Global Optimization of Gear-Ratios for Motorsport Applications, or How to Make a Fast Car Faster

Thomas W. Barr

May 7, 2008

Abstract

This paper describes a numerical gear-ratio optimizer for motorsport applications. Using a simplified model of a Formula One car, performance on straight-line segments is simulated for each straight of a measured track, using both low and high fuel models. The gear ratios of the car in this model are numerically optimized to find a minimum average lap time, considering all the straight segments of the course. The system was tested using torque, drag and braking curve data estimated for a Ferrari FW2004 3.0L V10 Formula One car over the Silverstone racing circuit.

Results show that global optimization is vastly more effective than a basic evenly spaced ratio setup, saving 0.24 seconds per lap. Optimizing for both high and low fuel loads improves overall performance a further 0.033 seconds. Including shiftpoints in the optimization yielded no improvement, as the optimal shift-point is always at redline for any sensible gear ratio set.

Formula One, like motorsport in general, has become extremely popular in recent years. Additionally, it has become extremely expensive, with top teams spending up to $500 million per year to develop and run their cars. Such money-is-no-object budgets go into exotic material design, countless testing in exotic locales, wind tunnels, and, increasingly, supercomputing. The BMW Sauber F1 team has lead the way in the use of supercomputing in Formula One with their *Albert* series of computer. The latest, *Albert 2*, has over a thousand cores, and is one of the fastest industry computers in the world. As supercomputing continues to become more easily available, F1 will become a major player in scientific computing, just as it has in the fields of aerodynamics and composite structures.

For this project, I wanted to develop a model of an F1 car, and optimize some parameter of it. While the aerodynamics of the car are highly complex, and therefore outside of the scope of a semester project, the gearbox is a critical part of the design of the car, and optimization is a tractable problem in the time available. The current F1 season includes both fast tracks, such as Monza, slow and twisty tracks, such as Monaco, and tracks that

are a combination of the two, such as Silverstone. The ratios of the gearbox must be set precisely to allow maximum performance on these tracks. Therefore, I developed a system to optimize gear ratios for an entire track, namely Silverstone.

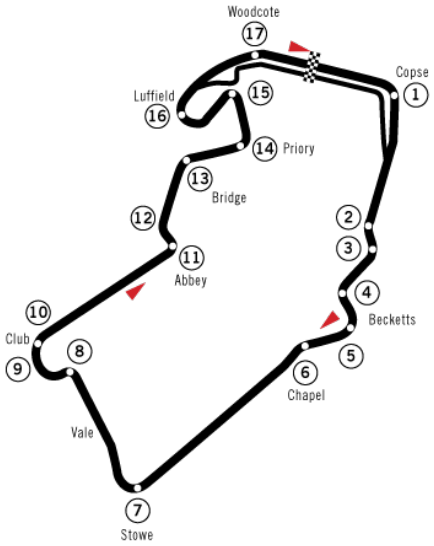

Figure 1: Silverstone track map.

1 Physics and model development

The amount of torque that any motor, especially a high-strung race version, generates is related to the speed that the engine is turning at. Maximum torque, and therefore acceleration, comes at a relatively high engine speed, and in a relatively narrow band. (It is often joked that F1 engines have "torque spikes" instead of torque curves.) I have estimated the torque curve of a 3.0L V10 from 2004 using a spline. The peak torque comes just before its 18,000 rev/min redline.

If a team were to simply attach the wheels to the engine, the acceleration force would be easy to find:

$$force(v) = k_1 \cdot torque(v/k_2) \tag{1}$$

k_1 here represents the maximum force that the engine can exert on the car, and k_2 is a combination of all the ratios, tire sizes and unit conversions between the engine and the

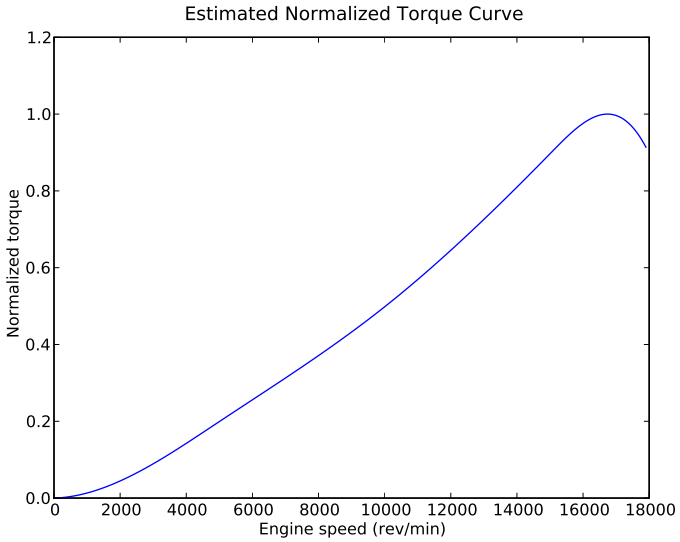

Figure 2: Estimate of a 3.0L V10 torque curve.

rear wheels. In other words, k_2 is defined such that $v/k_2 = rpm$. Clearly, this design is problematic, since the car needs to have decent performance at all speeds. Since the torque curve is not flat, this is impossible. At low speeds, the car will accelerate slowly because the engine is turning slowly and can't generate torque. What we would like to do is to be able to vary k_2 such that we can have the engine turning in the "power band" at all speeds.

This is done with a gearbox, a set of changeable gears that connect the engine to the rear wheels. This lets us set up a discrete set of k_2 values, optimal for different speeds. This would seemingly allow equation 1 to become

$$force(v) = k_1 \cdot torque(v/ratio[n]) \tag{2}$$

where n is the currently selected gear, and $ratio[n]$ is the value of k_2 for that particular gear. Unfortunately, remembering our grade school simple machines reminds us that things are not this simple. Any gearing system that increases speed necessarily decreases torque proportionally. Including this in our equation of force yields

$$force(v) = k \cdot \frac{1}{ratio[n]} \cdot torque(v/ratio[n]). \tag{3}$$

k is now a constant for all gears, and this equation is therefore usable in a simulator.

1.1 Drag and braking

While the engine was the main focus of my investigation, clearly this is not the only force on the car. At 325 km/h, a car as aerodynamically slippery as a brick generates tremendous drag, and the car must eventually be able to stop. I have assumed the drag on the car to be proportional to v^3 ($d(v) = k_{drag} \cdot v^3$), and that the car will be able to decelerate at a constant 4g. (This value was derived from observing the g-meter shown during telecasts of F1 races.) Both of these are gross simplifications, but they could be easily replaced with correct data if it were available.

1.2 Fuel load

Intially, fuel load only affected the mass term in the differential equation of motion. However, this led to incomplete results, so the model was modified. While I assumed that gear ratios would not affect corner speeds, and as a result could be eliminated from my model, corner speeds *are* affected by mass. With increased mass, the centripetal forces required are higher, but the available force for turning is not increased (as the majority of the normal force on the tyre comes from aerodynamic loads, which are invariant under mass changes).

The modified velocity through a corner is found by setting the forces in the equation for centripetal force to be equal for varying mass and velocity, and solving for the second velocity. Modified velocity is given as

$$v_2 = \sqrt{\frac{m_1}{m_2} v_1^2}. \tag{4}$$

1.3 Estimation of constants

Up to this point, the model for the car has been generic for any car. However, to get realistic results, I wanted to model a particular car as closely as possible. The 2004 Ferrari FW2004 3.0L V10 Formula One car is generally considered to be the fastest of all time, as regulations entering into force after that date reduced speeds significantly.

At Silverstone, the track used in this investigation, the car had a top speed of 325 km/h. It also made around 1000 horsepower with a redline of 18,000 rev/min. I have reduced this power to 800 horsepower to account for drivetrain losses, which are around 10% in a road car, and are likely higher in a race transmission. Assuming that the car is producing maximum power at top speed, we can estimate k in equation 3 as well as k_{drag}.

Dry mass of the car is taken as the minimum legal mass, 605 kilograms. Maximum mass is calculated as 8 "seconds" of fuel (fuel is delivered into the car by a pressurized, standardized fuel rig), at 12.5 litres/sec, yielding approximately 70kg of fuel.

1.4 Limitation of problem domain

This model estimates the performance of the car under full-throttle acceleration, however it does not model the car for turning performance. Since corners are generally taken without throttle to maximize turning force, the gear ratios are irrelevant in these sections of the track. Therefore, I have divided the straights into independent sections with starting and ending speeds equal to the speeds in the corners at the start and end of the segment, respectively.

1.5 Track map

The model of Silverstone used for this project was gathered from various Internet sources. Corner speeds were taken from a map of the track[1], and straight segment distances were measured using Google Earth[2].

This data is an estimate to serve as a proof of concept. Again, a real team would have easy access to much more accurate data.

2 Implementation

The simulator and optimizer were implemented in Python 2.5 using the SciPy and NumPy libraries. For each car setup to be tested, the simulator begins by simulating performance down a single, very long straight from a slow speed. This uses the differential equation

$$\frac{dv}{dt} = \frac{force(v) - drag(v)}{m} = \frac{1}{m}(k \cdot \frac{1}{ratio[gear(v)]} \cdot torque(v/ratio[gear(v)]) - k_{drag} \cdot v^3). \tag{5}$$

In this equation, $gear(v)$ is a Python function that determines the proper gear for the current speed. It does this by stepping through the ratios, and finding the first gear that will

[1] http://atlasf1.autosport.com/2000/bri/preview/brilap.html
[2] http://earth.google.com/

allow the engine to run below redline. Since $gear(v)$ is recomputed for every time step, we make the implicit assumption that gear changes take no time, and that drivers perform them perfectly. This is actually a relatively valid assumption since the gear change in a modern F1 car takes only a few hundred *micro*seconds, and the reaction time of the modern F1 driver is approximately the same. The result of that simulation, calculated using a simple Euler method[3], is used for the whole-track simulation.

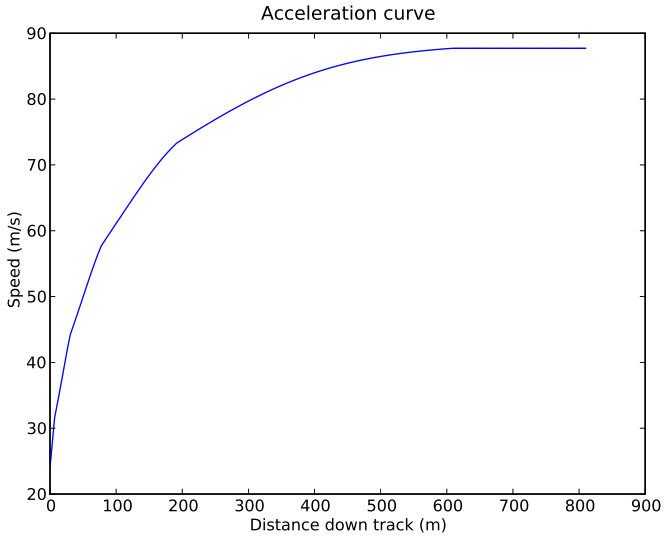

Figure 3: Acceleration curve from simulator. Shift points are visible as discontinuities in the first-derivative.

To save computation time, the acceleration curve (figure 3) is computed once, and then the dataset (containing points for time, speed and distance) is modified to represent different starting speeds and distances for each straight along the track. This is done by finding the time where the car is at the desired starting velocity, cropping the entire dataset at that time point, and then normalizing both distance and time to be zero the new initial value. Finally, we crop the dataset when the new normalized distance equals the length of the straight (see figure 4). This has the same effect as recalculating the DE from a new starting velocity without actually having to do so, and as a result is much faster.

Once the acceleration curve for a particular straight is calculated, a braking curve is determined using a similar method. This curve starts at high-speed, then brakes down to the entry speed for a particular corner. These curves are overlaid in the speed/distance

[3]Since the optimizer is slow, a simple ODE solver is used. In a production environment, more computing power would be available, and a more accurate solver would be practical.

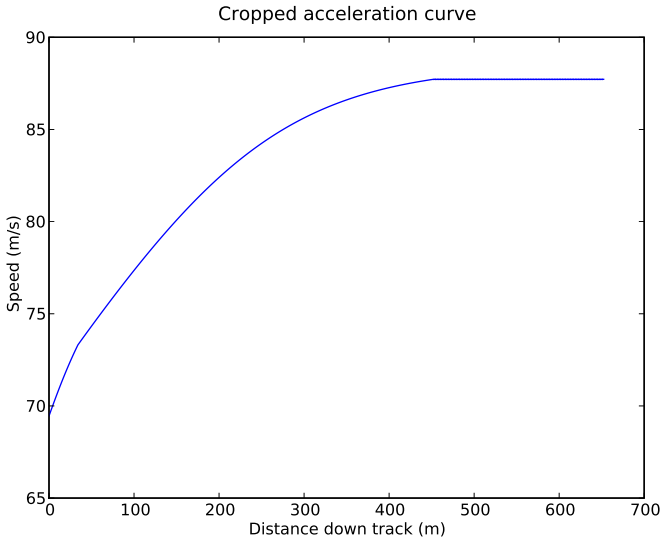

Figure 4: Cropped acceleration curve for Hangar Straight.

graph to determine when a driver would need to brake to reach the entry speed for that corner (see figure 5).

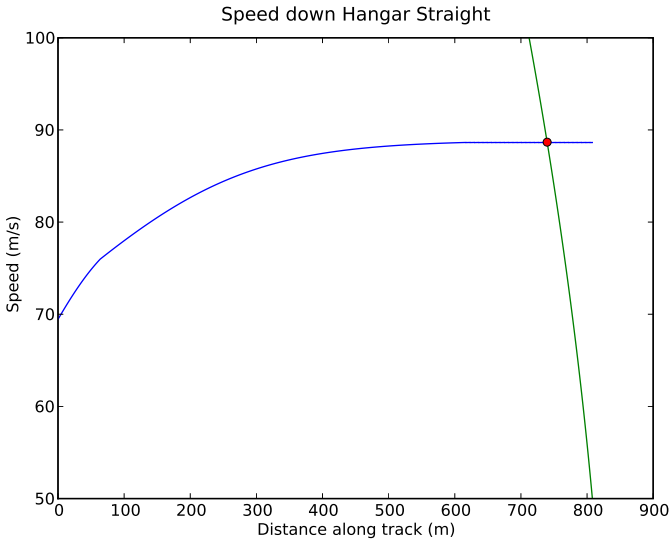

Figure 5: Combined braking/acceleration curve.

This intersection is found using a numerical sampled solver (see appendix A for details), based upon a bisection method.

Once this intersection is found, the simulator sums the time spent braking and the time spent accelerating to find the total time spent in the segment. This process is repeated for each segment along the track to find a total lap time.

2.1 Optimization

The overall lap time serves as a fitness function for the simulator. A simple downhill simplex method is used from the SciPy library is used to optimize the function. A more complex solver was not selected because "creative" solutions (negative ratios, mis-ordered ratios, wildly varying ratios) were more likely to be found, and would be entirely unphysical, much less driveable. Optimal ratios were found within a couple hundred invocations, running in less than ten minutes on a dual-core, 2.2 GHz workstation.

2.2 Fuel burnout

In a race, the car will be run over a wide range of fuel loads. Since fuel accounts for more than 10% of the wet mass of the car, fuel may have a significant impact on optimal ratios. Therefore, the fitness function simulates the car at both low fuel and high fuel, and the averages the results. Optimizing this average should yield reasonably optimal results for the entire spectrum of possible fuel loads.

3 Results and analysis

The solver starts with an initial guess, optimizes from there, and returns the final, optimized ratio set, the final time, and the improvement over the time for the initial guess. This framework provides the ability to investigate various optimization schemes and observe their impact on the final result.

3.1 Whole-track optimization

My initial guess for the gear ratios was an equally spaced setup, with the ratio of the highest gear set to redline at maximum speed. This setup is the most reasonable possible setup in the absence of any form of numerical optimization, and may be how teams currently set up their cars.

When this setup was optimized for the entire track, lap times fell by 0.24 seconds, a highly significant result. Interestingly, this setup is not ideal for any of the individual straights. If the ratios are set up for the back straight alone, shift speeds are raised, improving high-speed performance, but decreasing low-speed performance. Similarly, optimizing for slower and shorter segments causes the car to perform poorly in fast segments. The optimal balance is somewhere in between, and is best found numerically.

3.2 Shift point optimization

I allowed the solver to attempt to optimize the shift point (using a constrained optimization, limited to shift before redline) along with the ratios, however this yielded identical results to before with the shift point pegged to redline as it was before. This means that for the torque curve found on the F1 car, shifting exactly at redline is ideal.

This result is unsurprising because when a driver shifts the car, even at redline past the peak of the torque curve, torque at the rear wheels deceases substantially both due to the longer gear ratio in the higher gear as well as the lower torque produced by the engine at lower speed. Therefore, it is optimal to delay shifts as long as possible.

3.3 Fuel load optimization

Before corner speeds were adjusted for car mass in the model, the optimal setups for high, mixed and low fuel loads were all identical. Postulating that this result was due to the high speeds of the track, where aerodynamic loads are high, I derived a simple test track with slow corners and short straights, imitating a track like Monaco. However, even in this case, the optimal gear ratios were independent of fuel load.

When corner speeds were adjusted, however, the optimal ratio set did become dependent on vehicle mass. The ratio set was optimized three times, once for low mass lap times, once for high mass lap times, and once for mixed mass lap times. Each of these setups was tested and timed for performance in mixed fuel load settings.

Optimization Target	Time
Light fuel	56.949
Heavy fuel	56.918
Mixed fuel	56.916

There is significant advantage to including fuel loads in optimization of gear ratios. The heavy optimization is closest to the mixed optimization, demonstrating that the heavy

fuel load is most sensitive to gear ratio changes. Simply optimizing for heavy fuel, instead of light fuel would most closely approximate the ideal setup for mixed fuel.

4 Conclusions

In a large project, it may be tempting to isolate optimizations. However, in any complex system, changing one variable impacts the optimum value of other variables in many cases. Additionally, optimizing for one part of a task can harm another part. For example, optimizing for the front straight decreased performance on slower sections of the course.

In some cases, optimizations may be independent, and computation time can be reduced by eliminating these parameters from the optimization. In this system, the shift point was always optimum at the redline, and attempts at including it in the optimized parameter set only increased computation time.

A Sampled solver details

The sampled solver used to find the intersection of the acceleration and braking curves is a somewhat novel and complex algorithm.

It begins by ensuring that the graphs actually cross by inspecting both the first and last points out of both sets. The algorithm then subdivides the acceleration set in half, and finds the point in the braking set (using a binary search) that most closely matches the distance of the division point in the acceleration set. By comparing the speed of that braking point with the speed of the acceleration point at the midpoint, the solver determines which half contains the point where the two graphs cross (just as a bisection method would), and recursively continues with that half of the graph.