Finding Commonalities Between Friends is Harder than Expected: The Phase Transition of Computing Commonalities Between Agents and Objects.

Roberto Alonso, Raúl Monroy and Eduardo Aguirre*

Department of Computer Science,
Tecnológico de Monterrey, Campus Estado de México,
Carr. lago de Guadalupe Km 3.5, Atizapán, Estado de México, México

Abstract

We report on a study towards understanding the phase transition of an NP-complete problem, called Social Group Commonality (SGC). SGC is about finding commonality amongst the use of a set of objects by a given collection of agents; e.g., it is determining all the friends that have in common a given collection of Facebook users. In particular, our aim here is to study alternative candidates for the order parameter of the phase transition of SGC. To this aim, we have constructed a set of SGC instances, randomly sampling a real world process; we have attempted to solve all these instances, using a backtracking based algorithm, and labelling them according to the result of the solution attempt. Then, in order to distinguish what increases the computer expense at solving an instance, we have made C4.5 to build a classification tree, looking into a number of problem features, such as the number of objects, the number of agents, etc. Throughout our experimentation, we have found that an order parameter should be a function of the number of agents, and the size of a maximal group, amongst others. We conclude the paper, giving indications as to how to continue with the study of the phase transition of this problem.

Keywords: Phase transition, social structures, DNS

*Please, refer first to the corresponding author raulm@itesm.mx.
Email address: roberto.alonso@itesm.mx, raulm@itesm.mx, eduardo.aguirre@itesm.mx (Roberto Alonso, Raúl Monroy and Eduardo Aguirre)
1. Introduction

In 1991, Cheeseman et al. [1] showed that, for any NP-complete problem, there exists a phase transition that separates easy instances from hard ones, and that this phase transition can be found as one varies an order parameter\(^1\) around one or more critical values. Since then phase transition studies have been conducted for a number of NP-complete problems, see e.g. [3, 4, 5, 6]. This is because phase transition helps identifying key instances of a problem so as to build a benchmark set, with which one can perform a statistically significant comparison amongst different methods that attempt to solve that problem. Furthermore, the phase transition value can also be used to determine when a complete method is likely to succeed (or not) in finding a solution to a problem instance in reasonable time.

In this paper, we study the phase transition of an NP-complete problem, called Social Group Commonality (SGC) [7]. SGC is about finding the social structure that arises when a set of agents interact one another through executing actions upon a set of objects. It naturally arises in the context of detecting anomalies in DNS traffic: when observed over a time window, DNS queries (from IP addresses to URLs) form groups; this social structure is severely broken apart upon, e.g., a denial of service attack, as shown by Alonso [8].

SGC arises in various contexts. For example, [9, 10], proposed a method for topic (group) discovery, given a set of documents (agents); the method looks into the commonality of terms (objects) a document contains: the more common terms of a topic it refers to, the more related a document is to that topic. As another example, [11] proposed computing the commonality amongst the friends (objects) of a user (agent) so as to the adjust privacy settings (group) upon the appearance of a new friend, the rule being: sensitive information should be shared only if a new contact has several friends in common. [12] suggests that determining whether a set of animals (agents) have breeding areas (objects) in common helps identifying critical areas (groups) to protect. Finding commonalities amongst a set of agents is a common task in community detection; for example, [13, 14] all suggest that finding smaller communities may provide an insight of larger ones.

\(^1\)Following standard convention, we use “order parameter” to refer to a parameter that controls the complexity of finding a solution to a given problem instance [1], instead of using the more appropriate term control parameter [2].
For SGC, the phase transition shows an easy-to-hard-to-easy pattern. A critical value occurs when the ratio of the size of a maximal group (see Section 2 below) to the number of URLs is roughly 3/8. There, any problem instance would be computationally expensive and with a 98% probability of being solvable (see Section 3). By contrast, when this ratio tends to either zero or one, dealing with a problem instance is negligible, being insoluble in the former case, and solvable in the latter one.

Paper overview. The rest of the paper is structured as follows. We first introduce general knowledge of DNS and formulate SGC Section 2. Next, we outline how to conduct a phase transition study, Section 3, and then introduce our experimental methodology for determining a suitable order parameter, Section 4. Then, we present an algorithm for SGC, which applies backtracking and a black list in order to eagerly discard agents or objects that cannot form part of large groups, Section 5. Finally, in Section 6, we report on the phase transition of SGC, and, in Section 7, on the conclusions drawn from our investigations.

2. Social Structure Stemming from DNS Usage

The Domain Name System (DNS) is a distributed naming service for computers. It is mostly used to translate URLs into IP addresses, which is required for the localization of computer services worldwide. DNS is hence a critical service, and, not surprisingly, a common target of cybercrime, often to a Distributed Denial of Service (DDoS) attack.

DNS traffic consists of a sequence of DNS queries, each of which involves an IP address (henceforth called an agent) and a URL (henceforth called an object). When observed over a period of time (called a window), DNS traffic gives naturally rise to a social structure: agents are grouped together according to the commonality of the objects they have queried for.

In this section, we shall provide a formal definition of this social structure, the social group commonality problem, and the function symbols that are to be used throughout this document.

2.1. Social Group

Let \(\mathcal{W}\) be the set of all windows, ranged over by \(w_1, w_2, \ldots\); \(A\) the set of all agents, ranged over by \(a_1, a_2, \ldots\); and let \(\mathcal{O}\) be the set of all objects, ranged over by \(o_1, o_2, \ldots\). We shall use \(\text{qry}^w(a, o)\) to denote that agent \(a\) has queried object \(o\) during window \(w\).
object $o$ over window $w$. Then, the set of active agents, with respect to a
given window $w \in \mathbb{W}$, is given by $\text{agt}(w) = \{x \in A \mid \exists y \in O, \text{qry}^w(x, y)\}$. Likewise, the set of objects agents have queried for is defined as $\text{obj}(w) = \{y \in O \mid \exists x \in A, \text{qry}^w(x, y)\}$.

We first define social group, as well as a means of comparing groups one
another.

**Definition 1 (Social Group).** Let $w$ be a window, and let $A \subseteq \text{agt}(w)$ and $O \subseteq \text{obj}(w)$. Then, $\langle w, A, O \rangle$, written $g^w(A, O)$ for short, is a
social group of size $|O|$, and weight $|A|$, iff every agent in $A$ has queried all objects
in $O$: $\forall x \in A. \forall y \in O. \text{qry}^w(x, y)$; put another way: $\text{qry}^w$ is the Cartesian
product of $A \times O$.

**Definition 2 (Size-/Weight-Maximal Group).** Let $G^w$ denote all the groups in a given window $w \in \mathbb{W}$. Then, a group $g^w(A, O) \in G^w$ is called size-maximal (respectively, weight-maximal) if there does not ex-
ist $g^w(A', O') \in G^w$ such that $|O| < |O'|$ (respectively, $|A| < |A'|$). We call a
group trivial, if it is of size or weight equal to one, and non-trivial, otherwise.

**Definition 3 ($\leq$, $\prec$, Maximal Group).** Let $w \in \mathbb{W}$ be a window, and let $G^w$ denote all groups in $w$. Further, let $\leq$ and $\prec$ be defined as follows:

- $(a, c) \leq (b, d)$ iff $a < b$, or $a = b$ and $c \leq d$, and
- $s \prec s'$ if $s \leq s'$ and $s \neq s'$.

Then, a group $g^w(A, O) \in G^w$ is called maximal if there does not exist
$g^w(A', O') \in G^w$ such that $(|O|, |A|) \prec (|O'|, |A'|)$.

Given a window, $w \in \mathbb{W}$, the queries of agents $\text{agt}(w)$ over the correspond-
ning collection of objects $\text{obj}(w)$ can be succinctly represented by means of an
adjacency query matrix, introduced below.

**2.2. The Adjacency Query Matrix**

An adjacency query matrix, $Q^w$, of size $|\text{agt}(w)| \times |\text{obj}(w)|$, is such that
$Q^w_{(i,j)} = m$ implies that agent $a_i$ has queried $m$ times object $o_j$ across $w$. The gram matrix of $Q^w$, denoted $\text{gram}(Q^w)$ and given by $Q^w \times (Q^w)^\top$, pro-
vides valuable information about the activity of agents in $w$. In particular, $\text{gram}(Q^w)$ is symmetric, and such that the lower (respectively, upper)
triangular matrix contains information about all the distinct groups with
weight equal to two, including the one that is size-maximal. Notice that $\text{gram}(Q^w)_{(i,j)} = n, i \neq j$, implies that $w$ contains a 2-weight group of size
equal to $n$, involving the participation of agents $a_i$ and $a_j$. The main diagonal
of this matrix enables us to determine the number of actions performed by a top active agent in \( w \), since \( \text{gram}(Q^w)_{(i,i)} = n \) implies that agent \( a_i \) visited \( n \) distinct objects along \( w \).

Complementarily, \( \text{gram}((Q^w)^\top) \), given by \((Q^w)^\top \times Q^w\), provides valuable information about the popularity of objects in \( w \). In particular, the lower (respectively, the upper) triangular matrix of this matrix contains all the distinct groups of size two, including the one that is weight-maximal. Again, \( \text{gram}((Q^w)^\top)_{(i,j)} = n \) implies that \( w \) contains a 2-size group of weight equal to \( n \), involving the use of \( o_i \) and \( o_j \). \( \text{gram}((Q^w)^\top)_{(i,i)} = n \) implies that object \( o_i \) has been queried \( n \) different times along \( w \), from so we determine the top popular object in \( w \).

2.3. Social Group Commonality

Now, the decision version of the Social Group Commonality problem, SGC, as introduced in [7], is defined as follows:

INSTANCE: A window \( w \in \mathbb{W} \), a positive integer, \( z > 0 \), and, a positive integer, \( t > 0 \).

QUESTION: Is there a group of size \( z \) and weight \( t \)?

SGC has been proven to be NP-complete [7]. What is more, even determining whether every agent in a window belongs to a non-trivial group is still NP-complete.

3. Phase Transition Study

Phase transition is a means of selecting problem instances that are typically hard, and hence provide a fair basis for comparison of different algorithms. A phase transition, separating easy instances from hard ones, appears as one plots the expense of finding a solution to a problem instance against an order parameter. Interestingly, it often coincides to that area where the problem, stated as a decision problem, changes from having a YES-solution (solvable) to one having not (insoluble). The term is used in an analogy to the Physics phenomena: after a phase transition, a material dramatically changes its properties, e.g. from liquid to solid.

Some problems have been found to show an easy-to-hard-to-easy complexity pattern (e.g., travelling salesman [1, 15]): the cost of finding a solution increases at first, but then decreases later on to small values back again. Others (e.g. constraint satisfaction [16]) have been found to have similar cost,
Scaling has several implications; it can be used to construct an instance with a given probability, or a set of instances with similar cost, and this can be done for any problem size.

Conducting a phase transition study is a four-step approach:
1. Select an order parameter that succinctly captures the problem structure. This task may not be trivial. As pointed out by [1], using a different order parameter yields a different phase transition. While several NP-complete problems exhibit a natural order parameter, others require experimental evidence; e.g. Gent and Walsh [17] used an annealed theory to determine an order parameter for the phase transition of number partitioning. One of the contributions of this paper is to show how to apply C4.5 [18] to identify a suitable order parameter for SGC (see Section 4).
2. Collect a number of problem instances. This can be done either by randomly generating problem instances using the selected order parameter, or by collecting them from a real-world process, if any. For our study, we have collected a set of SGC instances randomly sampling real traffic logs from a recursive DNS server.
3. Select an algorithm that solves the problem, and then apply it on each instance of the set built from the second step; for each try, gather both computational expense and whether it is solvable or not. We shall introduce our algorithm for SGC in Section 5. Notice that, alternatively, we could use an efficient algorithm, e.g. a SAT solver; but then we would have to come out with a mapping from SGC to SAT, which is well beyond the scope of this paper.
4. Plot the computational cost found in the previous step against the order parameter, and then superimpose a plot of the probability of an instance being solvable against the order parameter.

4. On Identifying an Order Parameter

In order to identify an order parameter for SGC, we have characterized the commonality amongst both the instances that can be solved with little effort, and those that cannot. To that purpose, we have applied C4.5, which builds a decision tree from a training set containing already classified samples. This tree can be separated into decision rules, which explain what makes an instance to be one class or the other. In what follows, we first describe our
working dataset, and then how C4.5 was applied to it to discover an order parameter.

4.1. Dataset Construction

Our dataset, including both training and testing, has been built out of a number of windows, each of which comes from a traffic log of a DNS server. These kinds of logs are usually pretty long, comprising in average 2.4M queries. Following [19], to build a representative dataset, we first arbitrarily picked a few DNS logs (five in our case). Then, from each log, we randomly sampled a query window of a given size, 50 at first. We next repeated this window selection step as many times so as to attain a set of windows amounting to 40% of each log. We then repeated in turn this whole procedure for windows of size 75, 100, . . . , 150.

For each SGC instance in this collection, we proceeded as follows. First, we preemptively applied our algorithm in order to find a maximal group. If the instance could be solved in less than 20 seconds, we labelled it EASY; otherwise, we gave up solving it and labelled it HARD. Then, we inserted in the final dataset a tuple containing a feature vector representing the instance, and the associated instance label. We finally split this dataset, forming the training set (comprising 70% of the data) and the test set (comprising the remaining 30%).

4.2. Features Used to Characterize a Problem Instance

We now show the feature vector representing a problem instance. We insist that in the selection of all these features, we were driven by determining an order parameter, and that they all capture the likelyhood of an instance being HARD. These features are:

- $|\text{agt}(w)|$: the number of active agents in $w$.
- $|\text{obj}(w)|$: the number of queried objects in $w$.
- $H_{\text{agt}}(w)$: the entropy of agents’ activity over $w$. Take the sequence of queries in $w$, and form a probability distribution function as follows. For each agent $a_i \in \text{agt}(w)$, compute $w_{a_i}$, the sequence that results from deleting every query from $w$ other than those issued by $a_i$. Then, the probability that agent $a_i$ has issued a query in $w$, denoted $\Pr(q_{w_i})$, is given by $\Pr(q_{w_i}) = \frac{\text{length}(w_{a_i})}{\text{length}(w)}$. With this, we define agent entropy as:

$$H_{\text{agt}}(w) = - \sum_{k=1}^{\text{length}(w)} \Pr(q_{w}^{k}) \log_{2}(\Pr(q_{w}^{k}))$$

7
• $H_{\text{obj}}(w)$: the entropy of objects over $w$, which is defined likewise, except that, instead of $w_a$, we use $w^o_j$, the sequence that results from deleting every query from $w$ other than those that refer to object $o_j$.

• $|\{\text{qry}^w(x,y) \mid x \in \text{agt}(w), y \in \text{obj}(w)\}|$: the number of distinct agent queries, called the query length.

• An approximation of the total number of object combinations that need to be attempted to search for non-trivial groups, denoted $\text{comb}(w)$. Take a window $w$, compute $\text{gram}(Q)$, and then look for the three highest values of the lower triangular gram matrix and multiply these values.

• The ratio $\frac{|\text{agt}(w)||\text{obj}(w)|}{\text{length}(w)}$, we call the social degree of $w$.

• The number of queries issued from a most active agent in $w$.

• The number of queries issued to a most popular object in $w$.

• The size of a maximal group in $w$.

• The weight of a group with maximal weight in $w$.

• The number of groups with weight $t = 2$, computed from $\text{gram}(Q)$.

• Likewise, the number of groups with size $z = 2$.

4.3. Construction of the Classifier

We have built seven classifiers, one per window size considered in our dataset. The rationale behind this design decision is to observe whether, and if so how, window size is part of the order parameter. We have built each classifier using ten-fold cross-validation. Roughly, we first randomly picked 90% of the training set. Then, we obtained a classification tree from these data, using C4.5, as implemented in Weka [20]. Second, we tested the tree on the remaining 10% of the instances. Third, we repeated this procedure 10 times. Finally, we selected the best classification tree and validated it on the test set. The corresponding results are reported on below.

4.4. An Evaluation of the Classification Tree Performance

Table 1 shows the false positive rate and the output by our classifier, for various window sizes. The false positive rate (FPR) is the rate at which the classifier mistakes a HARD instance to be EASY, and the false negative rate (FNR) the other way round. There usually is a trade-off between these rates. Notice how the FPR grows as the window size does. This is explained by the larger the size of a window, the larger the proportion of instances labelled HARD. This implies that in the dataset, both training and validation, classes are not balanced. Thus, it is more likely that a HARD instance is wrongly classified.
Table 1: Classifier outcome: FPR and FNR.

<table>
<thead>
<tr>
<th>Window size</th>
<th>FPR</th>
<th>FNR</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.770</td>
<td>0.259</td>
</tr>
<tr>
<td>150</td>
<td>5.609</td>
<td>0.199</td>
</tr>
<tr>
<td>200</td>
<td>13.370</td>
<td>0.197</td>
</tr>
<tr>
<td>250</td>
<td>17.5</td>
<td>0.259</td>
</tr>
</tbody>
</table>

Figure 1: ROC curves for windows with size 100, 150, 200, and 250.

More thoroughly, we have evaluated the performance of our classifier using Receiver Operating Characteristic (ROC) curves (see Figure 1). A ROC curve is a parametric curve, generated by varying a threshold and computing both the FPR and the FNR, at each operating point. The upper and the further left a ROC curve is, the better the classifier is. Figure 1 shows that our classifier is able to recognise instances. Notice that the classifier performance improves with the window size.

In order to support these results, we have plotted precision over recall. Here, the upper and further to the right the curve is, the better classifier is. Figure 2 shows again how the classifier performance improves with the window size. This can be attributed to both class unbalance, and to the
occurrence of a higher proportion of HARD instances in windows of large size.

4.5. Order Parameter Discovery from Classification Tree

The classification trees we have obtained for all datasets are remarkably similar, regardless of the dataset window size. Figure 3 displays the one for a window size equal to 250. In general, the rules extracted out of these classification trees show that the features that separate a HARD instance from an EASY one are size of maximal group, query length, number of 2-weight groups, number of objects and number of agents.

We also noticed that a large number of instances of type HARD are captured by one rule, namely: *label instance HARD, if size of maximal group and number of objects in w are respectively greater than 3 and 57*. We constructed a classifier considering this rule only. Figure 4 shows that this classifier is able to distinguish most of the HARD instances, regardless the size of the window.

Considering this result, we have come up with the following order parameter, $\pi$. Let $z_{\text{max}}(w)$ denote the size of a maximal group in window $w$, then
Figure 3: C4.5 classification tree for window size 250. In this figure, query(w) stands for the length of distinct queries in w, and freq_t(w) is the number of groups with weight fixed to t = 2.

\[ \pi \overset{\text{def}}{=} \frac{z_{\max}(w)}{|\text{obj}(w)|} \], with \( \pi \in [0, 1] \). Notice that, as \( z_{\max}(w)/|\text{obj}(w)| \to 1 \), finding a maximal group is computationally harder, as we might need to explore the entire search space. By contrast, as \( z_{\max}(w)/|\text{obj}(w)| \to 0 \) most the search space can be pruned, since it is easy to discard the existence of large groups.

5. An Algorithm to Compute SGC

To conduct our phase transition study, we have used Algorithm 1, which, given a window and two integers (k and t), returns a group of size k and weight t, if any, along with the computational cost incurred. Function CommonAgentsQuerying(C) returns a set of agents, which have queried for all the combinations of objects in C. Notice that in lines 10-11 the algorithm returns a group with the corresponding objects and agents, if any; otherwise, it returns noSolution.

Following [21, 16, 15], we use the number of combinations explored as a measure of computational expense. The rationale behind this decision is

\[ \pi \overset{\text{def}}{=} \frac{z_{\max}(w)}{|\text{obj}(w)|} \], with \( \pi \in [0, 1] \). Notice that, as \( z_{\max}(w)/|\text{obj}(w)| \to 1 \), finding a maximal group is computationally harder, as we might need to explore the entire search space. By contrast, as \( z_{\max}(w)/|\text{obj}(w)| \to 0 \) most the search space can be pruned, since it is easy to discard the existence of large groups.

5. An Algorithm to Compute SGC

To conduct our phase transition study, we have used Algorithm 1, which, given a window and two integers (k and t), returns a group of size k and weight t, if any, along with the computational cost incurred. Function CommonAgentsQuerying(C) returns a set of agents, which have queried for all the combinations of objects in C. Notice that in lines 10-11 the algorithm returns a group with the corresponding objects and agents, if any; otherwise, it returns noSolution.

Following [21, 16, 15], we use the number of combinations explored as a measure of computational expense. The rationale behind this decision is

\[ \pi \overset{\text{def}}{=} \frac{z_{\max}(w)}{|\text{obj}(w)|} \], with \( \pi \in [0, 1] \). Notice that, as \( z_{\max}(w)/|\text{obj}(w)| \to 1 \), finding a maximal group is computationally harder, as we might need to explore the entire search space. By contrast, as \( z_{\max}(w)/|\text{obj}(w)| \to 0 \) most the search space can be pruned, since it is easy to discard the existence of large groups.

5. An Algorithm to Compute SGC

To conduct our phase transition study, we have used Algorithm 1, which, given a window and two integers (k and t), returns a group of size k and weight t, if any, along with the computational cost incurred. Function CommonAgentsQuerying(C) returns a set of agents, which have queried for all the combinations of objects in C. Notice that in lines 10-11 the algorithm returns a group with the corresponding objects and agents, if any; otherwise, it returns noSolution.

Following [21, 16, 15], we use the number of combinations explored as a measure of computational expense. The rationale behind this decision is

\[ \pi \overset{\text{def}}{=} \frac{z_{\max}(w)}{|\text{obj}(w)|} \], with \( \pi \in [0, 1] \). Notice that, as \( z_{\max}(w)/|\text{obj}(w)| \to 1 \), finding a maximal group is computationally harder, as we might need to explore the entire search space. By contrast, as \( z_{\max}(w)/|\text{obj}(w)| \to 0 \) most the search space can be pruned, since it is easy to discard the existence of large groups.

5. An Algorithm to Compute SGC

To conduct our phase transition study, we have used Algorithm 1, which, given a window and two integers (k and t), returns a group of size k and weight t, if any, along with the computational cost incurred. Function CommonAgentsQuerying(C) returns a set of agents, which have queried for all the combinations of objects in C. Notice that in lines 10-11 the algorithm returns a group with the corresponding objects and agents, if any; otherwise, it returns noSolution.

Following [21, 16, 15], we use the number of combinations explored as a measure of computational expense. The rationale behind this decision is

\[ \pi \overset{\text{def}}{=} \frac{z_{\max}(w)}{|\text{obj}(w)|} \], with \( \pi \in [0, 1] \). Notice that, as \( z_{\max}(w)/|\text{obj}(w)| \to 1 \), finding a maximal group is computationally harder, as we might need to explore the entire search space. By contrast, as \( z_{\max}(w)/|\text{obj}(w)| \to 0 \) most the search space can be pruned, since it is easy to discard the existence of large groups.

5. An Algorithm to Compute SGC

To conduct our phase transition study, we have used Algorithm 1, which, given a window and two integers (k and t), returns a group of size k and weight t, if any, along with the computational cost incurred. Function CommonAgentsQuerying(C) returns a set of agents, which have queried for all the combinations of objects in C. Notice that in lines 10-11 the algorithm returns a group with the corresponding objects and agents, if any; otherwise, it returns noSolution.

Following [21, 16, 15], we use the number of combinations explored as a measure of computational expense. The rationale behind this decision is

\[ \pi \overset{\text{def}}{=} \frac{z_{\max}(w)}{|\text{obj}(w)|} \], with \( \pi \in [0, 1] \). Notice that, as \( z_{\max}(w)/|\text{obj}(w)| \to 1 \), finding a maximal group is computationally harder, as we might need to explore the entire search space. By contrast, as \( z_{\max}(w)/|\text{obj}(w)| \to 0 \) most the search space can be pruned, since it is easy to discard the existence of large groups.

5. An Algorithm to Compute SGC

To conduct our phase transition study, we have used Algorithm 1, which, given a window and two integers (k and t), returns a group of size k and weight t, if any, along with the computational cost incurred. Function CommonAgentsQuerying(C) returns a set of agents, which have queried for all the combinations of objects in C. Notice that in lines 10-11 the algorithm returns a group with the corresponding objects and agents, if any; otherwise, it returns noSolution.

Following [21, 16, 15], we use the number of combinations explored as a measure of computational expense. The rationale behind this decision is

\[ \pi \overset{\text{def}}{=} \frac{z_{\max}(w)}{|\text{obj}(w)|} \], with \( \pi \in [0, 1] \). Notice that, as \( z_{\max}(w)/|\text{obj}(w)| \to 1 \), finding a maximal group is computationally harder, as we might need to explore the entire search space. By contrast, as \( z_{\max}(w)/|\text{obj}(w)| \to 0 \) most the search space can be pruned, since it is easy to discard the existence of large groups.

5. An Algorithm to Compute SGC

To conduct our phase transition study, we have used Algorithm 1, which, given a window and two integers (k and t), returns a group of size k and weight t, if any, along with the computational cost incurred. Function CommonAgentsQuerying(C) returns a set of agents, which have queried for all the combinations of objects in C. Notice that in lines 10-11 the algorithm returns a group with the corresponding objects and agents, if any; otherwise, it returns noSolution.

Following [21, 16, 15], we use the number of combinations explored as a measure of computational expense. The rationale behind this decision is
because it is not affected by the hardware on which experiments are run. This is in contrast with other measures, such as time-to-solve (used \textit{e.g.} in [22]). We have solved over 100 thousand of SGC instances using several computers.\textsuperscript{3}

6. The Phase Transition of the SGC Problem

We have conducted the phase transition study on both decision and optimality formulation of SGC considering the size of a maximal group divided by the number of objects and the size of a maximal group, respectively, as order parameters (see. sec 4).

6.1. Phase Transition of the SGC Decision Problem

Identifying the phase transition on the decision version of SGC aims to determine if the hardness and solvability property exhibit patterns as many well-known NP-complete problems. In order to observe such patterns, the

\textsuperscript{3}Our experimentation on this work lasted about 6 months of continuous calculations using two computers: the first one with a Core i7 2Ghz computer with 4 GB in RAM and the second one with two Xeon 3GHz processor with 8 GB in RAM.
Algorithm 1 Backtracking based approach to solve a SGC instance.

**Input:** A window \( w \) that contains actions from agents to objects and a size \( z \) and weight \( t \) of the group being looked for.

**Output:** A group with size \( z \) and maximal weight \( t \) with the corresponding witness, if any.

1: \( r \leftarrow 2 \)
2: \( \text{BlackList} \leftarrow \emptyset \)
3: while \( z \geq r \) do
4: \( C \leftarrow \text{obj}(w)^{r} \) // \( C \) is set of sets of objects
5: for all \( C \) in \( C \) do // \( C \) is a set of objects.
6: if \( C \notin \text{BlackList} \) then
7: \( \text{agt} \leftarrow \text{CommonAgentsQrying}(C) \) // \( \text{agt} \) is a set of agents.
8: if \( |\text{agt}| > 2 \) and \( z = r \) then
9: return \( g\{\text{agt}, C\} \)
10: // Return a group witnessed by agents and objects.
11: else
12: if \( |\text{agt}| < 2 \) then
13: \( \text{BlackList} \leftarrow \text{BlackList} \cup C \)
14: end if
15: end if
16: end if
17: end for
18: \( r = r + 1 \)
19: end while
20: return \( \text{noSolution} \)

The computational expense, the number of explored combinations in our case, is plotted against the order parameter, \( z_{\text{max}}(w)/|\text{obj}(w)| \).

Typically, this graph shows a region of hard instances where the computational expense is high with respect to the instances considered as manageable. Also, this graph could show a dramatic change in the hardness of solving the problem, going from an easy region to a hard region, the so-called easy-hard pattern.

It is possible to study the solvability property of a problem by plotting the probability of finding a solution (e.g., for the SAT problem a set of values satisfying a boolean formula) against the order parameter. Here, the probability of being solvable is computed by the function, if synthesized,
that generates the random instances, and if instances were taken from a real-
process they can be determined by the frequency of the solvable instances
divided by the total instances in a given value of the order parameter.

After plotting this graph, it is possible to notice changes in the solvability
property where, for example, a problem may go from a region with solution,
to region where instances cannot be solved after a dramatic change, namely
the phase transition.

Both the plot of hardness and the plot of solvability are usually superim-
posed so as to better illustrate the results of these properties.

6.1.1. Experimental Setting

We have randomly sampled from the DNS server under study, SGC in-
stances with a window of size from 50 to 150 in steps of 25 so as attain a
collection of 100 thousand instances. Then, we have solved them consider-
ing our backtracking based approach and reported the computational cost of
finding a group of size $z$.

For now, we have restrained to study groups with size $z$ from 4 to 6 with
weight $t \geq 2$, the so-called non-trivial groups. Henceforth, for the sake of
simplicity, we shall refer to a non-trivial group with size $z$ and weight $t \geq 2$
as a group with size $z$.

Attention is now turned into the resulting graphs.

6.1.2. Results

We first report our results considering groups of size $z = 4$. Figure 5
shows, in a solid line, the median cost of finding a group with size $z = 4$; the dashed line shows the percentile 25%, the easiest instances of SGC,
while the dotted line shows the percentile 90%, the hardest instances of
SGC; lastly, the dots indicate the probability of having a solution. Notice
that $z_{\text{max}}(w)/|\text{obj}(w)|$ should go from 0 to 1 but to illustrate better the
results, the order parameter was plotted in logarithmic scale, in symbols
$\hat{\pi} = \log_2(z_{\text{max}}(w)/|\text{obj}(w)|)$.

We can notice from the figure an easy-hard-easy pattern. Moreover, the
hardest instances happened at $\hat{\pi} = -1.5$ with a computational cost of 1400.
Notice that even if we consider the percentile 25%, the computational cost
remains similar at $\hat{\pi} = -1.5$ with 1100 explored combinations. Notice that
after the inflection point $\hat{\pi} = -1.5$, there is a decrease in the computational
cost until we reach a cost of 462 combinations at $\hat{\pi} \approx 0$. These results
suggest that finding a group with size approximately the number of objects
Figure 5: Percentile 90%, 25% and median cost of finding a group with size $z = 4$ and weight $t \geq 2$ for window sizes 50 to 150 in steps of 25.
in a window is relatively easier than expected. An explanation about this behaviour could be attributed to the fact that there is only one large group in the window; then, the algorithm easily discards the combinations that do not belong to this large and single group.

Finally, notice that before reaching a probability of 50% when \( \hat{\pi} < -4 \), it is relatively easy to solve SGC although it is unsolvable. By contrast, when \( \hat{\pi} > -4 \) it is possible to find solvable instances but with a high computational cost.

Our experimentation for \( z = 5 \), in Figure 6, shows the same easy-hard-easy pattern. Notice the inflection point \( \hat{\pi} = -2 \) before a sudden increase in the computational cost. At \( \hat{\pi} = -1.5 \) we have the hardest instances indicated by the percentile 90% with cost of 7000 combinations. After \( \hat{\pi} = -1.5 \) the computational cost decreases until it reaches a cost of 2000 combinations at \( \hat{\pi} \approx 0 \). Notice that the percentile 25% and the median show a similar behaviour both on its inflection points and in the computational cost.

![Phase transition plot for SGC with z=5](image)

Figure 6: Percentile 90%, 25% and median cost of finding a group with size \( z = 5 \) and weight \( t = 2 \) for window sizes 50 to 150 in steps of 25.

The resulting plot of studying SGC with \( z = 6 \) show the same easy-hard-easy pattern on its cost. Notice the same significant inflection points
at $\hat{\pi} = -2$ and in $\hat{\pi} = -1.5$. The hardest instances have a cost of 28000 combinations considering the percentile 90% at $\hat{\pi} = 1.5$ However, notice that near $\hat{\pi} = -1.5$ a sudden decrease in the computational cost, then the computational cost increases and drastically decreases before reaching 5000 combinations at $\hat{\pi} \approx 0$. Lastly, notice that considering the percentile 25% the hardest instances have cost of 18000 explored combinations.

Figure 7: Percentile 90%, 25% and median cost of finding a group with size $z = 6$ and weight $t = 2$ for window sizes 50 to 150 in steps of 25.

6.1.3. Discussion

For $z$ from 4 to 6, there is an inflection point at the maximum cost when $\hat{\pi}$ is near -1.5. Particularly, $\hat{\pi} = -1.5$ is $z_{\text{max}}(w)/|\text{obj}(w)| = 0.375$ meaning that the hardest instances are when there are 8 times more objects in the window than groups of size $z = 3$. In other words, the hardest instances follows a relation 3 to 8, in symbols $z_{\text{max}}(w)/|\text{obj}(w)| = 3/8$.

Moreover, we have evidenced that SGC exhibit the same easy-hard-easy pattern regardless of the size of the group and similar inflection points but with different computational costs.
As another observation, some well-known NP-complete problems exhibit the hardest instances at the region where there is 50% probability of having a solution. By contrast, we have showed that the hardest instances of SGC are at the region with solution. This result is relevant since an efficient algorithm may discard instances with a low chance of being solvable and focus on instances with a high probability of being solvable.

In general, an instance with \( \hat{\pi} < -4 \) is easy but unsolvable while an instance is hard but solvable when \( \hat{\pi} > -4 \).

After the major inflection point at \( \hat{\pi} = -1.5 \) there is a decrease on the computational cost of finding a group. The explanation for this is because after this inflection point the size of a maximal group is reaching the number of objects in the window, and our algorithm is easily discarding combinations that will not be used. This result shows that groups of size \( z = |w| \) are uncommon and easy to solve.

We have evidenced that SGC follows patterns similar to other NP-complete problems. Particularly, this problem goes from unsolvable to solvable drastically after exceeding \( \hat{\pi} = -4 \), namely when there are 16 times more objects than the size of the maximal group in a window.

### 6.2. Phase Transition on the Optimality Version of SGC

We now investigate the decision version of SGC where we want to find a maximal group.

#### 6.2.1. Experimental Setting

In this experimentation we have considered instances with windows of size from 50 to 150 in steps of 25.

Following an approach similar as in [17], we have considered to use the size of a maximal group as the order parameter. Also, the selection of this parameter is supported by our results from the C4.5 classifier (see section 4) which suggest this parameter as a good candidate, we shall refer to this parameter as \( z_{\text{max}}(w) \).

Next, we have split instances according to the window size in turn and plotted the mean cost of finding a group with size \( z = z_{\text{max}}(w) - d \). Where \( d \in \{1, 2, 3..., z_{\text{max}}(w) - 2\} \) is defined as the distance from the optimal solution, i.e. a maximal is found, in a given instance of SGC. At \( d = 0 \), namely \( z = z_{\text{max}}(w) \), a maximal is found while at \( d > 0 \), there is not a maximal group but we may find a group which could be part of a maximal. Notice
that we have ranged over these values because with the Gram matrix we have
certainty about the size of a maximal.

Last, we modify our algorithm (see. section 5) so as to report on the
computational cost of finding a group with size \( z = z_{\text{max}}(w) + d \). Since
finding a maximal group, implies finding all the groups of size \( z < z_{\text{max}}(w) \).

6.2.2. Results

Figure 8 for a window with size 75 and 100, and Figure 9 considering a
window with size 125 and 150, shows two type of behaviours. In the region
with solution, there is an exponential growth as the phase transition, i.e. the
boundary between solvable and unsolvable instances indicated by the dashed
line, is approached because we are reaching to a group with maximal size.
In the region without a solution, when \( d \to \alpha \) the computational cost gets
to zero because, according to the Gram matrix, there are not groups bigger
than a maximal.

Note that the mean cost of finding a maximal group is 1397, 17874, 31293,
51604 for the windows with size 75, 100, 125 and 150 respectively.

6.2.3. Discussion

Notice from the graphs that windows with size from 75 to 150 have an
inflection point when \( d = 20 \) showing that there is a maximal group with
size 20 or more. This suggest that even in smaller windows there are large
groups, in terms of its size. Also, after \( d = 20 \) the computational cost of
finding a bigger group increases until it reach \( d = 10 \).

Indeed, at \( d = 10 \) there is a drastic change in the computational cost
before reaching the size of the maximal at \( d = 0 \). This suggest that when
\( d < 10 \) half of the search space can be explored with half of the time to
explore the combinations. While after exceeding \( d = 10 \) means that it will
be expensive to solve the optimality version of SGC.

7. Conclusions and Indications for Further Work

Studying the computational complexity of intractable problems via phase
transition is an experimental way to understand better such problems. In
this work we have studied the Social Group Commonality problem. We
have evidenced that SGC behaves similarly to other well-known NP-complete
problems in the sense that, there are a number of problem instances for
which it is manageable to determine a solution. Moreover, SGC exhibits a
critical value at which the solvability and the hardness of the problem changes drastically, the so-called phase transition.

We have shown experimental evidence that our proposed order parameter is suitable for the SGC phase transition study. The experimentation shows that, amongst the selected features of SGC, there are three significant fea-
Our results showed that before we reach the phase transition, at the order parameter $z_{\text{max}}(w)/|\text{obj}(w)| = 0.0625$, i.e. $\tilde{\tau} = -4$, there is a region of SGC for which it is possible to compute with bounded resources a solution.

Figure 9: Mean cost of finding a solution for the optimality version of SGC for a window with size 125 and 150.

atures; namely, the size of a maximal group, the number of objects, and the number of 2-weight non-trivial groups.
which is negative, i.e. no group with size $z$ is found. By contrast, when $0.0625 < z_{\text{max}}(w)/|\text{obj}(w)| < 0$, i.e. $-4 < \hat{\pi} < 0$, there is a solution for SGC but with an increase in the computational cost.

This study used the order parameter $k_{\text{max}}/|\text{obj}(w)|$ to identify the phase transition on the decision version and $k_{\text{max}}$ on the optimality version of SGC. Roughly, the results shows that even if we look for groups with different sizes $k$, the mean cost follows the easy-hard-easy-hard-easy pattern and that the search space can be reduced drastically if we somehow find a way to predict which objects will not form a maximal group.

It is possible to keep studying SGC by means of larger windows. However, our initial experimentation have shown that larger windows may take months to be completed with our algorithm. To address this, we are currently working on using a MapReduce approach considering the number of explored combinations as the measure of computational expense.

Our results can be used to design efficient algorithms attempting to solve SGC given that they provide a fair baseline of algorithm comparison. For example, in order to test the performance of an algorithm it is possible to select the hardest instances that according to our results involves exploring 28000 combinations and test if the algorithm can solve more efficiently those instances.

Any method, e.g. heuristic based, aiming to compute the social structure will improve its performance if it considers our results.


