Finding Skilled and Socially Cohesive Teams in Large Scale Social Networks

Alex Kantchelian

***Abstract*—This work explores a mathematical formalization of the team-maker problem. In the team-maker problem, one is given a set of candidates along with all the pairwise relationships of a certain kind between them. Each candidate has a given set of skills, and the team-maker's task is to choose a set of candidates that will form a skilled and socially cohesive team. We propose and evaluate an exact algorithm for the task, based on an Integer Linear Programming formulation.**

I. The team-maker problem

This section defines and formalizes the team-maker problem. Generally speaking, the team-maker problem consists in selecting a set of team members out of a set of candidates such that the resulting team is both skilled and socially cohesive. Building on a social network approach, we furthermore propose a formal notion of social cohesiveness.

Definition ($\mathcal{R}$-cohesiveness) Let C be the set of all candidates. Let $\mathcal{R}$ be a symmetric binary relation over C. We say that a subset S of C is $\mathcal{R}$-cohesive if for any two distinct candidates $a, b \in S$, there exists a possibly void sequence of candidates $s_1, \ldots, s_n$ all in S such that $a\mathcal{R}s_1$, $s_1\mathcal{R}s_2$, $\ldots$, $s_n\mathcal{R}b$.

For example, $\mathcal{R}$ can denote mutual *friendship*, or mutual *willingness to work with*. In the case of friendship, a given group of candidates is friendship-cohesive if and only if any two people can relate to each other through a possibly void sequence of candidates from the group, where each pair of successive candidates are friends.

Notice that specifying $\mathcal{R}$ is equivalent to specifying a graph $G_{\mathcal{R}} = (C, E_{\mathcal{R}})$ where the edges $E_{\mathcal{R}}$ are the pairs of elements that are in $\mathcal{R}$-relation. We can thus think of cohesiveness as a connectedness property on the social network graph $G_{\mathcal{R}}$.

Definition ($\mathcal{R}$-cohesiveness) Let $G_{\mathcal{R}} = (C, E_{\mathcal{R}})$ be a social network graph where edges represent property $\mathcal{R}$. We say that a subset S of C is $\mathcal{R}$-cohesive if the graph which vertices are S and edges are induced by $\mathcal{R}$ over S is connected.

Finally, we attach to each candidate $c \in C$ a list of skills $\mathrm{sk}(c) \subset \{0, \ldots, n_s - 1\}$ out of n_s possible skills. We can now give a complete formal definition of the team-maker problem.

Definition (team-maker problem) Let $G_{\mathcal{R}} = (C, E_{\mathcal{R}})$ be a social network graph where edges represent property $\mathcal{R}$. Let $n_s \in \mathbb{N}^*$ and

$$\mathrm{sk} : C \to \mathcal{P}(\{0, \ldots, n_s - 1\})$$

be the skills of the candidates C. Let $q \subset \{0, \ldots, n_s - 1\}$ be a query set of skills. The team-maker problem is to find a minimal $\mathcal{R}$-cohesive team such that for any skill in q, there exists a team member who has the skill.

In the following, we will indifferently speak of a node or a vertex for a candidate, and a color for a skill.

II. NP-Equivalence

In this section, we study the computational hardness of the team-maker problem (TMP) and we show that TMP is NP-Equivalent. We first begin by giving a decision version of the team-maker problem.

Definition (DTMP) Let $(G_{\mathcal{R}} = (C, E_{\mathcal{R}}), n_s, \mathrm{sk}, q)$ be a team-maker problem. Let $S \subset C$ and $k \in \{0, \ldots, |C|\}$. The decision team-maker problem $(G_{\mathcal{R}}, n_s, \mathrm{sk}, q, S, k)$ where the decision is to know whether or not there exists a connected set of vertices of size k that contains all the colors in q and includes S.

Lemma 2.1: TMP is polynomial-time reducible to DTMP and DTMP is polynomial-time reducible to TMP.

Proof:

- Let (G, n_s, sk, q) an instance of TMP. The following algorithm with a DTMP oracle solves the TMP instance in $O(|C|)$ steps. We give a proof of correctness. The

Algorithm 1 TMP polynomial time solver with DTMP-Oracle

```
k ← 0
while k ≤ |C| ∧ ∄DTMP(G, n_s, sk, q, ∅, k) do
  k ← k + 1
end while
if k = |C| + 1 then
  return ∅
end if
S_0 ← ∅
i ← 0
for c ∈ C do
  if DTMP(G_R, n_s, sk, q, S_i ∪ {c}, k) then
    S_{i+1} ← S_i ∪ {c}
  end if
  i ← i + 1
end for
return S_i
```

algorithm trivially terminates. Now, if $S_{|C|} = \emptyset$, then no solution exists by definition. If $S_{|C|} \neq \emptyset$, then we know that there exists a solution to TMP which elements are in $S_{|C|}$. We furthermore know that the minimal solution is of size k. Suppose $S_{|C|}$ is not a solution to TMP. Then,

there exists $S' \subset C, S' \neq \emptyset$ such that $S_{|C|} \cup S'$ is a solution to TMP. Then, there exists $c \notin S_{|C|}$ such that DTMP$(G_{\mathcal{R}}, n_s, \mathrm{sk}, q, S_{|C|} \cup \{c\}) =$ TRUE. Let k^* be such that DTMP is called with $S_{i^*} \cup \{c\}$. By definition, the call returns TRUE and c is appended to the solution. Contradiction.

- Let $(G_{\mathcal{R}}, n_s, \mathrm{sk}, q, S)$ an instance of DTMP. We construct in polynomial time $O(|S|)$ a corresponding TMP instance with the following inputs: $n'_s = n_s + |S|$; $\mathrm{sk}'(c) = \mathrm{sk}(c)$ if $c \notin S$, else $\mathrm{sk}'(c) = \mathrm{sk}(c) \cup \{n_s + k_c\}$ where k_c is unique per candidate integer between 0 and $|S|-1$; $q' = q \cup \{n_s + k | k \in \{0, \ldots, |S|-1\}\}$. It is straightforward to verify that $(G_{\mathcal{R}}, n'_s, \mathrm{sk}', q')$ has a solution if and only if the DTMP instance has a solution.

■

Proposition 2.2: DTMP is in NP.

Proof: A certificate for a DTMP instance is simply a connected (in the induced subgraph) set of vertices A of size k where are the colors of q are present and $S \subset A$. Checking whether this polynomial-size certificate is valid is done by verifying its connectedness in $O(|A|)$ time by any usual graph traversal technique, by inspecting the skills (or colors) of each vertex in $O(|A|n_s)$ time and by verifying that all the vertices of S are present in $O(|S|)$ time using the adequate data structures. ■

Proposition 2.3: DTMP is NP-Complete

Proof: By proposition 2.2, DTMP is in NP. We exhibit a reduction from 3-SAT. Let Φ be our 3-SAT formula in conjunctive normal form defined over m variables $\{x_1, \ldots, x_m\}$:

$$\Phi = C_1(x_{i_1}, x_{i_2}, x_{i_3}) \wedge \cdots \wedge C_n(x_{i_{3n-2}}, x_{i_{3n-1}}, x_{i_{3n}})$$

where the $(C_k)_{1 \leq k \leq n}$ are the n clauses. As a reminder, a clause is a disjunctive formula of three literals. A literal is either x or $\bar{x}$. For example, a valid clause is $C_1(x, y, z) = x \vee \bar{y} \vee z$.

We now embed the 3-SAT instance into a DTMC instance. The core ideas of our reduction are:

- to use a multi-floor structure. There is exactly one floor per clause in Φ.
- to use the connectedness constraint for building a path which traverses all floors using exactely one channel per floor if and only if there is an assignment of true and false values to the variables such that Φ is satisfied. Specifically, a floor is traversed by the solution if and only if there exists a realizable assignment such that one literal of the associated clause is true.
- to use the size k property to enforce the logical conditions.

Figure 1 depicts the structure of a floor. We will now give in turn a semi-formal description of the various parts.

a) Connectors: Surrounding floors, connector vertices each have a globally unique color. Thus, if a solution exists, all the connectors will be part of it.

b) Stoppers: Stopper vertices enclose channels at both ends, share the same color per row and every row has a globally unique color. As their name suggests, stopper cells are used to prevent a solution from entering two channels of the same floor, so that there can be only one selected channel per floor in a valid solution.

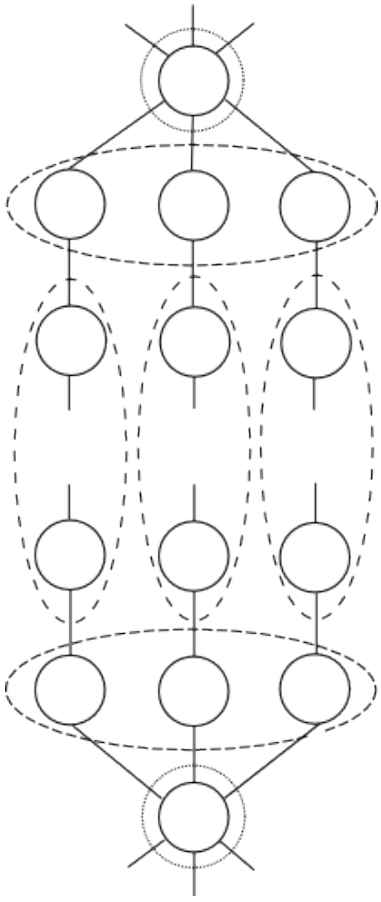

Fig. 1: Floor structure of $G_{\mathcal{R}}$ for embedding 3-SAT. There are n floors between $n+1$ enclosing connectors (sharply dashed). We distinguish three special families of vertices per floor: 2×3 stoppers (medium dashed) enclosing the 3 channels (medium sparse dashed). There are at most $3 \times (2 + 2(n-1))$ vertices between two connectors.

c) Channels: The channels are the central elements of the reduction. There are three channels per floor, corresponding to literals containing x_{i_3k-2}, x_{i_3k-1} and x_{i_3k} of clique C_k. We will choose the colors so that a valid solution passes through a channel if and only if n assignment to the corresponding variable such that the literal is true does not conflict with any other assignment to the same literal variable in any other clique.

There is a clear graphical representation of the situation. Let $G = (V, E)$ be a graph such that V is the set of clauses literals ($|V| = 3n$). An edge (v, v') exists if and only if:

1) v and v' are not from the same clause and
2) there exists a variable x such that $\{v\} \cup \{v'\} = \{x, \bar{x}\}$.

E can be constructed in time $O(n^2)$ by considering in turn every literal of every clause. Following our definition, the maximal degree of a vertex is $2(n-1)$ by noticing that a variable can appear in at most two distinct literals in every

clause. In the graph, a realizable assignment is simply an independent set of vertices.

Now, we arbitrary assign a unique color to every edge. It follows that an assignment is realizable if and only if the list of all adjacent edges to the selected vertices does not contain any repetition of color. Put in other words, the list is an independent set.

The filling strategy for the channel vertices is now clear. Each channel receives all the colors of the adjacent edges associated to its corresponding literal in graph G. As there is at most $2(n-1)$ adjacent edges, we fill the remaining vertices with one globally unique color per remaining vertex.

Constructing G and sk is possible in polynomial time. Indeed, there are $O(n^2)$ vertices in our construction, and thus also $O(n^2)$ colors. Filling the various regions of G (constructing sk) is also done in polynomial time by greedily attributing increasing integers for colors in this order for example:

1) Filling the connectors with globally unique integers is done in $O(n)$.
2) Filling the stoppers with globally unique but shared within a row integers is done in $O(n)$.
3) Filling the channels is done in $O(n^2+n^2) = O(n^2)$ by our previous discussion.

Finally, we claim that the 3SAT instance is satisfiable if and only if DTMP$(G, 6n^2-3n-1, \mathrm{sk}, q^*$end connectors$, 2n^2+n-1)$ returns TRUE, where $q*$ is the set of all colors except the filling colors in the channels. Indeed, $k = 2n^2+n-1$ is the diameter of G. The connectedness constraint implies that a solution exists if and only if exactly one channel is selected per floor. Note that $n_s = 6n^2-3n-1$ is the total number of vertices. ■

III. An Integer Linear Programming Solver Based algorithm

In this section, we describe an Integer Linear Programming formulation for the TMP and turn this formulation to an actual algorithm using a few heuristics.

A. An Integer Linear Programming formulation

Proposition 3.1: Let (G, n_s, sk, q) a TMP instance. Fixing an arbitrary indexation of the vertices, we implicitly designate a vertex by its index. Let $\phi(G) \subset \mathcal{P}(E)$ be the set of all cycles of graph G. The following 0/1 linear program describes a solution of the TMP where each team member gets assigned a single and unique skill in the team.

$$\min \sum_{i \in C} x_i$$

$$\text{s.t. (T)} \sum_{(i,j)\in E} y_{i,j} = \sum_{i \in C} x_i - 1$$

$$\text{(CC)} \ \forall c \in q, \sum_{i \in \mathrm{sk}^{-1}(c)} z_{c,i} = 1$$

$$\text{(XZ)} \ \forall i \in C, \forall c \in \mathrm{sk}(i), x_i \geq z_{c,i}$$

$$\text{(XY)} \ \forall (i,j) \in E, y_{i,j} \leq x_i; y_{i,j} \leq x_j$$

$$\text{(CE)} \ \forall \alpha \in \phi(G), \sum_{(i,j)\in\alpha} y_{i,j} < |\alpha|$$

Where $x \in \{0,1\}^{|V|}$ represent the vertices of the solution, $y \in \{0,1\}^{|E|}$ represent active edges associated with the vertices and $z \in \{0,1\}^{\sum_{i\in C} |\mathrm{sk}(i)|}$ are the color assignment variables.

The above formulation has 5 families of constraints briefly described below.

d) x-y consistency (XY): Once an edge is active, both adjacent nodes must be selected as well. There are $2|E|$ constraints in this family.

e) x-z consistency (XZ): Assigning color c to node i selects the node. There are $\sum_{i\in C} |\mathrm{sk}(i)|$ constraints in this family.

f) Color completeness (CC): Each color of the query set is exactly attributed once in the solution. There are $|q|$ constraints in this family.

g) Tree constraint (T): Coupled with the cycle elimination constraints (CE), the tree constraint enforces that the subgraph induced by the x and y variables is a tree. There is one tree constraint.

h) Cycle elimination constraints (CE): Coupled with the tree elimination constraint, these constraints forbid any cycles in the induced x, y subgraph, resulting in a connected subgraph, and thus enforcing the connectedness property. There is an exponential number of such constraints, so a direct implementation of this formulation will not scale properly. Instead, we do not immediately enforce these but will add them as necessary.

Proof: It is straightforward to verify that a solution to the TMP with the unicity per node and uniqueness of query colors is indeed a solution to the 0/1 formulation. The only delicate part for showing the equivalence is to verify that the (T) and (CE) constraints effectively enforce the connectedness of the solution described by the x variables. This basically relies on a small graph theoretic lemma which states that any graph with n vertices and $n-1$ edges which has no cycles is a tree and thus is connected. This can be easily shown by induction on n, by noticing that in a graph with no cycles, there always exists a vertex which has at most one adjacent edge and applying the induction property on the induced subgraph obtained by removing the node and its edge. ■

B. Algorithm Design

As stated above, the exponential number of cycle elimination constraints makes any direct attempt to solve this formulation impractical. However, we suspect that on most TMP instances, an overwhelming majority of the (CE) constraints are in fact redundant. Thus, our basic design choice is to initially leave those constraints out and to add them one by one to the formulation only when needed.

Our general approach is to take advantage of the performance of today's integer linear programming solvers to solve formulation 3.1. Looking at how the Traveling Salesman Problem is efficiently solved by the branch-and-cut approach, we decided to adopt a similar approach for the task.

Ideally, one would design a branch-and-cut algorithm for the formulation by first devising good problem-specific family of cutting planes, a good problem-specific heuristic for branching, a min-cut based constraint oracle for introducing

constraints (CE) prior to any generated cut, and a simplex primal-dual algorithm for warm-solving the linear relaxations.

However, given the strict time restrictions for this final project, we took a more straightforward approach: we used a high performance commercial Integer Linear Programming (ILP) solver freely available for academic research: Gurobi 4. The solver acted as a grey box component in our general algorithm: we know how many branches and cuts the solver uses, can setup some heuristics levels and presolving options, or can even use the built-in callback mechanisms to interfere with the actual branch-and-cut process by adding our own cutting planes or use our own branching heuristic. However, the solver does not support lazy evaluation of the constraints, which is the usual approach for formulations with an exponential number of constraints. Changing the model by introducing those during the branching process is highly discouraged by the reference manual. Thus, we keep adding the island elimination constraints one at a time and re-optimizing, until a valid solution is found, or the problem is deemed infeasible by the solver. Of course, the global efficiency of this process is highly dependent of the warm-restart capabilities of the solver when adding a new constraint and re-optimize.

Algorithm 2 presents our approach. Except for the ILP solver which is out of the scope of this work, there is no particular algorithmic difficulty as we only deal with integral 0/1 values from the returned partial solution.

Algorithm 2 Violated constraint oracle based ILP for solving TMP

$A \leftarrow$ {(T) constraint} $\cup$ {(CC) constraints} $\cup$ {(XY) constraints} $\cup$ {(XZ) constraints}
$C \leftarrow \emptyset$
X = solve($A \cup C$)
while $X \neq \emptyset$ **do**
  if X is connected **then**
    return X
  else
    $C \leftarrow C \cup$ ViolatedCEConstraint(X)
    X = solve($A \cup C$)
  end if
end while
return Infeasible instance

C. On-the-fly cycle elimination constraints

The cycle elimination constraint oracle *ViolatedCEConstraint* is a cycle enumeration algorithm. We used the classic Johnson's cycle enumeration algorithm (1975).

The most sensitive part of the approach is certainly the choice of which cycle elimination constraint to add after each partial solution returned by the solver. As the partial solution can have more than one cycle, there is usually more than one possible cycle elimination constraint to add. The choice of which constraint to add may appear neutral, as we know that by adding any one of these, we effectively forbid the partial solution from occurring later in the process. However, this tabu-search like view of the algorithm is somewhat misleading. In fact, the added constraint should be as stringent as possible in order to be efficiently handled by the solver in an heuristic attempt to minimize the total running time of the algorithm. By choosing the highest cardinality cycle constraint in terms of x variables, we have by definition the most stringent constraints on the x variables. It appears to be in practice both a highly profitable and fast heuristic.

Consequently, our algorithm tends to behave as follow. It gradually constructs a relevant cycle elimination constraint "basis" by successively exploring feasible regions subject to the other non-CE constraints. Thus, in the early execution of the algorithm, the running time of the solver is generally low as the log shows it merely uses its internal heuristics and a logical presolve module, without need for branching. As the number of cycle elimination constraints grow, the algorithm moves toward more challenging regions where both the cycle elimination constraints and other constraints start to interact. Thus, most of the branch-and-cut process occurs during the final and most time consuming iterations of the algorithm.

D. Variations

There is a number of variations one can make to the ILP formulation 3.1. First, because of the binary nature of the variables, the constraints $x_i \geq y_{i,j}$ and $x_j \geq y_{i,j}$ can be replaced by $x_i + x_j \geq 2y_{i,j}$. However, doing so actually degrades the performances. Also, it is possible to relax the uniqueness of skill constraints (CC) by requiring at least one skill instead of exactly one skill for the query skills: $\forall c \in q, \sum_{i \in \mathrm{sk}^{-1}(c)} z_{c,i} \geq 1$. Additionally, one can also enforce each actor to posses exactly one skill instead of sharing multiple skills by modifying (XZ) to $\forall i \in C, \sum_{c \in \mathrm{sk}(i)} z_{c,i} = x_i$.

IV. Evaluation

A real empirical evaluation of the algorithm's performance would require the kind of data one can find at some well-known social networking companies. Apart for the difficulty of getting such data, there are also privacy concerns that forced us to consider another kind of evaluation. More specifically, we devised a restriction of the TMP problem which is still NP-equivalent and generated appropriate input for evaluation of the algorithm on the restricted TMP. We first introduce a restriction of the TMP, an inverted sudoku-like game, for which we omit the NP-equivalent proof in this report.

A. The Colorful Connected Set Problem

Let $\mathcal{A}$ be an alphabet of symbols. Let $(T_{i,j})_{1 \leq i,j \leq n}$ be a square matrix of symbols from $\mathcal{A}$.

Definition We say that two cells (i, j) and (i', j') are *neighbors* if and only if $(i = i'$ and $|j - j'| = 1)$ or $(j = j'$ and $|i - i'| = 1)$.

Thus, corner cells have two neighbors, non-corner bordering cells three, and interior cells four. Figure 2 illustrates our definition of neighbors.

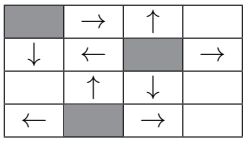

Fig. 2: Three cells and their neighbors in a 4×4 matrix.

Definition (Colorful Connected Set Problem) Given $\mathcal{A}$ and T, the complete connected set problem is to find a connected set of cells of T which contains all the symbols of $\mathcal{A}$ once and only once.

Figure 3 shows an instance of the CCSP with a valid solution.

F	U	H	O	Y	U	U	L	D	G
I	H	N	N	Z	J	X	K	K	Z
L	H	P	B	P	A	B	I	B	H
N	H	S	C	M	W	P	A	E	O
Z	Z	L	T	A	L	U	Q	P	R
K	J	J	N	O	M	B	A	F	R
J	Y	L	T	R	S	I	F	B	I
I	B	Y	N	R	A	A	E	H	S
Z	K	V	I	D	Y	E	M	X	D
P	P	B	H	L	B	K	F	S	N

Fig. 3: A 10×10 instance of a CCSP with 26 symbols. The shaded cells represent a valid solution – a connected set of size 26 which contains all the alphabet symbols.

Notice that CCSP is just a special case of TMP where one defines G as a flat 2D grid of vertices. Remarkably, this restriction to a 2D grid does not change the NP-equivalence of the problem and the core ideas of our previous reduction still apply.

B. Evaluation

We construct an evaluation set of various difficulty 10×10 CCSP instances from Edgar Allan Poe's corpus of tales by randomly selecting sentences and filling the grid with the successive alphabetic characters, ignoring punctuation, spaces and capitalization. There is 1,000 instances in this set. Figure 4 show the most important performance characteristics of our approach on this evaluation set. The evaluation was done using Gurobi 4 ILP solver on a laptop with a quad-core 2.53GHz Intel i5 processor.

There are two central questions that our algorithmic approach raises:

- How many cycle elimination constraints will have to be added before termination?
- How efficient are the solver and its warm-start mechanism - if any?

As can be seen in the evaluation, there is only a moderate number of added cycle elimination constraints, as expected. On the other side, the warm-restart capacities of the solver seem limited as shown in the last graph: the solving time is not linear in the number of added constraints, but rather quadratic: the sequence of running times of the ILP solver tends to be arithmetic, at least on this particular data set.

C. Recreational instances

We give here 4 instances of text-based Complete Connected Set Problem as recreational examples. 3 of them are feasible, the other one was proved to be infeasible.

V. Conclusion and further work

The approach followed in the resolution of the TMP is rather unconventional. Instead of using a regular LP solver and devising our own branch-and-bound algorithm, we directly used a high performance ILP solver which already implemented the state-of-the-art techniques in terms of cutting planes and heuristics. We managed to solve the majority of instances in our $1o \times 10$ evaluation set in under 7s, while the worst time was still under a minute.

Some questions are left open by this report and demand further investigation. First, because of the lack of sufficient data, an evaluation our algorithm was not possible for its primary goal: finding skilled and cohesive teams in social networks, let alone actually observing the social effects of such optimal groups. Secondly, the TMP has a natural counterpart in the field of computational biology, and more particularly in approximating molecular evolution distances via querying protein-protein interaction networks. It would be interesting to observe how well our approach performs when compared to Torque [1] todays' best know approach for this problem.

References

[1] Sharon Bruckner, Falk Hüffner, Richard M Karp, Ron Shamir, and Roded Sharan. Topology-free querying of protein interaction networks. *Journal of computational biology : a journal of computational molecular cell biology*, 17(3):237–52, March 2010.

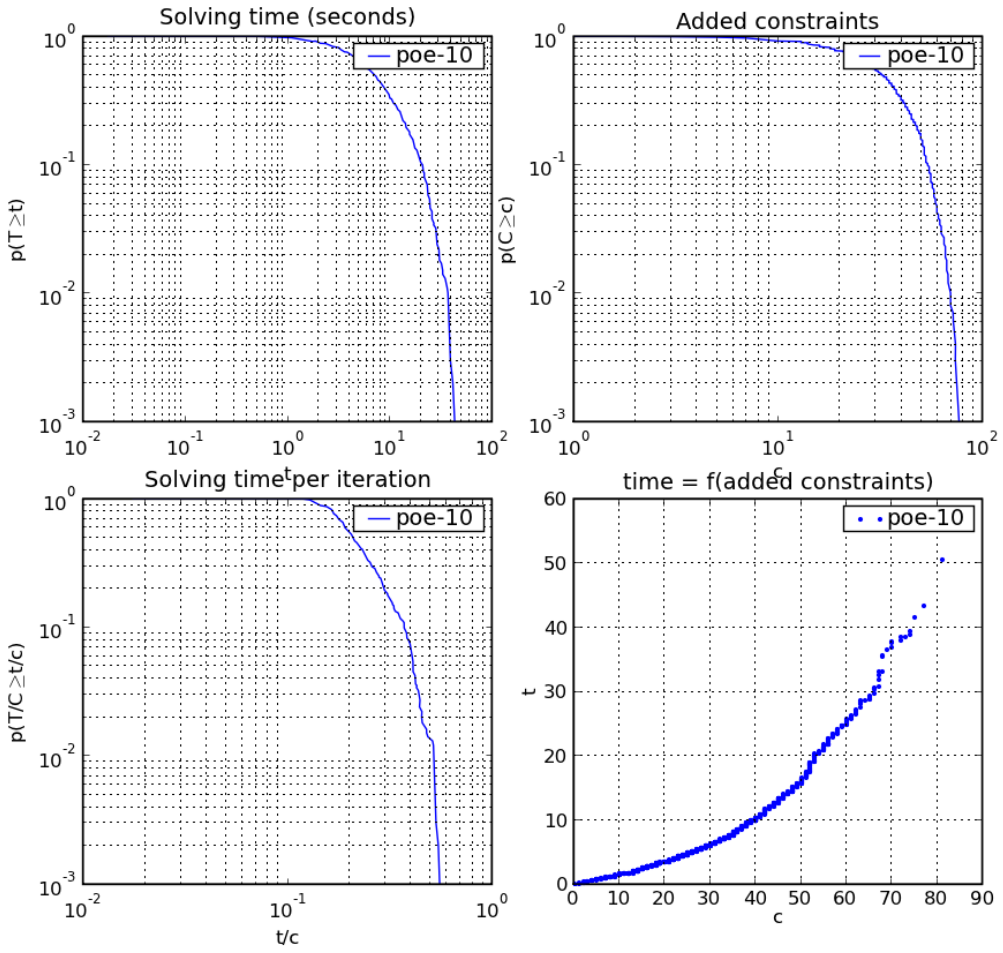

Fig. 4: The median solving time on the data set is 7s. The largest solving time observed was 42s. The median number of calls to the oracle is about 30.

t	h	i	s	m	a	n	f	o	l
l	o	w	s	m	e	e	v	e	r
y	w	h	e	r	e	w	i	t	h
n	o	t	h	i	n	g	b	u	t
h	i	s	g	o	a	t	s	k	i
n	p	a	r	c	h	m	e	n	t
a	n	d	w	r	i	t	e	s	i
n	c	e	s	s	a	n	t	l	y
b	u	t	i	o	n	c	e	c	a
u	g	h	t	a	g	l	i	m	p

(a) *This man follows me everywhere with nothing but his goatskin parchment and writes incessantly, but I once caught a glimp…*

p	e	o	p	l	e	o	f	a	l
l	k	i	n	d	s	a	r	e	s
t	r	e	a	m	i	n	g	i	n
t	o	t	h	e	c	i	t	y	f
o	r	t	h	e	f	e	a	s	t
d	a	y	a	m	o	n	g	t	h
e	m	t	h	e	r	e	a	r	e
m	a	g	i	c	i	a	n	s	a
s	t	r	o	l	o	g	e	r	s
s	e	e	r	s	a	n	d	m	u

(b) *People of all kinds are streaming into the city for the feast day. Among them, there are magicians, astrologers, seers and mu…*

h	e	s	t	a	r	e	d	d	u
l	l	y	a	t	t	h	e	p	r
i	s	o	n	e	r	f	o	r	a
w	h	i	l	e	t	r	y	i	n
g	p	a	i	n	f	u	l	l	y
t	o	r	e	c	a	l	l	w	h
y	t	h	i	s	m	a	n	w	i
t	h	t	h	e	b	r	u	i	s
e	d	f	a	c	e	w	a	s	s
t	a	n	d	i	n	g	i	n	f

(c) *He stared dully at the prisoner for a while, trying painfully to recall why this man with the bruised face was standing in f…*

a	c	h	a	i	r	h	a	d	a
l	r	e	a	d	y	b	e	e	n
p	l	a	c	e	d	o	n	t	h
e	m	o	s	a	i	c	f	l	o
o	r	b	y	t	h	e	f	o	u
n	t	a	i	n	w	i	t	h	o
u	t	a	g	l	a	n	c	e	r
o	u	n	d	t	h	e	p	r	o
c	u	r	a	t	o	r	s	a	t
i	n	i	t	a	n	d	s	t	r

(d) *A chair had already been placed on the mosaic floor by the fountain. Without a glance round, the procurator sat in it and str…*