ftp> get vote.1.tar.Z
200 PORT command successful.
150 Binary data connection for vote.1.tar.Z (137.120.13.8,3468)
226 Binary Transfer complete.
local: vote.1.tar.Z remote: vote.1.tar.Z
3983 bytes received in 0.0036 seconds (1.1e+03 Kbytes/s)
ftp> bye
221 Goodbye.
$ 
```

This should leave you with the file `vote.1.tar.Z` in your directory. Move this file to where you want to have `vote.lsp` and `uncompress` and `untar` `vote.1.tar.Z` in this directory:

```
$ uncompress vote.1.tar.Z
$ tar xf vote.1.tar
$
```

Go into the directory named `vote` and read the file `README`.

Below, the complete program is listed.

```
(defun dorun (&optional (n 1))
  (enter-seed)
  (princ " experiments start with ")
  (princ *random-state*) (terpri) (terpri)
  (dotimes (i n)
    (run 0 ; nr. of rounds
      0.0 ; initially required
      15 ; nr. of participants
    ))
```
(defun run
  (max-rounds
   initial-required
   nr-of-participants
   modify-type
   initial-quorum
   initial-prob-for)

  (when *grap-enabled*
    (princ "LP") (terpri))

  (when *header-enabled*
    (princ "rounds ") (princ max-rounds)
    (princ ",", initial-required)
    (princ ",", participants "") (princ nr-of-participants)
    (case modify-type
      (required (princ ", shift of quo") (terpri))
      (quorum (princ ", quorum shift") (terpri))
      (req-quo (princ ", quo + quorum shift") (terpri))
      (req-pro (princ ", quo + prob shift") (terpri)))
    (princ "quorum ") (princ initial-quorum)
    (princ ", probability pro ") (princ initial-prob-for)
    (terpri))

  (case modify-type
    ((req-quo req-pro)
      (let ((n (+ nr-of-participants 5)))
        (setq visit-array (make-array (list n n)))
        (dotimes (i n)
          (dotimes (j n)
            (setf (mref visit-array i j) 0))))))

  (when *grap-enabled*
    (princ ".sp 1") (terpri)
    (princ ".ps -1") (terpri)
    (princ ".G1") (terpri)
    (case modify-type
      (req-quo (princ "frame ht 1.9 wid 1.9") (terpri)
        (princ "coord x -1," (princ nr-of-participants)
        (princ "y 0," (print nr-of-participants))
      (req-pro (princ "frame ht 1.9 wid 1.9") (terpri)
        (princ "coord x -1," (princ nr-of-participants)
        (princ "y -1," (print nr-of-participants))
      (t (princ "frame ht 1 wid 3.8") (terpri)))
      (princ "draw solid") (terpri))

  (prog* ((required initial-required)
    (d-req (/ 1.0 nr-of-participants))
    (quorum initial-quorum)
    (d-quo 1)
    (prob-for initial-prob-for)
    (d-prob (/ 1.0 nr-of-participants))
    (max-nr-visit 0))

    (dotimes (round max-rounds)
(case modify-type
  (required
   (princ round) (princ " ")
   (print (round (* nr-of-participants required))))
  (quorum
   (princ round) (princ " ") (print quorum))
  (req-quo
   (let*
    ((array-point (round (* nr-of-participants required)))
     (visited+1 (1+ (mref visit-array array-point quorum)))
     (princ array-point) (princ " ") (print quorum)
     (setf (mref visit-array array-point quorum) visited+1)
     (when (< max-nr-visit visited+1)
       (setq max-nr-visit visited+1)))
  (req-pro
   (let*
    ((array-point (round (* nr-of-participants required)))
     (round-prob (round (* nr-of-participants prob-for)))
     (visited+1 (1+ (mref visit-array array-point round-prob)))
     (princ array-point) (princ " ") (print round-prob)
     (setf (mref visit-array array-point round-prob) visited+1)
     (when (< max-nr-visit visited+1)
       (setq max-nr-visit visited+1))))
    (terpri) (princ "ronde ") (princ (1+ round))
    (princ ", kiesdrempel ") (princ required)
    (princ ", quorum ") (princ quorum)
    (princ "") (princ (ceiling (* quorum required))) (princ ")")
    (when
     (proposal-adopted
      quorum
      required
      prob-for)
    (princ " adopted")
    ;; dit is tegelijk het voorstel en de adoptie ervan
    (case (parameter-to-be-changed modify-type)
      (required (cond
        ((<= required d-req)
         (setq required (+ required d-req)))
        ((evenp (random 10000))
         (setq required (+ required d-req)))
        (t
         (setq required (- required d-req))))
      (quorum (cond
        ((<= quorum 1)
         (setq quorum (1+ quorum)))
        ((<= nr-of-participants quorum)
         (setq quorum (1- quorum)))
        ((evenp (random 10000))
         (setq quorum (1+ quorum)))
        (t
         (setq quorum (1- quorum))))
      (probability (cond
        ((<= prob-for d-prob)
(setq prob-for (+ prob-for d-prob)))
((<= (- 1.0 d-prob) prob-for)
 (setq prob-for (- prob-for d-prob)))
((evenp (random 10000))
 (setq prob-for (+ prob-for d-prob)))
(t
 (setq prob-for (- prob-for d-prob)))))

(case modify-type
  (required
    (princ round) (princ " ")
    (print (round (* nr-of-participants required)))
  )
  (quorum
    (princ round) (princ " ") (print quorum)
  )
  (req-quo
    (let*
      ((array-point (round (* nr-of-participants required))))
      (visited+1 (1+ (mref visit-array array-point quorum))))
      (princ array-point) (princ " ") (print quorum)
      (setf (mref visit-array array-point quorum) visited+1)
      (when (< max-nr-visit visited+1)
        (setq max-nr-visit visited+1)))
  )
  (req-pro
    (let*
      ((array-point (round (* nr-of-participants required))))
      (round-prob (round (* nr-of-participants prob-for)))
      (visited+1 (1+ (mref visit-array array-point round-prob))))
      (princ array-point) (princ " ") (print round-prob)
      (setf (mref visit-array array-point round-prob) visited+1)
      (when (< max-nr-visit visited+1)
        (setq max-nr-visit visited+1))))

(when *grap-enabled* (case modify-type ((req-quo req-pro)
  (dotimes (req (+ nr-of-participants 4))
    (dotimes (quo (+ nr-of-participants 3))
      (let*
        ((nr-visit (mref visit-array req (1+ quo)))
        (size (floor (* 10.95 (/ nr-visit max-nr-visit)))))
        (when (not (zerop size))
          (princ "bullet size +") (princ (1- size))
          (princ " at ")
          (princ req (princ ",")
          (print (- (1+ quo) (expt (/ size 16.4421) 2))))))))

(when *grap-enabled*
  (princ ".G2") (terpri)
  (princ ".ps +1") (terpri)
  (princ ".sp 1") (terpri))

(if *grap-enabled*
  (return-from run ".LP)
  (return-from run ".t"))

(defun parameter-to-be-changed (modify-type)
  (case modify-type
    (required 'required)
    (quorum 'quorum)
    (req-quo (if (evenp (random 10000)) 'required 'quorum)
              (req-pro (if (evenp (random 10000)) 'required 'probability)))

(defun proposal-adopted (nr-of-voters required prob-for)
(let
  ((votes-for 0)
   (absolutely-required (ceiling (* (- required 0.001) nr-of-voters)))
   ;; aaargh! dit is echt ERG
   ;; (ceiling (* 1 1)) --> 2????
   ;; (cut-random (round (* 10000 prob-for)))
   (dotimes (nr-minus-one nr-of-voters)
     (when (<= (random 10000) cut-random)
       (setq votes-for (1+ votes-for)))
     (cond
      ((<= absolutely-required votes-for)
       (return-from proposal-adopted 't))
      ((<
         (- nr-of-voters 1 nr-minus-one)
         (- absolutely-required votes-for))
       (return-from proposal-adopted 'nil)))
   (return-from proposal-adopted 'nil))

(defun simulated-binomial-stochast (N p)
  (let
    ((mu (* N 0.5))
     (rho (sqrt (* N 0.5 (- 1.0 0.5)))))
    (do
      ((s (round (gaussian-stochast mu rho))
         (round (gaussian-stochast mu rho))))
       (and (<= 0 s) (<= s N)) s))))

(defun gaussian-stochast (&optional (mu 0) (rho 1))
  (if (null *iset*)
    (prog (v1 v2 fac)
      A (setq v1 (- (* 2.0 (ran1)) 1.0))
      (setq v2 (- (* 2.0 (ran1)) 1.0))
      (setq r (+ (* v1 v1) (* v2 v2))
        (when
          (<= 1.0 r)
          (go A)))
      (setq fac (sqrt (/ (* -2.0 (log r)) r))
        (setq *gset* (* v2 fac))
        (setq *iset* 't)
        (return (+ mu (* rho (* v2 fac)))))
    (prog ()
      (setq *iset* 'nil)
      (return (+ mu (* rho *gset*))))))
  (setq *iset* 'nil *gset* 0.0)

(defun ran1 ()
  (/ (random 615867342) 615867342))

(defun sqr (x) (* x x))

(defun enable-header ()
  (setq *header-enabled* 't))
(defun disable-header ()
   (setq *header-enabled* 'nil))

(enable-header)

(defun enable-grap ()
   (dribble "overdoen15")
   (princ ".nr PS 12") (terpri)
   (princ ".nr VS 14p") (terpri)
   (princ ".LP") (terpri)
   (setq *grap-enabled* 't))

(defun disable-grap ()
   (setq *grap-enabled* 'nil))

(defun grap () (enable-grap))

(disable-grap)

(defun enter-seed ()
   (princ "Please enter a seed for the random number generator -- ")
   (setq *random-state* (make-random-state :data (read))))

(load "/array.lsp")

(setq *currentload* "vote")

(setq *random-state* (make-random-state :data 2)) (grap)

(run 10000 ; nr. of rounds
  0.0 ; initially required
  15 ; nr. of participants
  'req-quo ; modification type
  1 ; initial quorum
  0.50 ; initial probability
)

) (exit)

(defun ps-r (participants &optional (m 1))
   (dribble "d")
   (dotimes (quorum* participants)
      (when (zerop (mod quorum* m))
         (let ((quorum (1+ quorum*)))
            (dotimes (quotum (+ participants 2))
               (when (zerop (mod quotum m))
                  (princ quotum) (princ " ")
                  (princ quorum) (princ " ")
                  (format t "Ä3,,,'0@aÄ%" (round (* 100
                                   (- 1
                                   (cum-binomial
                                    quorum
                                    (1- (ceiling (* (/ quotum participants) quorum))))
                                    0.98))))))))))

(defun ps-o (participants &optional (m 1))
   (dribble "d")
   (dotimes (prob (+ participants 1))
      (when (zerop (mod prob m))
         (dotimes (quotum (+ participants 2))
            (when (zerop (mod quotum m))
               (princ quotum) (princ " ")
               (princ quorum) (princ " ")
               (format t "'3,0@a"%" (round (* 100
                                   (- 1
                                   (cum-binomial
                                    quorum
                                    (1- (ceiling (* (/ quotum participants) quorum))))
                                    0.98)))))))))))


(princ prob) (princ "")
(format t ""3,,0@a"%"
(round (* 100
(- 1
(cum-binomial
12
(1- (ceiling (* (/ quotum participants) 12)))
(/ prob participants)))))
)

(defun gek (participants &optional (m 1))
  (dotimes (quorum* participants)
    (when (zerop (mod quorum* m))
      (let ((quorum (- participants quorum*))
        (dotimes (quotum (+ participants 2))
          (when (zerop (mod quotum m))
            (princ " ")
            (format t ""3,,0@a"
              (round (* 100
                (- 1
                (cum-binomial
                  quorum
                  (1- (ceiling (* (/ quotum participants) quorum)))
                  0.5)))))
              )))
        )
      ))
  ))
)

(defun cum-binomial (n k &optional (p 0.5))
  (do
    ((i 0 (1+ i))
     (s 0 (+ s (binomial n i p))))
    ((< k i) s)))
)

(defun print-binom (n &optional (p 0.5))
  (dotimes (i (1+ n))
    (princ i) (princ " ")
    (print (binomial n i p))))
)

(defun print-cum-binom (n &optional (p 0.5))
  (do*
    ((i 0 (1+ i))
     (s (binomial n i p) (+ s (binomial n i p))))
    ((> i n) 't)
    (princ i) (princ " ") (print s)))
)

(defun print-omgekeerd-cum-binom (n &optional (p 0.5))
  (do*
    ((i 0 (1+ i))
     (s (binomial n i p) (+ s (binomial n i p))))
    ((> i n) 't)
    (princ (1+ i)) (princ " ") (print (- 1 s))))
)

(defun normal (x &optional (mu 0) (rho 1))
  (1
   (*
    rho
    *sqrt-2-pi*
    (exp
      (/ (sqr (- x mu))
       (* 2 (sqr rho)))))))
)
(defun sqr (x) (* x x))

(setq *pi* 3.14159269)
(setq *sqrt-2-pi* (sqrt (* 2 *pi*)))

(defun binomial (n k &optional (p 0.5))
  (*
    (comb n k)
    (expt p k)
    (expt (- 1 p) (- n k))))

(defun comb (n k)
  (do*
    ((d (- n k))
     (i 0 (1+ i))
     (b 1 (* b (/ (+ d i) i))))
    ((eq i k) b)))

(defun print-g1 (participants)
  (princ ".LP") (terpri)
  (princ ".ps -1") (terpri)
  (princ ".G1") (terpri)
  (princ "frame ht 4 wid 4") (terpri)
  (princ "coord x -1,") (princ participants)
  (princ " y 0," ) (print participants))

(defun print-g2 ()
  (prog ()
    (princ ",G2") (terpri)
    (princ ",ps +1") (terpri)))