One year of LCOGT v1.0
Lessons learnt the hard way

Eric Saunders
Software Team Lead
Las Cumbres Observatory Global Telescope
Main takeaways

1. What do you want?
2. It is possible to solve very large scheduling problems quickly, with the right formalism
3. It is not sensible to optimise everything
4. Uncertainty, fast recomputes, cadences etc. are hard problems
5. Do simulations!
6. How people perceive your system is important
7. The output of a clever scheduler will be beyond your understanding - good metrics are crucial
LCOGT Network

- Texas
- Hawaii (2015)
- Tenerife (2015)
- China/Tibet (2016)
- Chile
- South Africa (2015)
- Australia
The (astronomy) problem we solve

- Accept requests from many users, at any time, from anywhere with an internet connection
- Support many different kinds of science
- Be highly responsive to new input
- Utilise the full potential of a telescope network
- Be a killer follow-up observing engine
Network design philosophy (1)

- A single global observing abstraction
- Identical instrumentation
- Redundant resources
  - Cost-effective
  - Mitigates technical failures at site
  - Allows concurrent observing
Network design philosophy (2)

- Spectra and imaging at every site
- Fully robotic operation
- Tolerant to network outages
- Layered software intelligence
- Globally scheduled
LCOGT v1.0 capabilities

- Globally optimised observation placement
- Simultaneous and cross-site observing
- Automatic weather rescheduling
- Automatic hard constraint enforcement
- Automatic rescheduling of unsuccessful observations
- Target of opportunity observations, placed on-sky within 15 minutes of submission
- Human and machine request interfaces
- Cadence-driven, multi-site sequences
- Support for solar-system objects
- Low-dispersion spectrographs at both 2m sites
- Pseudo-real-time interface for education users
Common beliefs which are wrong

- an easy-to-state problem has an easy solution
- local optimisation implies global optimisation
- solving large NP problems is impossible (airlines do it by magic)
- being fast enough to recompute an entire schedule in real-time is impossible
- science “requirements” (e.g. cadence goals, seeing) are clear, hard lines
- if you could just make your function a tiny bit cleverer, it would make people happy
1) What do you want?

- Be **very clear** about goals
- Not everything cool is feasible
- Desirable goals often conflict, sometimes irreconcilably
- *Telescope Scheduling (1)*: the art of deciding who loses
2) Can you formalise?

- Astronomical scheduling - it’s not very good
- Telescope scheduling has no direct analog in CS or OR, but they are still way ahead of us
- Formalise your problem, and you can leverage existing work
- Clearly distinguish between astronomy, abstraction, and implementation
What is the telescope network scheduling problem?

Five key phrases for literature searches:

- **offline** (problem has fully specified input)
- **interval scheduling** (non-overlapping discrete time windows)
- **slack** (flexible start/end times)
- **multi-resource problem** (concurrent, not interchangeable)
- **scheduling-time dependencies** (logical connectors)
Many flavours of constraints and dependencies

- *a-priori* constraint: e.g. visibility, airmass, moon phase
- *real-time* constraint: e.g. seeing
- *scheduling-time* dependency
  - one request depends on another
  - one time depends on another (cadences)
- *post-completion* dependency
  - subsequent scheduling depends on post-completion function (*reactive scheduling*)
How do we solve the telescope network scheduling problem?

- Reduce the world state (actionable requests + environment + existing request states)
- Convert and discretise world state into a large integer programming model
- Solve the integer programming problem
- Convert back from abstraction
- Distribute the schedule
- Monitor and repeat to continuously solve the (actual) online problem
- As fast as possible (of course)
3) It is not sensible to optimise everything

- multi-objectives are *heuristic* by nature
- therefore messy, arbitrary and unsatisfying
- trade off one thing for another
- tend to “smear” outcomes
- simple objectives are rarely satisfactory
- complex functions tend towards making everyone equally *unhappy*

- if you have to do this
  - keep it as simple as you can
  - exhaustive simulation is essential
4) Hard problems: uncertainty

- *Telescope Scheduling (2)*: uncertain temporal decision-making
- Dealing with uncertainty is hard
- Dealing with correlated uncertainty is very hard
  - e.g. statistical variations in weather
  - e.g. weird user submission behaviour
- Evaluating behaviour post-run with perfect hindsight is tricky
4) Hard problems: being fast

− Large discrete optimization problems are usually solved in standard ways
  ○ linear/integer/mixed integer programming
  ○ constraint programming
  ○ a hybrid approach (e.g. branch and bound + constraints)
  ○ please - don’t try and reinvent this yourself

− Solving != exhaustively prove
− Solving == good enough
− Solving usually takes us less than 5 minutes
4) Hard problems: cadences

"Observe 75% of this variable star's period" (exact, abstract by phase)
"Equally space these 10 observations somewhere good" (exact, abstract by space)
"Complete all 10 observations of this SN, or give up" (exact, all)
"Make a best effort to obtain 10 obs, but less is better than nothing" (exact, most)
"Make at least 5/10 observations, or give up" (exact, \( n \) of \( N \))
"Make 4 observations, a day apart, somewhere good" (approx, abstract)
"Make 4 observations, a day apart, each +/- 6 hours" (approx, jitter)
"Make 10 observations, each between 3 and 6 hours apart" (approx, min/max)
"Complete all 10 observations of this SN, or give up" (approx, all)
"Make a best effort to obtain 10 obs, but less is better than nothing" (approx, most)
"Make at least 5/10 observations, or give up" (approx, \( n \) of \( N \))
"If an observation failed, observe again as soon as possible" (approx, continue on fail)
“Observe this target 20 times, with logarithmic spacing” (non-linear cadence)
5) Do simulations!

- Simulation framework is time well-spent
- Keep it modular (you will need to revise it)
- Don’t over-simulate things that don’t matter (think spherical cows)
- Simulation is only as good as its input and assumptions
- Evaluation of schedule quality (simulated or real) is hard
6) Human perception is important

- **Justifying** complex robotic scheduling to users/stakeholders is hard
  - top question: “why didn’t observation X happen”?
- People will attempt to **twiddle** even the most carefully nuanced scheduler, by hand
- Humans need to build a **mental model** of your process before they will trust you
- **Stability** in the continuum of schedules is highly desirable
7) Good metrics are crucial

- Metrics are the windows on your system
- What’s going on, and is it any good?
- **Basic**: How many/how much, observing efficiency,
- **Intermediate**: Contention, programme completeness, any time series
- **Combinations**: normalised efficiency, any x vs y
- **Meta**: Inter-schedule stability, d/dt (schedule runtimes)
Remember this

1. What do you want?
2. It is possible to solve very large scheduling problems quickly, with the right formalism
3. It is not sensible to optimise everything
4. Uncertainty, fast recomputes, cadences etc. are hard problems
5. Do simulations!
6. How people perceive your system is important
7. The output of a clever scheduler will be beyond your understanding - good metrics are crucial
Further reading / gory details

- An integer linear programming solution to the telescope network scheduling problem, Lampoudi et al., ICORES 2015
- Novel scheduling approaches in the era of multi-telescope networks, Saunders et al., SPIE 9149, 91490E
- Telescope network scheduling: rationale and formalisms, ICORES 2013
Questions?
Identifying the formal problem

- Decomposed the problem
- Expressed the problem formally
- Checked it hadn’t been solved
- Published the formalism (the *telescope network scheduling problem*), for others to attack
- To find a practical solution, devised various approximate transformations into problems that can be solved with known techniques
Astronomy by Intersection

Moon distance or other *a priori* constraints would be applied in the same way
Random grab-bag

- retrying on failures has multiple subtleties
- state gets complicated quickly
  - hard to understand
  - expensive to maintain and fix
- automated tests (unit, integration) are essential if you want to improve safely
- build modularity everywhere: you won’t get this right
- ToOs are a pain, and need special consideration