What you should know about Adjumo

Chuan-Zheng Lee[*]

March 15, 2016

Abstract

At Thessaloniki WUDC 2016, we ran a new adjudicator allocation system, Adjumo. We built it to implement the adjudication core's policy of balancing adjudicator quality with regional, language status and gender representation. However, its core ideas were more fundamental: first, its objectives should be set by the adjudication core, not by tab programmers; second, that widely-used optimisation packages would improve the quality of adjudicator allocations.

This paper explains how the optimisation part of Adjumo works. It gives a primer on how optimisation solvers work and the scale of the problem at hand, and compares various optimisation methods. It then describes the way in which we modelled adjudicator allocation for the solver. The major component is a *score function* reflecting the desirability of a panel–debate pairing. Adjumo tries to maximise a weighted sum of logarithms of the score functions for each debate.

This paper then outlines the problems we encountered with adjudicator allocations in Thessaloniki and explains what caused them. Here, there were several instances of human error, for which we apologise. The optimisation algorithm itself performed as expected. Finally, noting that there is still much room for improvement, the paper suggests what work could be done to improve Adjumo for use at future tournaments.

Contents

[*]The author can be contacted at czlee@stanford.edu.

1 Contributors

The proper attribution of credit is important. Adjumo is about far more than just algorithms, and could not possibly have been undertaken by one person.

Philip Belesky wrote the user interface for Adjumo and did a world-class job. For those of you familiar with Tabbycat's drag-and-drop adjudicator allocation interface, Adjumo's interface is inspired by it, but—as is the nature of Adjumo—it is far more complex, and built from the ground up. Like Tabbycat, Adjumo highlights relevant conflicts[1] on rollover, and provides switches to highlight by region, gender and language status. It also highlights when adjudicators have already seen teams or other adjudicators ("history"), adjusting shade for how recent the encounter was. It shows detailed information about each adjudicator or team on rollover, updates score function components (see section 5.6) in real time, and provides mechanisms to group adjudicators together, and lock or ban them from particular debates. Our goal was to provide the adj-core with a tool through which they could make swift and informed manual adjustments to adjudicator allocations. All this work took hundreds of hours. It's difficult for

[1] Also known as clashes, strikes or scratches.

a text description to do it justice, so there are screenshots in Figures 1 and 2, and we've set up a demonstration site at http://czlee.github.io/adjumo/.

Chris Bisset did the hard work of transforming adjudication core policy into a quantitative scale, which I briefly discuss in section 5.6. Many experienced debaters have a rough intuition for goals in adjudicator allocation, but making these explicit enough for a computer is quite another undertaking. It took hundreds of hours through multiple iterations, thinking about trade-offs in adjudicator allocation likely more deeply than anyone has before.

I owe much gratitude to my friend Oliver Hinder, a PhD student in optimisation at Stanford University. Oliver is not an active member of the world debating community, but nonetheless enthusiastically volunteered his expertise towards the progress of our sport. His advice spanned model formulation and practical techniques to make Adjumo run in minutes rather than hours. Both were essential and are deeply appreciated.

2 Background

A goal of the Thessaloniki WUDC 2016 adjudication core (adj-core) was to account for adjudicator quality, and regional, language status and gender representation, as well as history and conflicts, when allocating adjudicators to panels. While some tab systems can display some relevant information to adj-cores, no existing system provides adj-cores with tools to consider all of these factors rigorously for as many debates as there are in a round at WUDC.

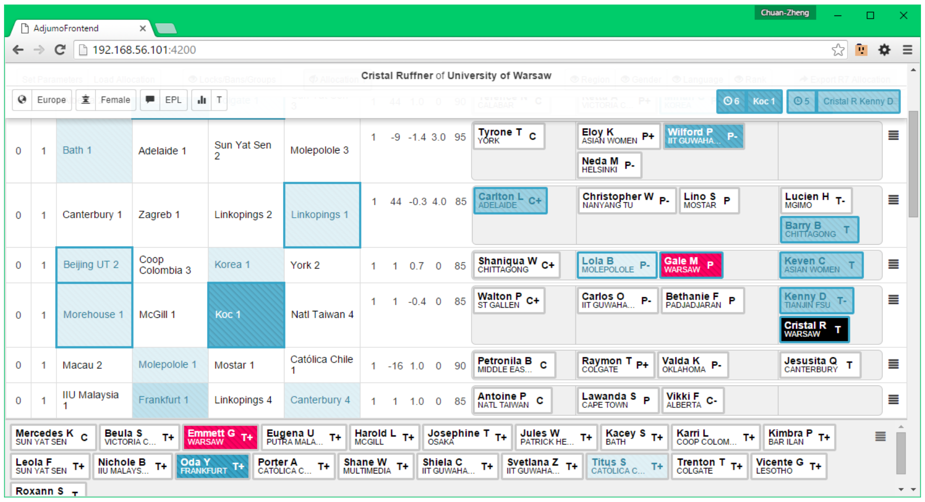
To help effect this part of the Thessaloniki adj-core's policy, we wrote a new adjudicator allocation system, called Adjumo (Allocating Debate Judges Using Mathematical Optimisation). This system comprises two components. One of them, the subject of this paper, uses optimisation software to recommend an allocation to the adj-core. The other is a new user interface that allows the adj-core to adjust allocations, displaying information about all of the aforementioned factors. Adjumo deals only with adjudicator allocation; the rest of the tab was done in Tabbie2.

Adjumo was used for the first time at Thessaloniki WUDC. In many ways, it functioned as intended, but in many ways, it did not. This document explains how Adjumo works and what difficulties we faced during the tournament.

3 Core principles

The idea of allocating adjudicators using optimisation is far from new. All major tab systems do this in some way. As we've said, unlike existing systems, Adjumo also tries to balance adjudicator quality, and regional, language status and gender representation. However, the core ideas behind Adjumo are more fundamental.

First, the objectives of the optimisation model should be set by the adj-core, not by tab programmers. Before Adjumo, adjudicator allocators were a



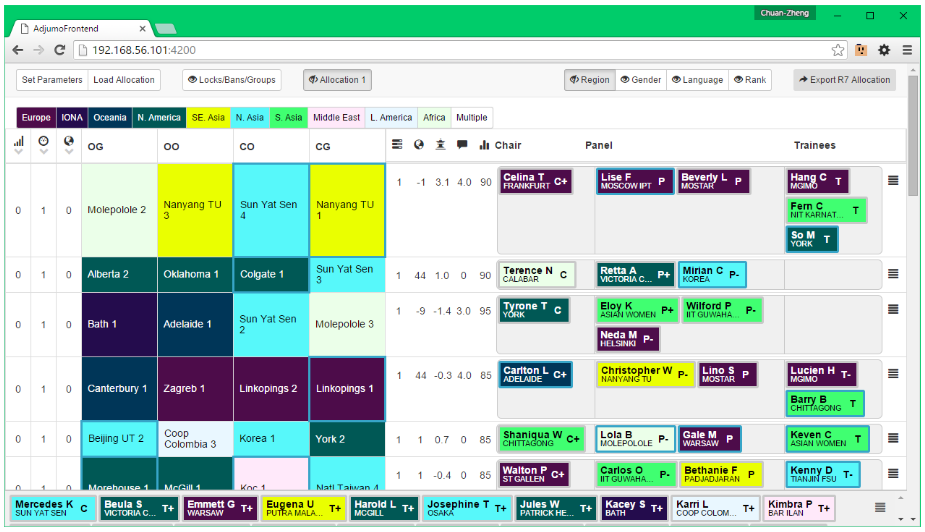
Cristal Ruffner of University of Warsaw

Europe | Female | EPL | T | 6 Koc 1 | 5 Cristal R Kenny D

0	1	Bath 1	Adelaide 1	Sun Yat Sen 2	Molepolole 3	1	-9	-1.4	3.0	95	Tyrone T YORK C	Eloy K ASIAN WOMEN P+; Wilford P IIT GUWAHA... P-; Neda M HELSINKI P-	
0	1	Canterbury 1	Zagreb 1	Linkopings 2	Linkopings 1	1	44	-0.3	4.0	85	Carlton L ADELAIDE C+	Christopher W NANYANG TU P-; Lino S MOSTAR P	Lucien H MGIMO T-; Barry B CHITTAGONG T
0	1	Beijing UT 2	Coop Colombia 3	Korea 1	York 2	1	1	0.7	0	85	Shaniqua W CHITTAGONG C+	Lola B MOLEPOLOLE P-; Gale M WARSAW P	Keven C ASIAN WOMEN T
0	1	Morehouse 1	McGill 1	Koc 1	Natl Taiwan 4	1	1	-0.4	0	85	Walton P ST GALLEN C+	Carlos O IIT GUWAHA... P-; Bethanie F PADJADJARAN P	Kenny D TIANJIN FSU T-; Cristal R WARSAW T
0	1	Macau 2	Molepolole 1	Mostar 1	Católica Chile 1	1	-16	1.0	0	90	Petronila B MIDDLE EAS... C	Raymon T COLGATE P+; Valda K OKLAHOMA P-	Jesusita Q CANTERBURY T
0	1	IIU Malaysia 1	Frankfurt 1	Linkopings 4	Canterbury 4	1	1	1.0	0	85	Antoine P NATL TAIWAN C	Lawanda S CAPE TOWN P; Vikki F ALBERTA C-	

Mercedes K SUN YAT SEN C | Beula S VICTORIA C... T+ | Emmett G WARSAW T+ | Eugena U PUTRA MALA... T+ | Harold L MCGILL T+ | Josephine T OSAKA T+ | Jules W PATRICK HE... T+ | Kacey S BATH T+ | Karri L COOP COLOM... T+ | Kimbra P BAR ILAN T+
Leola F SUN YAT SEN T+ | Nichole B IIU MALAYS... T+ | Oda Y FRANKFURT T+ | Porter A CATÓLICA C... T+ | Shane W MULTIMEDIA T+ | Shiela C IIT GUWAHA... T+ | Svetlana Z IIT GUWAHA... T+ | Titus S CATÓLICA C... T+ | Trenton T COLGATE T+ | Vicente G LESOTHO T+
Roxann S T

Figure 1: Screenshot of the Adjumo user interface, default view (with fake data)



Europe | IONA | Oceania | N. America | SE. Asia | N. Asia | S. Asia | Middle East | L. America | Africa | Multiple

			OG	OO	CO	CG						Chair	Panel	Trainees
0	1	0	Molepolole 2	Nanyang TU 3	Sun Yat Sen 4	Nanyang TU 1	1	-1	3.1	4.0	90	Celina T FRANKFURT C+	Lise F MOSCOW IPT P; Beverly L MOSTAR P	Hang C MGIMO T; Fern C NIT KARNAT... T; So M YORK T
0	1	0	Alberta 2	Oklahoma 1	Colgate 1	Sun Yat Sen 3	1	44	1.0	0	90	Terence N CALABAR C	Retta A VICTORIA C... P+; Mirian C KOREA P-	
0	1	0	Bath 1	Adelaide 1	Sun Yat Sen 2	Molepolole 3	1	-9	-1.4	3.0	95	Tyrone T YORK C	Eloy K ASIAN WOMEN P+; Wilford P IIT GUWAHA... P-; Neda M HELSINKI P-	
0	1	0	Canterbury 1	Zagreb 1	Linkopings 2	Linkopings 1	1	44	-0.3	4.0	85	Carlton L ADELAIDE C+	Christopher W NANYANG TU P-; Lino S MOSTAR P	Lucien H MGIMO T-; Barry B CHITTAGONG T
0	1	0	Beijing UT 2	Coop Colombia 3	Korea 1	York 2	1	1	0.7	0	85	Shaniqua W CHITTAGONG C+	Lola B MOLEPOLOLE P-; Gale M WARSAW P	Keven C ASIAN WOMEN T
0	1	0	Morehouse 1	McGill 1	Koc 1	Natl Taiwan 4	1	1	-0.4	0	85	Walton P ST GALLEN C+	Carlos O IIT GUWAHA... P-; Bethanie F PADJADJARAN P	Kenny D TIANJIN FSU T-

Mercedes K SUN YAT SEN C | Beula S VICTORIA C... T+ | Emmett G WARSAW T+ | Eugena U PUTRA MALA... T+ | Harold L MCGILL T+ | Josephine T OSAKA T+ | Jules W PATRICK HE... T+ | Kacey S BATH T+ | Karri L COOP COLOM... T+ | Kimbra P BAR ILAN T+

Figure 2: Screenshot of the Adjumo user interface, highlighting by region (with fake data)

"black box" to adj-cores: it gave a result, but there was little understanding of what it was aiming for, except among those who wrote the code. Little wonder, then, that adj-cores would spend a long time correcting for what it didn't do. Typically, you can play with a handful of parameters, but you can't specify what it *means* for a panel to be "high quality" or "regionally representative". With Adjumo, the adj-core set its goals: the model itself was a statement of adj-core policy.

Therefore, contrary to what appears to be believed by some, it is expressly not the goal of Adjumo to replace the role of adj-cores in allocating adjudicators. It is the exact opposite: adjudicator allocation software should be a tool that works *with* an adj-core to achieve what they want, not one that just gives an output for them to change.

Second, there exists commercial and open-source software that draws on decades of academic research in integer optimisation. We should take advantage of this to improve adjudicator allocation. For comparison, Tabbie2 uses simulated annealing, a method often seen as a "last resort" in optimisation circles. Tabbycat[2] uses the Hungarian algorithm, which can't work with panel representation. I discuss these in more detail in section 5.1.5.

4 Scope

Having understood the contributions and the core principles behind Adjumo, we're now in a position to outline the scope of this document. This document will cover the following:

- A description of how the optimisation part of Adjumo works.
- The causes of the major problems Adjumo faced during the tournament.
- Future steps for Adjumo.

This document will *not* cover any of the following:

- Adjumo's user interface. If you're interested in seeing this—and you should be—Philip Belesky has set up a demonstration site.
- The adjudicator feedback system used at Thessaloniki. Adjumo doesn't know anything about feedback; it just pulls adjudicator rankings directly from Tabbie2.
- A guide on how to use Adjumo.
- Any delays or other problems that were unrelated to Adjumo.
- A principled justification of why Adjumo took representation into account. As hinted in section 3, I'd argue that a tab director who engages in this would be overstepping the proper boundaries of her role.

[2]Tabbycat is used for two-team styles like Australs, and is mainly used in Asia and Oceania.

5 Technical outline

Perhaps flippantly, we described Adjumo as a "powerful algorithm". Actually, to an engineer, our contributions don't really have anything to do with algorithms. Algorithms for solving optimisation problems are well-known and encapsulated in the commercial solver that Adjumo uses.

Rather, our novelty was in how we *model* adjudicator allocation—that is, how we formalise the problem to give to the solver.

5.1 A primer on optimisation and solvers

5.1.1 The beginning

If you did any calculus at high school, you've probably seen a first glimpse of optimisation. You probably had to solve a problem like this:[3]

> A diver dives into a pool. The depth d metres that she reaches after t seconds is given by $d(t) = 1.25t^2 - 4t$. Find the greatest depth that she reaches.

And you probably learnt that to maximise $d(t)$ with respect to t, you should find its derivative, $d'(t)$, and solve for $d'(t) = 0$.

Any problem in which we wish to maximise or minimise a function with respect to its variables is called an **optimisation problem**. The above example is the simplest imaginable, and has the luxury of a simple method of solving it. In many real-world problems—including adjudicator allocation—we have many variables to choose, constraints on what solutions we are willing to accept, and there's no concept of a derivative. There are numerous classes of optimisation problems, ranging from the efficiently solvable to the practically impossible. Decades of research has gone into understanding how to solve them.

Drawing on this research, there are a number of software packages, some commercial and some open-source, for working with optimisation problems. The part of this software that solves a problem is called a **solver**. To use a solver, one defines a **model**, a mathematical description of the problem at hand, and passes it to the solver. The solver returns a **solution**, along with other useful information. Adjumo thus uses a solver: it transforms information about the draw, teams and adjudicators into a model, and transforms the returned solution into an allocation of adjudicators to insert into the draw.

5.1.2 How hard is it?

It's worth taking a moment to understand the scale of the problem at hand. Say there are 300 accredited (non-trainee) adjudicators and 100 debates. (At Thessaloniki WUDC 2016, there were about 350 and 96.) Then there are

[3] This was in the 2014 exam for NCEA Level 2 mathematics and statistics in New Zealand, which students sit in the penultimate year of high school.

$\frac{300!}{(3!)^{100}} \approx 4.7 \times 10^{536}$ possible allocations of adjudicators to debates. For comparison, there are about 10^{80} atoms in the observable universe. This isn't necessarily the best measure of complexity—solvers (and humans) have far better methods than exhaustive search—but it's still a mind-bogglingly large problem.

To understand complexity, computer scientists classify problems by the relationship between how long it takes to solve them, and their size—in our case, the size of the tournament. Some optimisation problems can be solved *in polynomial time*, which, informally speaking, means they can be solved reasonably quickly. Adjudicator allocation, a type of *integer program*, isn't one of them.

Instead, our problem is in a class known as *NP-hard*, which means it's at least as hard as the problems in another class, *NP-complete*. No-one has ever *proven* that this makes it *impossible* to solve "quickly"—this is one of the largest unsolved questions in computer science. But to put things in perspective: you'd have an easier time breaking modern computer encryption schemes, which rely on our inability to solve NP-complete problems without taking forever.

Not all hope is lost. Despite the theoretical intimidation, there are still decent approaches to tackling the problem in practice. Solvers use these, but bear in mind that even for a computer, this is an extremely difficult problem.

5.1.3 The gap

This is not the place for a detailed discussion of solvers, but one aspect will help. When given a problem, solvers work by finding both a *lower bound* and an *upper bound* to the optimal value. The goal of the solver is to push these bounds together. When they're close enough—typically within about 0.01%—it will call whatever it has a solution.

One benefit of using solvers is that they are able to characterise this *gap* between the bounds. If the gap is, say, 3%, that means that it is *at most* 3% suboptimal: we don't know what the best possible value is, but we know that it can't be 3% better than what we have now.

To make Adjumo run faster, we told the solver 1.2% would be good enough. While 0.01% is appropriate in some settings, for our purposes it's unnecessary.

5.1.4 How long does it take?

The solver we used, Gurobi, is one of the fastest commercial solvers on the market.[4] Nonetheless, as we learnt during development, if we *actually* tried to find the global optimum for a problem of this scale, we'd have to leave a solver running for far longer than we have between rounds in a debating tournament. I couldn't even guess how long—I know it's at least hours, but it wouldn't surprise me if it was months or years.

We therefore made trade-offs between optimality and running time. I describe the main strategy in section 5.8, but the salient point here is that "how

[4]We tried open-source solvers too; they didn't compare well.

long does it take?" isn't a very useful question. For a problem of this complexity, how long it takes is a design decision about how to navigate this trade-off, not an intrinsic property of the algorithm.

We decided to aim for about 10 minutes. We felt this was the best trade-off between allowing the solver to find a good solution, and allowing enough time for the adj-core to make manual adjustments. The time-saving tactic described in section 5.8 was tuned to this goal. We instructed the solver to stop after $12\frac{1}{2}$ minutes and return whatever it had, if it hadn't already reduced the gap to 1.2% by then. Sometimes it would reach 1.2% faster; sometimes it would take all $12\frac{1}{2}$ minutes and return a solution with a gap of around 10%.

5.1.5 Comparison to other solver methods

The method used to solve integer programs by most solvers, including Gurobi, is one of a type known as *branch and bound.*[5] This method repeatedly partitions the search space into smaller spaces, using analytical methods to skip partitions that can't possibly offer anything better than what's already been found. This allows it to keep track of the *gap* described in section 5.1.3 above. However, its running time is not very predictable, and it can be slow. Solvers, including Gurobi, typically apply many additional techniques to help it run faster.

The optimisation method used by Tabbie2 is simulated annealing. This method starts with an initial guess, makes a random change, accepts it if it's an improvement and *might* accept it even if it got worse, then repeats. It's not intended to return the global optimum, but it gives a decent intelligent guess, so it's commonly used where the problem won't work with a more specialised method (like branch and bound) and a heuristic is good enough. Typically, it is run for a specified number of iterations, so it can be run for exactly as long as one likes.[6] But it can't provide the gap or any indication of how confident it is in its solution.

Both of the above methods are essentially intelligent manners of trying various allocations and comparing them to previously-tried allocations. The difference is in how next attempts are chosen. Branch and bound explores the search space systematically. Simulated annealing uses random variations to generate possible new solutions.

Although the use of a solver package in adjudicator allocation is new and we believe it is a better approach, it's worth noting that in principle, the solver could be replaced with simulated annealing without affecting the model formulation or any other part of Adjumo.

The optimisation method used by Tabbycat is the Hungarian algorithm, also known as the Munkres algorithm. This method returns the optimal solu-

[5]It's not possible to do this comparison justice in reasonably short space, partly because Gurobi's solver is *much* more complicated than what I describe here. For readers interested in a deeper discussion, the internet has plenty of good resources on all of these techniques. For example, the Wikipedia articles on branch and bound, simulated annealing and the Hungarian algorithm, as well as a primer on mixed-integer programming on the Gurobi website, might be a good place to start.

[6]In the case of Tabbie2, you can choose this in the energy configuration.

tion in polynomial time, but is much more limiting in what models it accepts. Specifically, the problem must be formulated as a standard *assignment problem*. Tabbycat's model complies with this by considering each adjudicator position as an independent slot: it defines in advance how many adjudicators are on each debate's panel, and it doesn't account for any relationships between adjudicators on a panel. Any model seriously interested in the representation present on panels cannot be formulated in this way.

5.2 How it all fits together

When Adjumo is asked to generate an allocation, here's what happens:

1. Adjumo reads data from files exported from Tabbie2. This includes the draw, properties of all teams and adjudicators, as history and conflicts.

2. Adjumo formulates the model using this data. This involves computing the score function (section 5.6) for all panel–debate combinations, as well as the other inputs outlined in section 5.3.

3. Adjumo passes this model to a solver. The solver finds the optimal solution to the problem (or as optimal as it can get) and returns it to Adjumo.

4. Adjumo transforms the solution into an adjudicator allocation that can be presented to the adj-core for review and adjustment.

5.3 Framework for model

I'll describe the model in standard mathematical terms, since I suspect those with relevant backgrounds will be interested, but I'll also try to make it as accessible as I can.

Every round, we have a set of debates $\mathcal{D} = \{d_1, d_2, \dots\}$ and a set of adjudicators $\mathcal{A} = \{a_1, a_2, \dots\}$. (To avoid doubt, d_1 represents a debate, not a number.) We have a set of panels $\mathcal{P} = \{p_1, p_2, \dots\}$, which we call the **feasible panels set**. I'll come back to exactly how we construct this set in section 5.8, but you can think of each feasible panel as a set of adjudicators. For example, if $a_1, a_2, \dots$ are adjudicators, $p_1 = \{a_1, a_4, a_7\}$ might be a panel, as might $p_2 = \{a_2, a_3, a_4, a_5\}$. In formal terms (ignore this if it means nothing to you): $\mathcal{P} \subseteq 2^{\mathcal{A}}$, where $2^{\mathcal{A}}$ is the power set of $\mathcal{A}$.

We want to allocate exactly one panel $p \in \mathcal{P}$ to each debate $d \in \mathcal{D}$. To help us understand which should go where, we define a score function $\sigma(p, d)$. This function takes a panel p and a debate d, and returns a number representing how "good" that panel is for that debate ($\sigma : \mathcal{P} \times \mathcal{D} \to \mathbb{R}$). This function encapsulates all of the adj-core's objectives: quality, representation, history, conflicts. Because we were determined that it should reflect adj-core priorities, not those of a tab programmer, defining the score function was by far the deepest, most challenging task of developing Adjumo. I'll come back to this in section 5.6, but for now, it suffices to understand that the better panel p would be for debate d, the higher $\sigma(p, d)$ will be.

Not all debates are equally important, so we assign each debate d a weight $w_d > 0$. I'll explain this in more detail in section 5.7, but roughly speaking, live rooms have a high weight, dead rooms have a low weight.

Note that each adjudicator is on many feasible panels. All debates will be assigned a panel, but not all panels will be assigned a debate. Clearly, each adjudicator should be on no more than one panel that *is* assigned. To facilitate this, we define one more function $q(p, a)$, which we call the **panel membership function**:

$$q(p,a) = \begin{cases} 1, & \text{if adjudicator } a \text{ is on panel } p \\ 0, & \text{if not.} \end{cases}$$

Now we are ready to state the optimisation problem. Let $x(p, d)$ represent whether panel p is assigned to debate d, *i.e.*,

$$x(p,d) = \begin{cases} 1, & \text{if panel } p \text{ is assigned to debate } d \\ 0, & \text{if not.} \end{cases}$$

The job of the solver is to choose all of the values of $x(p, d)$—that is, choose whether $x(p, d)$ should be 1 or 0 for every panel–debate combination (p, d)—in order to do the following:

$$\begin{aligned} \underset{x:\mathcal{P}\times\mathcal{D}\to\{0,1\}}{\text{maximise}} \quad & f(x) = \sum_{d\in\mathcal{D}} \sum_{p\in\mathcal{P}} w_d \log\left[\sigma(p,d)\right] x(p,d) & \\ \text{subject to} \quad & \sum_{p\in\mathcal{P}} x(p,d) = 1 & \text{for each } d \in \mathcal{D} \\ & \sum_{d\in\mathcal{D}} \sum_{p\in\mathcal{P}} x(p,d) q(p,a) \le 1 & \text{for each } a \in \mathcal{A} \end{aligned}$$

A statement of the above form is called an optimisation problem. The first line is called the **objective**, which in our case is to maximise $f(x)$. We call $f(x)$ the **objective function**.[7] The small "$x : \mathcal{P} \times \mathcal{D} \to \{0, 1\}$" underneath the word "maximise" reminds us that we are choosing $x(p, d)$ for every panel–debate combination (p, d).

The lines following "subject to" are called **constraints**. An explanation of what each of these lines mean follows.

[7]Note that x is a function, so the objective function $f(x)$ "acts on a function". If this bothers you, you can also think of $x(p, d)$ as a matrix X with one row for each panel and one column for each debate, such that its (i, j)-th element is $X_{ij} = x(p_i, d_j)$. Then the solver is trying to choose every element of X to maximise $f(X)$. This is exactly equivalent. Indeed, because there are a finite number of debates, panels and adjudicators, $\sigma(p, d)$ and $q(p, a)$ can be recast similarly, and in fact are in the software implementation. I just found it easier conceptually to think of them as functions. If this doesn't help, just imagine $f(x)$ as a single symbol representing "the objective quantity", so to speak.

5.4 Objective

To understand this expression, recall that $x(p, d)$ is always 1 or 0, so the objective is the *weighted sum* of the *logarithms* of the *scores* of the *selected* panel–debate allocations. Another way to write the same objective, is

$$f(x) = \sum_{d \in \mathcal{D}} w_d \log \left[\sigma(p_x(d), d)\right]$$

where $p_x(d)$ is the panel allocated to debate d under the allocation represented by x. In other words, find the score achieved by the panel for each debate, and find the weighted sum of their logarithms.[8]

Why the logarithm? In short, it's to allow more precision in how we weight the importance of rooms. In broad terms, we want the optimisation to try to do two things:

1. Maximise the overall quality of the allocation.
2. Allocate panels proportionally to the weights of debates (so a debate with twice the weight will get "twice as good" a panel).

These two goals often conflict. The logarithm enforces a trade-off between them. To get some idea, here's an intuition pump: for two debates of equal weight, it will prefer to halve one debate's score if it can more-than-double another's. We could have chosen other trade-offs from a family of functions known as weighted α-fairness (ask me if you're interested); the logarithm is the case when $\alpha = 1$.[9]

5.5 Constraints

The first set of constraints says that every debate must have exactly one panel.

The second set of constraints uses the panel membership function to insist that every adjudicator must not be on more than one panel which is allocated. That is, every adjudicator must be allocated at most once.

Ideally, we would want every adjudicator to be allocated *exactly* once. But it turns out that the solver runs *much* faster if we don't insist on this. Intuitively, the solver has more freedom to walk through possible allocations this way. Because of how we designed the score function, the solver would still try to allocate as many adjudicators as it could. In practice, we found it was typically less than a dozen or two low-ranked panellists that needed to be allocated manually after the solver had completed.

[8] For the pedantic: Note that this is only equivalent because of the first constraint.

[9] In fact, any concave, strictly increasing function would have done fine, with caveats around how you construct weights.

Table 1: Components of the score function

symbol	**description**	**function of**
$\pi(p)$	panel quality	panel
$\alpha_\pi(p,d)$	regional representation	panel, debate
$\alpha_\gamma(p,d)$	language representation	panel, debate
$\alpha_\phi(p,d)$	gender representation	panel, debate
$\iota_o(p,d)$	team–adjudicator history	panel, debate
$\delta_o(p,d)$	team–adjudicator conflicts	panel, debate
$\iota_\delta(p)$	adjudicator–adjudicator history	panel
$\delta_\delta(p)$	adjudicator–adjudicator conflicts	panel

5.6 Score function

The idea behind the score function seems simple enough: define a function that gives higher scores for better panel–debate pairings and lower scores for worse ones. As we learnt over the course of developing Adjumo, this is more complex than it sounds. For this to work, the adj-core has to define *fully* the trade-offs we want the solver to apply.

The score function is the most important part of Adjumo and demands the most attention from anyone most interested in understanding how the adj-core's priorities were represented. However, its details are too complex to describe fully in this document. I'll give a high-level overview, and refer the curious reader to our Google Spreadsheet or `score.jl` in the source code for further details.

We modelled the score function as a weighted sum of "components". There are eight components, as listed in Table 1. Using the symbols in that table, we can then write the score function as (I omit the last four terms, but you get the idea)

$$\sigma(p,d) = v_\pi \pi(p) + v_{\alpha\pi}\alpha_\pi(p,d) + v_{\alpha\gamma}\alpha_\gamma(p,d) + v_{\alpha\phi}\alpha_\phi(p,d) + \cdots \qquad (1)$$

where v_π is the weight given to panel quality, and so on. Note that, since all scales are arbitrary, the raw weights $v_\pi, v_{\alpha\pi}, \ldots$ cannot alone be taken to reflect the relative importance of components: it is also their job to compensate for natural variations in the scales of components arising from the metrics used.

To give a rough idea of what each component seeks to achieve:

- **Panel quality.** Panels with highly-ranked adjudicators get high scores. Penalties apply if trainee-ranked adjudicators are on the panel. Highly-ranked adjudicators always improve panels, but they do so by a greater margin if the panel would not otherwise have a highly-ranked adjudicator.

- **Regional representation.** Penalties apply for team regions not represented on the panel. Smaller bonuses apply for adjudicator regions not represented in the debate. (There's a lot more to it than this—see the spreadsheet or source code.)

We used ten regions, grouping countries by debating circuit rather than geography. You can see the groupings in the source code in `importtabbie2.jl` (you'll have to decipher the ISO-3166 codes yourself, sorry).

- **Language representation.** Penalties apply for panels without an ESL or EFL adjudicator.[10] Bonuses apply for panels with multiple ESL or EFL adjudicators, but with diminishing returns. Greater weight is given to rooms with a mix of EPL, ESL and EFL teams in them.

- **Gender representation.** Penalties apply wherever non-males comprise less than half the panel.[11] Greater weight is given to rooms with a mix of all-male and not-all-male teams. Greater weight is given to all-non-male teams than to mixed teams.

- **History.** For each time an adjudicator has seen a team in the debate or another adjudicator on the panel, a penalty of $\frac{1}{c-h}$ is applied, where c is the current round and h is the round where they saw each other.

- **Conflicts.** A large penalty applies for each conflict. No distinction is made between personal and institutional conflicts.

To give an example to help demonstrate how this function works, the (imaginary) panel–debate pairing in round 6 of a tournament shown in Table 2, would be scored as summarised in Table 3.

It cannot be emphasised enough that readers interested in how we modelled adj-core objectives should peruse the Google Spreadsheet and source code for further details. It should also be said that there was a lot more nuance we would have liked to add to each of these components, but it will become apparent in that spreadsheet that even a simplistic model requires a lot of complexity to be tractable.

5.7 Debate weights

On day 1, all debates had equal weight. On days 2 and 3, we applied the following principles:

1. A debate is *maximally live* if, in a hypothetical scenario in which this round were the last preliminary round, this debate would affect whether or not any team in it breaks. Maximally live rooms have a weight of 5.

2. A debate is *live* if it may affect whether or not any team in it breaks (after nine rounds); that is, if there is a team that could still only just break by winning every round, or a team could only just not break by losing every round. A live room on y points has weight $5 - 0.2|y - m|$, where m is the number of points of the closest maximally live room.

[10] We made no distinction between ESL and EFL adjudicators.

[11] We made no distinction between genders other than male for the purposes of adjudicator allocation.

Table 2: Score function example: teams and adjudicators in debate

Teams			**Gend.**	**Lang.**	**Region**
Sheffield 2			m, m	EPL	IONA
Mostar 1			f, f	EFL	Europe
King's London 2			f, m	EPL	IONA
Cape Town 1			m, f	EPL	Africa
Adjudicators	**Rank**	**Institution**	**Gend.**	**Lang.**	**Region(s)**
Dion Cuthbert (c)	C−	Leiden	m	EPL	Europe
Josef Deming	P	Pretoria	m	EPL	Africa
Ethelyn Robichaud	P	Los Banos	f	ESL	SE Asia

Table 3: Score function example: calculation for debate in Table 2

Component	**Raw**	**Weighted**
Panel quality (weight 5)	50.000	250.000
20 for C−, 10 for each P		
10 for C− chair		
Regional representation (weight $\frac{1}{2}$)	−55.125	−27.562
IONA not represented: −18		
Majority from regions in debate: −45		
One external adjudicator: +7.875		
Language representation (weight 1)	0.000	0.000
One EFL team: class weight 3		
One non-EPL adjudicator: panel score 0		
raw score = class weight × panel score		
Gender representation (weight 5)	−0.958	−4.792
Teams 1 all-non-male, 2 mixed: class weight 5.75		
Panel $\frac{2}{3}$ male: panel score $-\frac{1}{6}$		
raw score = class weight × panel score		
Team–adjudicator history (weight 25)	−0.833	−20.833
Cuthbert saw Sheffield 2 in round 3: $-\frac{1}{6-3}$		
Robichaud saw Mostar 1 in round 4: $-\frac{1}{6-4}$		
Score		**196.813**

3. A debate is *above-bubble* if every team in it is guaranteed to break.[12] Above-bubble rooms have a weight of 3.

4. A debate is *dead* if every team in it is guaranteed not to break. Dead rooms have a weight of 1.

The principles sound straightforward enough, but they are surprisingly difficult to calculate,[13] and even more difficult to predict for the ESL and EFL breaks. Ideally, we would have liked to compute liveness automatically based on the current standings. Instead, we ended up using Thevesh Theva's Debatebreaker app (which provides best- and worst-case scenarios with no information about current results) and manually entering debate weights (see `debateweights.jl`).

For the ESL and EFL breaks, we used previous years' break cut-offs as a guide, and erred on the side of including more rooms as live rather than fewer. For each room, we calculated what the weights would be for each of the open, ESL and EFL categories (if applicable) and assigned the room the highest of the three.

5.8 Feasible panels set

The "feasible panels set" is the set of all panels that may be considered by the solver. To understand how we constructed it, it will help first to consider a naïve approach that would *not* have been practical.

A naïve approach would have been just to take all possible combinations of three or four adjudicators. With 350 non-trainee adjudicators at the tournament, there would have been $\binom{350}{3} + \binom{350}{4} \approx 630$ million such panels.[14] Furthermore, it was adj-core policy to permit highly-ranked trainees to be voting panellists if it would help sufficiently with representation, which would add about 100 million more panels.

We would filter out panels that were clearly infeasible: any panel with an adjudicator–adjudicator conflict would be excluded from the feasible panels set, as would any panel whose quality was below a threshold. But even this would still leave several hundred million possible panels. I don't know how long this would take, but what I can tell you was that when I tried simulating rounds with millions of possible panels through the solver, it wouldn't complete even after an hour or so. Even worse, moderately large cloud computing instances (computers with lots of power and memory provided by Amazon or Google specifically for heavy computations like this) would typically run out of memory first.

This served as our biggest eye-opener to the scale of the task of adjudicator allocation—a task that we typically expect a team of humans to do in about twenty minutes. Even for a computer, we would need to cut down the problem quite aggressively. One way to do this is to reduce the number of feasible panels under consideration.

[12]By "guaranteed" I mean save for exceptional circumstances, like medical emergency.

[13]I *still* don't know how one would write an efficient and reliable script to do this.

[14]The notation $\binom{n}{k}$ refers to the binomial coefficient.

We found we needed to work with about 12,000 feasible panels in order to get the solver to find a solution in $12\frac{1}{2}$ minutes. I omit the details of how these 12,000 panels would be generated—it was partly random but partly structured—and refer interested readers to `feasiblepanels.jl` in the source code. (There are several methods in the file; we used the one called `permutations`.)

This reduction of the feasible panels set obviously sacrifices optimality. Because we never simulated a full round, without feasible panel reduction, to completion (it would probably take months, if not years), I can't tell you by how much. The solver stills give us a bound on how suboptimal its solution might be, but only within the realms of the feasible panels set we give it.

At the same time, it's worth remembering that simulated annealing, the algorithm used in Tabbie2, has no way of characterising its level of suboptimality. Furthermore, the basic problem of scale—there there are endless combinations of adjudicators—is intrinsic to the problem of adjudicator allocation, and all algorithmic (or human!) methods must necessarily deal with or evade it somehow.

5.9 Trainee allocation

Trainee allocation was not part of the model given to the solver, and therefore is basically not part of the Adjumo algorithm. Trainees were allocated to chairs on a relatively random basis, checking only for conflicts. Low-ranked chairs were given trainees before high-ranked chairs.

There was a catch, though. Some of you may recall that there were some rooms at the venue that were too small to support large panels of adjudicators. We therefore included a provision to try to keep 20 rooms to panels of three, and 37 rooms to panels of four. This meant that many rooms would not get trainees, leaving them to be loaded on to other (already large) panels. Trainee distribution was therefore by design uneven. We regret that the constraints of the venue necessitated this measure.

5.10 Interface with Tabbie2

We defined two JSON file formats: one that Tabbie2 generates to send to Adjumo with information about the round, and one that Adjumo generates to send to Tabbie2 with the completed allocations. In each system we wrote manually-invocable functions to export and import these files.

6 Things that went wrong

As participants will be aware, things did not always go to plan. This section explains the major problems with adjudicator allocation that occurred at Thessaloniki WUDC 2016.

It is worth noting that we did not have a rigorous understanding of the causes or a quantitative understanding of the impact of some of these problems during the tournament. Our priority at the time was to the run the tournament as quickly as we could. A lot of the diagnoses in this section derive from analyses of backup files, that I conducted well after the tournament had concluded.

It will become apparent in this section that none of the problems were caused by the algorithm (which by abuse of terminology I'm now using to mean "the model and the solver"), nor did any of them arise from any core aspect of the system. We're under no illusions about the degree of consequence of these errors or the need to build in mechanisms to prevent them in the future, but any assessment of Adjumo's future potential should bear this in mind.

6.1 Round 1 allocations

The round 1 allocations weren't the ones we wanted to put up. There were two contributing factors.

The first cause was that adjudicator rankings hadn't been perfected before round 1. This, naturally, caused the algorithm to return allocations that were perfectly sensible according to the rankings it was given, but were counterintuitive to those familiar with individual adjudicators. The adj-core was well aware of this and proceeded to make the many consequential adjustments using the Adjumo user interface, as planned.

The second cause was that I attached the wrong file when sending the final allocations to Jakob for import into Tabbie2. Specifically, the file I sent contained the allocations *before* they had been adjusted by the adj-core using Adjumo's user interface. I regret the error.

The correct file was, of course, on hand and I then sent this to Jakob to import into Tabbie2. There is no reason we can think of for why this shouldn't work, but for whatever reason the second import failed, and already being several hours behind from earlier delays in the day, we weren't inclined to risk another attempt.

We were still able to, and did, adjust some allocations using the Tabbie2 interface. However, with the delays from earlier in the day, these were limited to moving adj-core and tab team members who intended not to judge that round (and whom we had forgotten to exclude from the initial export from Tabbie2), away from chair positions. After the tournament, I compared the pre-adjustment and post-adjustment files. There were differences between them in the allocations of 63 adjudicators in 44 debates.

To prevent future similar human error, I commented out (disabled) the code that generates the file straight after the solver is completed but before it had been adjusted in the user interface. Ironically, this precaution would bite us in round 4, when we had an unexpected issue with the user interface (described in section 6.2) and the obvious workaround—importing the pre-adjustment file—was no longer available to us.

6.2 Delays on day 2

Note: This is outside the general scope of this document, but is included because there is interest among debaters in understanding it.

Before round 4, we experienced delays with the user interface. The interface is quite heavy and typically takes a couple of minutes to load, but here, it was taking exceptionally long (more than a few minutes) and we believed it was failing to load. We found a workaround for round 4, which we'll come back to in section 6.3.

For rounds 5 and 6, the algorithm was still working fine and the adj-core believed its work was essential to adjudicator allocations, so we continued to use it. I uncommented (re-enabled) the code that generates a file suitable for import into Tabbie2. We ran the Adjumo algorithm and bypassed the Adjumo user interface, using Tabbie2's interface instead for adjudicator allocation adjustment. None of the delays on day 2 had anything to do with the algorithm.

We never found the root cause of the user interface issue. We had never seen it before in our earlier tests with similar-sized data sets, and we could not reproduce it with the same code base and same data files on another machine. In fact, after further investigation, we no longer believe that there was actually a bug in the software. The most likely explanation is that the data that needed to be downloaded to the client computer from the server had grown[15], and that while our internet connection was generally sufficiently reliable, it wasn't reliable enough to download this much data within the bounds of our patience.

We didn't encounter this issue on day 3. For unrelated reasons, we switched to running Adjumo locally (on my computer) rather than on a remote Amazon EC2 server.[16] We used the entire Adjumo system, including the user interface, and we never saw the issue again.

6.3 Trainee designation

The trainee designation of some adjudicators oscillated during the tournament; some remarked that they seemed to be "random". There were a few reasons why this might have been the case. Some of these were intentional, but there was also an error that caused highly-ranked trainees to be allocated incorrectly as panellists on day 1. A summary of the causes of trainee designation changes, and the number of adjudicators affected by each, is given in Table 4.

[15]The data increases each round because it includes the history between every adjudicator, and every team and other adjudicator, so that they can be highlighted in the interface.

[16]During development, the solver was taking far too long to complete. The obvious first strategy is to try computers with more processor cores and memory, which are readily available in the cloud, on Amazon EC2. This helped a little, but not enough, so we started trying a variety of more aggressive strategies, eventually settling the strategy described in section 5.8. Believing that computing resources were still a bottleneck, we continued to run it on a powerful EC2 instance. It never occurred to me until late in the night after day 2 to try a full-sized data set on a notebook computer again with the new speed-up strategy. Needless to say, with the benefit of hindsight, we wish we had run Adjumo locally for the whole tournament.

[17]Because some adjudicators were affected by multiple categories, the different causes add up to more than 100 per cent.

Table 4: Summary of causes for trainee designation changes

Cause	**Adjudicators affected**
Off-by-one error inflating rankings on day 1	78
Trainee-plus allocated as panellist to aid representation	42
Promoted from trainee rank to panellist rank	48
Demoted from panellist rank to trainee rank	6
Total[17]	**121**

By design, there are two reasons why an adjudicator might move into or out of a trainee position. The first, obviously, is that he might be promoted or demoted. This was the case with 48 and six adjudicators respectively.

The second is that there was a distinction between trainee *ranking* and trainee *position* on a panel. Adjumo would sometimes put adjudicators of trainee-plus rank on panels where it helped sufficiently with representation, but this does not mean that said adjudicators' rankings were promoted. This trade-off between panel quality and representation was part of adj-core policy. This happened to 42 adjudicators, six of whom more than once.

A third cause of trainee designation change was an off-by-one error that caused all rankings to be imported into Adjumo one rank higher than they actually were. (There is a small conversion to do: Tabbie2 uses a 100-point scale; Adjumo implemented the adj-core's system, which uses nine categories.) It's not clear whether this error was the result of miscommunication or a coding error. We fixed this for round 4, but it would have meant that in rounds 1 to 3, some adjudicators who were actually of trainee-plus rank would have appeared to Adjumo as of panellist-minus rank, and therefore been allocated to panellist positions.

Although this theoretically affected round 4, in practice the impact was muted. The workaround to the user interface issue described in section 6.2 was to display information from the adjudicators file from round 1, which had the old off-by-one rankings, instead of the file from round 4.[18] This did *not* affect the initial allocation recommended by the algorithm, which used the correct updated rankings. It only affected the rankings *displayed* to the adj-core while they were making adjustments. Still, we then manually moved almost everyone who (erroneously) appeared to be of panellist rank into a panel. This then masked the most observable effect of the off-by-one fix. Owing to the delays, adjustments for this round were mostly limited to this exercise. We regret the use of a file whose rankings were inaccurate. Our priority at the time was to

[18]Rumours that day 2 rankings reverted to initial rankings probably refer to this. While they have plausible origin, they are false. In the circumstances, the only real effect was that those trainee-plus adjudicators affected by the off-by-one error were manually moved back into panellist positions. Trainees who earned promotions were still on panels, as the algorithm used the correct round 4 rankings, and the adj-core did not remove those adjudicators from panels.

minimise further delays.

Instead, the off-by-one fix became apparent in round 5. Altogether, the off-by-one error affected 78 adjudicators. Of those, 67 were panellists in round 4 and trainees in round 5. We apologise for the stress caused to all of these adjudicators.

One final glitch was that, in one round, a chair had the "(t)" designator by her name. The trainee paradigm used by Adjumo is different to, and therefore not fully supported by, Tabbie2. Specifically, Tabbie2 didn't support trainee *positions* on panels, so we added a patch to get it to display "(t)" next to the names of adjudicators that Adjumo marked as having trainee positions. This worked fine for most rounds, but as described in section 6.2, we reverted to Tabbie2's interface for the manual adjustments in rounds 5 and 6. Where the algorithm allocated an adjudicator to a trainee position, but the adj-core moved that adjudicator to a chair position, the trainee label persisted. The algorithm's placing a chair-quality judge as a trainee was caused by an inaccuracy in the associated ranking in Tabbie2; we regret the error.

6.4 Trainee distribution

We fielded some questions about why trainees were so unevenly distributed between rooms. Constraints with room sizes necessitated this; I explain this in section 5.9 above.

6.5 Fire!

Rumours that there was at any point a small fire in the tab room are patently false.

7 Future work

A project like Adjumo is never complete. We had plenty more ideas for refinements, features and tests. Moreover, many questions remain open. Here are a few:

- **Debate weights.** We put a lot of thought into how to weight debates (section 5.7), but we couldn't come up with a coherent, rigorous basis for them.

- **Tracking teams over time.** Not all teams will get good or representative panels in every round. We can hope that they'll "average out", but there's no guarantee—it might be better to nudge the model towards prioritising certain factors for teams that have been short-done in previous rounds.

- **Presentation of information in the user interface.** With all the factors and attributes to take into account, there is a lot of information that needs to be displayed. We put many hours of thought into how to present this in an easy-to-consume manner. Still, it all ended up being far too much to digest in the time an adj-core can reasonably spend on adjudicator allocations. How to summarise all this in a manner that adj-cores can use effectively and quickly—or indeed, whether it's even possible to do so—remains an open question.

It goes without saying that future work would also involve mechanisms to protect against the types of errors described in section 6.

There is also much room for potential improvement in our understanding of how to work with the optimisation model. Here are a few potential avenues for further investigation:

- Whether there are any alternative formulations to what was described in section 5.3, in particular with how scores are aggregated into a single objective function and how "weighted fairness" between debates is enforced
- Developing a better way to succinctly describe an intuition what a "score function" means, to better guide adj-cores in how to devise one
- How to normalise the different components of the score function (section 5.6), so that we can give debate weights a proper meaning (e.g., components of equal weight should have a "similar" impact on the outcome)
- Characterising more precisely the trade-off between optimality and speed, and whether there are any ways of navigating this trade-off other than the approach described in section 5.8
- Whether there are good ways to take advantage of parallel computing in formulating the model or computing the score function (we tried, but couldn't get it to help that much)

An important final note: If a principle of Adjumo is that its priorities should be set by adj-cores, not by some tab programmer of yesteryear, then the model of adjudicator allocation used by Adjumo should not be simply used as-is by future tournaments. Every adj-core should think about what principles they wish to use to allocate adjudicators, and write a score function and debate weights to reflect them. (These will then need to be implemented in the Adjumo source code.) If their goals align with those of Thessaloniki's adj-core, then they can and should use Thessaloniki's model as a starting point, while bearing in mind that that model is far from perfect.

We would welcome anyone interested in contributing to the project. The optimisation part of the code is written in Julia, a technical computing language that should be easy to pick up for anyone familiar with Matlab or Python. The user interface is written in JavaScript using the Ember framework. We won't insist on any requisite background from contributors (I learnt Julia while doing the project), so we encourage anyone interested to get in touch with us. The two parts are basically separate, so contributors need not be involved with both components.