

Good Advice From Kelly Cline

A Dude Whose Team Won Outstanding in the MCM Contest

Anatomy of a Paper

Summary

This is without a doubt THE most important part of the paper. The difference between an honorable mention and a successful participant is that the judges never read more than the summary of an SP paper. The summary should be written LAST. Let me say that again: You should not write the summary until the rest of the paper is done. In an ideal timetable, all of Monday should be set aside for writing the summary.

The summary must briefly hit all the main points and ideas of your paper. If you did anything creative, it must be here. Further you MUST put numerical results in the summary: “Our final algorithm performed 67.5% better than a simple greedy algorithm, and 123.3% better than a random choice”.

Ideal summary length is hard. You must include all your main ideas in the summary, but brevity is VERY important. I would try to make the summary around half a page, definitely not more than $2/3$.

The summary (and ideally the whole paper) should be written collaboratively: as a team. One way to do this is to have each person individually sit down and spend an hour (at least) writing the best summary that they can. Then, come back together and read the summaries out loud to each other (out loud is very important). After discussing them, set them aside and as a team write a new summary together up on a blackboard.

Introduction

In the introduction, you want to restate the problem as you interpret it. Within an MCM problem, almost every team will find a different ‘problem’ to work on. After the contest if you read the papers from other teams, you will be amazed by how differently they approach the problem. Often they won’t even be working on the same thing!

So what you need to do in the introduction is to clearly explain how you interpreted the problem, and what you decided to work on. It’s also a place to give a little more background on the problem, and show what you learned while researching the problem.

The introduction can usually be written first as a Friday project. It can help make sure that all the team members are in synch about what they are really working on.

The Model

This is the first big section of the paper. I believe that every problem can be broken down into three parts: the model, the solutions, and testing methods. The model is what you’ll use to generate the data that you’ll test your solutions on. For the submarine problem, you would describe the methods of modeling waves traveling through water. For the velociraptor problem, this models dinosaurs stalking and chasing each other. For the MRI problem, this part of the paper would describe how you created simulated data representing biological tissue.

In general, the modeling will occur on the computer, so the challenge is to translate the code into words, and justify each step. In order to create these models for continuous problems, I would recommend having a clear understanding of how to solve differential equations, however many C-problems have not involved this. For the animal population problems, you would want to be able to write differential equations describing the relationship between the predator and prey populations, then numerically integrate them. Know what a partial differential equation is – know the wave equation and the diffusion equation and what they mean. With this knowledge in hand, the concrete slab problem would have looked almost like an old friend!

Remember, this is the mathematical MODELING competition, so do not gloss over this section. It may be simple – for the grade inflation problem, this section would merely involve simulating the actual grades for a class, then using some method to skew them due to inflation. In general, for the discrete problem, you want to be familiar with how to generate random number sets with different properties – this can be very useful in constructing the sets to test your algorithms with. The programmer should be writing the code to create these models Friday morning, and so this section should be roughed out on Friday or Saturday.

The Solutions (PLURAL!) The second BIG section of the problem. Here, we describe our methods of dealing with the data generated by the first section. This is the section that actually describes how we solve the problem. In the submarine problem, this is our algorithm that takes the simulated data from waves traveling through water, and uses this data to

guess at the position of the submarine. In the velociraptor problem, this is our algorithm which mathematically states how the raptor tries to catch the thescelosaur, and how the thescelosaur tries to get away. (Occasionally a problem will explicitly give you the solutions they want you to test. In the Escaping a Hurricane problem, the solutions were to reverse lanes on the freeways and possibly the surrounding roads.)

You **MUST** have more than one solution. Let me say that again: **MORE THAN ONE SOLUTION**. In order to show that you have a brilliant method of finding submarines or cross-sectioning gridded MRI data, you need a baseline, something to compare your solution with. You want to start with the simplest, most obvious algorithm possible, then gradually build on it, refining it until get to your best solution.

Often for the discrete problem, the simplest solution may be merely to make random choices. For the meeting scheduling problem, you might want to have one algorithm which just randomly makes up the schedules. Then when you compare your better solutions with it, they look good!

You want to show that you've explored the problem thoroughly, and that you've tried many different approaches. Even if you started with your best algorithm, then tried a bunch of blind alleys, in the paper you want to present things as if you started with the dumbest most basic solution, then gradually refined it and finally arrived at your best solution.

What if you tried a more sophisticated algorithm, which didn't work well? Put it in the paper! Show all the angles you tried, even if your best solution is not the most complex and interesting one. In real life, that happens very often!

Solution Testing Methods

Sometimes the problem will state very clearly what the goal is, so it makes your algorithm testing methods fairly easy. For the submarine problem, you create a simulation of sound waves propagating through water, bouncing off a submarine, then being received by an array of microphones. This data is passed to your various algorithms which all take a guess at where the submarine actually is. All you have to do is find out how far each algorithm was from the mark and you have an easy method of comparison.

However, you usually have to make some decisions in how you compare the results of your algorithms. In the MRI cross-sectioning problem, you can compare your algorithms' estimate of tissue density with the actual density created by your models for each of the thousand or so points. But do you just average the variance? Maybe you should look at RMS error? Are you concerned with making sure that no point is drastically wrong or that the overall error is small?

With a lot of problems there will be many ways to compare your different algorithms, and there's good reason to use more than one method to evaluate them. Evaluation methods should be one area of brainstorming that you keep working on all weekend.

Results

Here, you need to actually present the results of the testing. This section should be very focused, because you've described everything else. If possible, you want a lot of data to back up your conclusions. Have your models generate a whole bunch of different data sets, and see how your algorithms do. In general, you're going to end up with a lot of parameters to play with – in models, your algorithms, and comparison methods. Try to explore as much of this parameter space as possible. You want to show that you've taken a mature approach to the problem, and probed all aspects of it as best you could.

The specifics of data presentation are difficult. If you can make graphs, by all means do so. In the MRI problem, we made residual plots for each of our algorithms and each of our data sets. But ultimately if you've thoroughly explored the parameters of your models, algorithms, and comparison methods, you're going to have a huge quantity of numbers to present.

Give then the numbers from all your results in tabular format, but don't expect them to read the tables. You want to create a narrative in the text, going through the tables, pointing out the general trends, the exceptions to the rules, and the overall results.

IMPORTANT: Many teams create a model, a solution, run a test of their solution, present the results, and stop. You must run **MULTIPLE** tests! You must find out if your solution is stable! Will it hold up under slightly different circumstances? At this point your code is debugged and generating useful information – take just another 20 minutes and run a few more cases, varying the parameters just a bit. If you're doing the Escaping a Hurricane problem, run your code with a few more cars, or a few less – see how flexible your results are. If you're doing the MRI Problem, test your routines on several different simulated organs. If you're doing the Air Traffic Control problem, throw the planes at your simulated airport a bit faster, a bit slower, or take a runway out of comission and see how your program copes. Prove to the judges that your solution is flexible and stable, or honestly admit that your solution doesn't work as well against certain challenges. Your paper will look a whole lot better!

Conclusions – S&W – Future Work

First, present the bottom line, even if you just presented it in the Results section. ‘Overall, algorithm A performed 34% better than B, and 67 % better than C.’

You need general numbers which summarize everything, so figure out a way to somehow average all the data and distil this into a few numbers which numerically rank the algorithms. The results section is where you say ‘Algorithm A worked better in these circumstances, but B had the advantage when we did this.’ In the conclusions section, you have to boil it down and say ‘A is best.’ This is also what you will need to put in the summary, so be sure you get specific overall results.

Strengths and weaknesses is a great way to go over the key things – explain the good and bad points of your algorithm. I like to use a bulletized list. Nothing new should go into S&W – you are summarizing. The main points of your results section should be here, along with the shortcomings, and any limiting assumptions as well.

In order to show that you are mature in your approach to the problem, a future work section may also be useful. What would you do if you could work on this problem for the next few months? Were there any algorithms that you thought would be great, but just could not impliment on the computer? The MCM is very limiting, so this is a place to show that you can see the big picture.

Overall Recommendations: An skimmable paper

The first thing to keep in mind is that the judges are looking at a lot of papers in a very small amount of time. In their first pass, (in which half of the papers are thrown into the successful participant pile!), the judges can only give FIVE MINUTES to each paper. This means that they’ll read your summary, skim through the rest and then your paper gets tossed in one of two piles. That’s it! Half of the papers never make it past this stage. Even worse, in the second round of judging, your paper will STILL get only about FIVE MINUTES. Only by getting through the first two rounds will your paper be read in any detail. This means that your #1 goal is to beat the five minute read.

We’ve talked about the summary (THE most important part of the paper), now what about the skim? Things that may get noticed are headings, bulletized lists, tables, diagrams, and figures. What you don’t want are large unbroken blocks of text, they make the paper look boring, and may never be read completely. You want the words to be clear and easy to read, and the text to be broken regularly by headings, lists, figures, diagrams, anything you can think of to keep it interesting.

Headings

are very important. If you take out all of the text of your paper, the headings should read like an outline. The judge should be able to read just the headings and see the flow of your paper (which should match the ideas presented in your summary). You want at least a couple layers of headings (heading, subheading, subsubheading) that easily breaks your paper into small pieces, each with a clear purpose and goal. Try not to go more than a paragraph or two without a heading. It not only makes your paper skimmable, it helps the paper stay focused, keeping the paper from wandering. As an example, here is the outline of my team’s outstanding MRI paper.

Summary

- Introduction
 - Assumptions
- The Models
 - Coordinate Systems and Definitions
- Interpolation Algorithms
 - Method 1 – Proximity
 - Method 2 – Density Mean
 - Method 3 – Trilinear Interpolation
 - Method 4 – Polynomial Interpolation
 - Method 5 – Hybrid Algorithms
- Testing and Results
 - Overall Results
- Strengths and Weaknesses
- Future Work
- References and Supporting Data

Bullet Items

By this I mean: ‘To set up this model, we included 4 key ideas’

- First idea
- Second idea
- Third idea
- Fourth idea

These types of lists, whether numbered or not, have three really important purposes.

1. They break up blocks of texts, making reading less tedious.
2. They emphasize important ideas.
3. They are easily noticed when skimming.

Tables of Data

If you get a good computer program working, don't waste it! Run it a few hundred times with all kinds of values of your input parameters (submarine depth, concrete slab size, whatever you can think of). Then organize the data in graphs (if you can) or tables. They can be very impressive even if the judge doesn't read them in detail. They show that you have lots of data to support your conclusions and you've explored the parameters of the problem thoroughly.

Diagrams and Figures

A picture really is worth a thousand words. Diagrams can be very useful in the model section to show how you set up the problem, and figures are always the best way to display data. When my team did the velociraptor problem, I think one of the best things we had were clear diagrams showing our mathematical models for the dinosaurs and how they moved. Describing their allowed trajectories in the paper was very difficult, and I can't even begin to recall how we phrased it – the shapes are geometrically rather complex. However I could sketch out our diagram in a couple of seconds and you would understand the idea completely.

Backwards Design

The most straightforward way to create your paper is to start with the problem, work out your series of solutions, then write your paper explaining your solutions. However, it's also very useful to turn things around: Even without thinking of your solutions in detail, try to design what you want your paper to look like, and spend your time working to fill it in. Essentially this is what I've been doing, sketching out a paper in the most general sense. Once you've looked at the problem, you'll be able to do much better, and my general sketch here may not all apply. Sometime early in the contest, write an outline of what you think the 'ideal' paper should look like. Find a blackboard and outline the major sections and subsections. Make quick sketches of all the figures and diagrams you can think of that might be useful. Brainstorm things to do bulletized lists of and other things that can go into the paper.

Make sure you all look at this outline and discuss it – it will help you all stay focused on the same goal. When you're down in the trenches trying to code out some algorithm or analyze some data it's easy to lose sight of the big picture and spend several hours trying to do something that's not completely essential. By keeping the outline of your ideal paper in your mind, you can let the paper direct where you spend your time. There WILL be some things that you want to do that would look great in the paper, but may just be too hard to do for one reason or another. By working out your paper, even before you've worked out the solutions, you can stay focused and do only the work that needs to be done to make the paper look like it should.

The Team

I'm including this section not because there is one right way to break up the responsibilities during that contest – that's silly – but to try to get you to think about what role you will fill, what responsibilities you can assume: to show you one way to form a team that happened to work. This is how we broke things up when I did the MRI problem and it was very effective. When you tell one team member that their #1 responsibility during the contest is to write the paper it assures that writing begins right from the beginning and that lots of time is spent on rewriting and revision – not just monday afternoon! Assigning a team member to take charge of programming can also be very useful. It is possible to do some problems purely analytically, but the MRI problem was one which demanded a lot of programming, so if possible you should go in with at least one person ready to handle that area. Therefore I think this way of assigning roles is not completely arbitrary.

Right now, think of what role you can fill, and how you can best prepare.

Programmer

Any successful team is going to do a huge amount of programming, and this was the role I filled in my second two attacks on the MCM. Basically everything that really happens will happen on the computer. The programmer must first implement the models to create the data, then code up the algorithms to deal with the data, and finally hack out the comparison algorithms. You'll never know what works or what doesn't until you actually get through the third step in the process. This will involve a HUGE amount of programming and debugging.

Make sure that you know what you'll be programming in, and that you're comfortable with everything. C/C++/Matlab/IDL/Fortran, whatever does the job. If you're doing the continuous problem, do some practice and get some templates built. Make up a little program that integrates an ODE with 2nd order Runge Kutta. Do you know what I'm talking about? – If not, you should!

The bibles are Numerical Recipes and Numerical Methods that Work. These are almost certainly THE most important books you will ever use during the MCM if you're going continuous, and maybe even if you aren't. Diff EQ integrators, a whole bunch of interpolation routines, and just about anything else you might need. This is where you want to look when creating your algorithm. All the codes from Numerical Recipes are up on the web, so you don't actually have to type them. As practice, you should get at least one integrator and one interpolator running. Visualization is also very important. Make sure you've got good software for looking at your data sets. Again, this may or may not be important, but in the MRI problem it was key. MATLAB is great, as is IDL if they're available. Just make sure that you know how to plot out data.

Writer

Now that I've gushed about how important the programmer is and how he/she does everything, I'm going to turn that all around. Ultimately what's written in the program doesn't mean squat. Yes you will turn in your program codes in an appendix, but no one will ever actually read them. The paper is what your team is judged on. Period! If it's not in the paper, it doesn't matter.

The paper must be written very, very clearly. It must say things very simply and precisely. You do not need to make things sound complicated – anyone can do that. Your program is already more than complicated enough. As writer, your goal should be to make everything as simple as possible. Don't write in that phony 'The data were collected.' style. Use personal pronouns: 'We collected the data.' This makes the writing much more active, and much less dry. Also, the key to interesting writing is often a good verb. If you've got a sentence that's dry and boring, try to find the action in it – what HAPPENS! – and try to find a good verb to rebuild the sentence around.

READ THE PAPER OUT LOUD! Let me say that again: read the paper OUT LOUD. You never know how things sound until you actually hear them – it makes problems so much more easy to spot. If things are even the slightest bit confusing, then they must be changed. Remember, you're in the middle of the competition – if you don't quite understand it, the judges won't have a clue.

In the end, I think that paper writing should be a completely team project, but someone has to take the lead. The writer's goal should be to create a thorough draft of the paper by noon on Sunday, covering everything.

You must get everyone to critique every part of the paper, and one of the hardest parts is learning not to take that criticism personally. Writing without an ego is difficult, but that's what the MCM requires. As writer, you need to get chunks of the paper written, then circulate them around, get feedback, and go back and write again. Be comfortable with writing, and rewriting, and rewriting over and over again.

As preparation, you need to do a whole lot of reading. Read every MCM paper you can get your hands on, especially the outstanding ones. The outstanding papers in the UMAP Journal will have a judges commentary, which is EXTREMELY important. This is the only place where you actually get to look into the mind of the judges and see what they are thinking. After you read an outstanding paper, try to predict what the judges will say – do your best to get inside the judges' heads! Get a notebook and jot your ideas down, looking for things that you like and things that you don't. You need to know what makes a good paper!

It's also probably the writer's responsibility to get familiar with whatever word processing package the final paper will be written on. Be sure you know how to do equations, headings/subheadings, bullet lists, etc. I'm partial to LaTeX, but Word has a good equation editor. Make sure you know the basics.

The Third

Writing is the cornerstone of every attack on the MCM, and programming can be almost as important, there are many other important tasks. In order for a team to win, all three people must be working at their peak, so the third must look for things that need to be done.

The first big hurdle is research. Find out as much as you can about the problem, and possible ways to solve it. To be useful in the MCM, research has to be very specific. Dig through math texts looking for specific things that can be implemented on the computer. Get to know Numerical Recipes & Numerical Methods that Work really well, and look for things you can use.

You should be involved in writing as well. Be a sounding board for what the writer has done, and make sure that everything is very, very clear. Everything should flow and sound completely natural. Don't just be a reader of course. This should be a collaboration all the way – everyone should write.

Computer stuff – Can you program? If not – learn! One of the best things any team can have are two people who can program well. When I did the MRI problem, I had one team member who did not know C, but by the end he knew quite a lot. Get in there and learn as much as you can. If you can help with the programming/debugging, you definitely should!

If you know a bit less, then you need to be around for the data collection phase. Sometime on Saturday (or Sunday at the latest), the code must be frozen, and you need to start generating results to quantitatively support your work. The data sets have been created, and most of the algorithms are done, so it just takes a fair bit of time for all of the algorithms to be tested on each data set, and compared with all the methods.

If the programmer is the only one who actually runs the programs, then the team is in SERIOUS SERIOUS trouble. In the MRI contest, I worked at the programs until I thought they were running perfectly, then I gave them to the third, explained what he had to do, then told him to start generating data to support our work. Naturally, the code that worked perfectly for me broke within seconds for him, so I found the bugs and gave things back to him. After about three cycles of this the programs stopped bugging out, and we started generating usable data that ended up in the final paper. I think one of the most important roles for the third is to take the code out of the programmers hands and generate the actual data which will go into the paper. In my MRI team, all three of us ran the programs, although the writer had almost no programming experience to start with.

Teamwork

During the contest you should never find yourself out of a job, with nothing to do, picking away at some peripheral detail that will never make it into the final paper. If you ever find yourself spending time on something that is not really important, not really contributing, then be sure to recognize this as a huge warning sign. A successful team must have every person spending every minute on something really vital. This goes especially for more junior members of a team. If you're a freshman or a sophomore, and you're on a team with an upperclassman or two, it may be harder to find ways to contribute, to stay in the thick of things, and to add important ideas when everyone else seems to be an expert on all this. You must keep yourself involved because there is far too much work for them to do this without you, or with you idling along at half throttle. If you find yourself in this situation go to your teammates and tell them point blank: "What should I be doing? How can I contribute?" If you're a more senior member of a team, then don't take everything on your shoulders. Work with your teammates and don't be afraid to delegate. My biggest problem the second time I did the contest was that I tried to do everything important myself, carry the entire load, and I utterly refused to ask my teammates to do anything really important. We got an honorable mention, but I know we could have done a lot more if we'd been working as a team, instead of me selfishly hoarding everything vital all for myself.

So if you're a freshman or a sophomore, what should you do? How can you contribute? What role should you play? First you must make sure that you understand the math that your team is using in every detail. Suppose your teammates come up with some really cool algorithm to solve the problem: Step one of explaining their ideas in the paper should be to explain things to you. Make them explain things until they become absolutely crystal clear. Remember you have probably the freshest perspective on things, so whether or not you are the official team writer, try to act as a gatekeeper for material to get into the paper. Look for things that don't make sense, that are vague, that are fuzzy and unclear, and then make a big stink about it until the smoke is blown away. A freshman is in a great position to be the official team skeptic. Ask lots of questions, every question you can think of. Why did you do this step? What does this mean? Where does this equation come from? Your final paper should be written in much the style of a textbook, it should be written in order to teach someone about the methods that you used. Of course the best way to evaluate a textbook is to give it to a class of freshmen and see if they can understand it at all, or if the whole thing is just a confusing jumble of jargon. Don't let your teammates snowplow you: If they can't explain what they're doing to you, they don't have a prayer of explaining it in the paper.

A Timetable

Before the contest

Together you should read lots of problems. Be comfortable with brainstorming together. After reading a problem, see if you can break it down into the three parts: What needs to be modeled to generate data? What will your solution algorithm need to do? How can you compare algorithms? Make sure that you are comfortable together, and that when you're brainstorming everyone participates, and you don't have one person dominating.

Thursday evening

– Spend five or ten minutes reading the problems carefully. If everyone agrees which problem to do right off the bat – go with it! The more fun and interesting the problem is to you, the easier it will be to work like hell on it. One of the hardest choices is if you have one problem which looks totally cool and one which is less interesting but you think you could do a better job on it. Given my own experience and that of people I know, I say go with what is most interesting – you’ll be more motivated, and you may surprise yourself. Remember, this contest is not about what you know at the beginning, but what you have taught yourself by the end.

Spend about twenty minutes or so brainstorming. This means

1. Set a loose time limit.
2. Appoint one person to write down all ideas on a blackboard. If each person writes down their own ideas then they become personal property. Instead every idea should be public.
3. Get every idea you can think of on the board.
4. Try to build on the thoughts of your team mates.
5. No criticism or negative comments allowed! That’s for later.

After brainstorming, go back over things in a more organized manner Break each problem into the three main parts: What are you modeling? What ideas for algorithms do you have? How will you compare the algorithms? Every problem has these three components, and you MUST be able to see exactly what they are. Without breaking down the problem, you can never get a foothold on it. Also, breaking down the problem into these standard pieces keeps you from neglecting a potentially important area. Spend some time discussing all three sections, and make sure you all agree on precisely what the three parts involve.

Look for other, less generic ways to break down the problem. You want to chunk out the problem into manageable pieces that can be attacked one at a time. Can you break the problem up using a timetable – a certain sequence of events? What about different classes of strategies?

Identify whether the problem is well focused or very diffuse. If the problem is very big and broad then you have a lot of choice in what problem to do, and how to formulate this in mathematical terms. Later when you read papers from other teams, you’ll be amazed that the teams weren’t really doing the same problem! Remember this throughout the contest – You are not only solving a problem, but creating the problem as well. When I did the submarine problem (the grand-daddy of all diffuse problems!) we formulated things in one way, then found that we couldn’t solve it, and spent the rest of the contest banging around that blind alley. If it’s very diffuse, there may be hundreds of possible problems that you can chose to solve – chose the one that you can solve in the best and most interesting way.

That being said, some MCM problems are well focused and you don’t have a lot of options – there really weren’t too many ways to approach the MRI problem. Right from the beginning, try to recognized how focused the problem is and what choices you have.

At this point you should make a tentative decision about which problem to do. If you focus on one problem the whole time then you will avoid wasting resources. The next step after deciding on the problem is research. Hit the library and find as much information as you can about all aspects of the problem. Get the resources and find out what is there and what needs to be read.

Each time I participated, I held the record for most consecutive hours without sleep. (I don’t remember exactly how many I managed – things got kind of fuzzy near the end.) But you know your own limits. This contest is about pushing yourself to the limits, but don’t go beyond them. This isn’t about making yourself sick! You want to focus every lucid moment of the day on the problem, but you also need to maximize the number of lucid moments you have!

After a jaunt through Numerical Recipies & Numerical Methods that Work, the programmer should start hacking out code – either thursday night or friday morning. You have three huge subroutines to create and you need to get started as soon as possible. Usually the model, creating the data sets, can be done first – get that running as soon as possible. Get a good head of momentum going and sit there programming solid for as long as you can stand.

Friday

On friday, the writer and third should begin with lots of research. Most MCM problems can be seen as specific examples of a general type of problem – MRI was an interpolation problem, submarines and the concrete slab problem had partial differential equations. Find out what type of problem you’re doing, and learn as much about it as possible. Go to school Read textbooks. You have to teach yourself everything you can on the subject. When we did the MRI problem, all three of us put each other through a short course in interpolation methods.

Papers work out best if you can use some established ‘textbook-type’ math to solve your problem. When I did the Velociraptor problem, I think our biggest weakness was that we failed in the research side of thing, and ended up having to create all the strategies ourselves. Later, in reading other problems I discovered that other teams had found ways to incorporate geometry, game theory, and differential equations into their solutions – and the power of the established mathematics made these solutions incredibly effective. Sometimes it’s obvious what kind of math you can use (interpolation methods for the MRI problem). Other times you will have to take some vaguely related area of mathematics, and shoehorn it into the problem, trying to make it fit as best you can. This should be a big research goal – find something established that you can apply.

The writer should start on a draft of the introduction as soon as possible. If you get that done, keep going, rough everything out as thoroughly as you can. It is your job to keep in touch with the programmer – make sure everything he/she puts into the code actually gets into the paper. Get everyone to read everything you write and get lots of feedback. By Saturday, research should be over, and you should be rewriting and rewriting. This is all subject to the whims of the problem of course – in the MRI problem, on Saturday both the writer and the third were running the code generating final data because the paper was looking pretty good.

Keep in touch with each other! Don't get fragmented – this is especially a danger for the programmer. Check in with each other every few hours and explain to each other what you are doing. If you feel like you aren't doing something important or that you are wasting time SPEAK UP! You are responsible for making sure that every minute of your time is spent on something vital! If it's not, then find something to do! This is VERY VERY important!

When I did the Velociraptor problem I had one team member sitting largely idle throughout much of Saturday. The person was working, but not on anything which would go into the paper – and I think this really hurt us. If you see this happening, take action! Don't be afraid to speak up and find something for this person to do – You must use all your resources if you want to win!

Saturday

Sometime on Saturday, the code should be ready to be turned over to the third for data collection. Complete debugging of the code will never happen until someone other than the programmer actually runs it. Hopefully by this time you will have played around with the code sufficiently to spot check your algorithms, but at some point you have to start generating real data which goes into the paper.

Once you get your final results (reversing the freeways allows South Carolina to complete a hurricane evacuation 29% faster), this is the point where you need to run a few more cases to demonstrate the stability of your work. Vary the parameters, the data sets, the number of airplanes, cars, or dinosaurs. Are your results very sensitive to these things, or do they not make much of a difference? It is very important to find out, and to show the judges that you've really put your model and solution through a thorough test.

Also on Saturday, my feeling is that the writer should be almost finished writing. The bulk of the paper should be fleshed out by this point – all the necessary ingredients should be there. In order to see writing objectively, you have to look away from it for a while, so I advise the writer to go help with data collection for a while.

One particular worry sometimes pops up at about this time: You've got your teeth into the problem a bit, you've got something running, and you've got some results, but you're worried that you haven't done anything that every other team hasn't done too. You may be concerned that you've only done the simplest, most obvious things to do, so there is nothing to set your team apart, nothing to really give you that extra edge in this grand competition. If you and your team find yourself worrying about this, I have three suggestions:

1. First and most importantly, whatever type of math you're doing, do it well, do it precisely, do it carefully, do it right, make sure that you understand this corner of mathematics at the deepest level, and then explain it so well and so clearly that the concepts just melt off the paper. The best papers are often ones that do something fairly simple, fairly basic, but they do things and explain things so carefully that it is obvious that they really understand the methods that they're using. That is the mark of a great paper: Use the right mathematical tool for the problem, and make it clear that you understand this tool inside and out, its strengths, its weaknesses, its limitations, why it is useful, and where it will fail.
2. You will often find that your problem is a specific type of a general problem (e.g. a 'knapsack' problem, an interpolation problem, a 'traveling salesman' problem, a differential equation problem), and that a lot of smart people have worked on this type of problem so that there are well established methods for dealing with it. So learn these methods thoroughly, explain them well, and then customize them to your particular problem. Read the problem statement very carefully looking for the specific details and idiosyncracies that make your problem unique, that distinguish it from the general case. I think this is what really put my MRI team over the top: We recognized right off that we had an interpolation problem, so we hit numerical recipes and a couple of textbooks to bone up on what interpolation was and how it worked. Then we noticed that in the problem statement they said that they didn't want to blur out the boundaries between different types of tissue. So to customize our algorithm, we put together a method for finding these boundaries, and turned off our interpolation routines whenever they hit a boundary. I think that was what made our paper stand out from the crowd. We found the right tool (interpolation), we did our homework so that we could explain it in crystal clear detail, and then we adapted this method around the specific details of our problem.
3. Of course it never hurts to strive for some real creativity. Saturday is probably a really good time to go back over your initial brainstorming and hunt for other methods and angles on the problem that you have not pursued. Get your team back together and try to do some more brainstorming from scratch. Try to see if your problem might fall into more than one general category: Could this be approached as either an interpolation problem or as a differential equation problem? That could make for a truly awesome paper if you could show how two very different mathematical tools could be adapted to the same particular problem, allowing you to compare and contrast their effectiveness and their limitations.

Sunday Morning

Sunday morning, tie down the loose ends and GET THE FINAL DATA! Work on visualization, graphing, making pretty pictures etc. You MUST have virtually ALL the data that will go into the tables in your paper by Sunday at noon!

The hardest thing about this contest is knowing when to stop. You're involved in this because you enjoy a challenge. You want to do more than just turn in your homework and get a passing grade – you want to really understand what's going on. You're tired of the wading pool: You want to swim in the ocean. But that means it will be very difficult for you to say "This is far enough," to end your work, stop programming, stop making figures, and focus your whole team on serious writing.

At the end of the contest, all you will submit is your paper, and there's one secret to writing a good paper: Spend a lot of time on it. If you spend a lot of time writing your paper, it will be a good paper. If you spend only a little time, it will not.

So here's the hardest thing, perhaps the biggest challenge you will face during the contest: Can you set a deadline for Sunday at noon, when all work must stop, and everyone puts 100% into writing the paper? Can you stick to this deadline? Can you stop when you are just ten minute away from some great new data? I have known only a few teams with the strength to hold this deadline, and they have all produced excellent papers. After you've written the paper together during Sunday afternoon/evening, you can go back and get those last few things, and add them to the paper. Think about this very carefully: Can you set this deadline? Can you stick to it?

Sunday Afternoon

Okay, in my opinion, here's what put my MRI team over the top and won us the outstanding. On Sunday afternoon we all stopped what we were doing, we read over the draft of the paper, discussed it a bit, and then we threw it away.

Together, we spent the next few hours writing our entire paper on the blackboard. The draft written by the writer at that point was very good, and it made sure that all the important ideas were in the air and that nothing was forgotten. However, no paper written by one single member of your team will ever be as good as a genuine collaboration.

We went through the paper from beginning to end, writing it all as a team, up on the blackboard. We argued over sentence structure and word choice, sweating over each phrase for several minutes, reading it out loud, trying to find a better way to say things. In some places, the existing draft was excellent, and we copied this up on the board – but putting it on the board took it away from any one person and gave the paper to all of us. We filled up every chalkboard in sight, and that Sunday afternoon/evening was what put us over the top.

When you're writing the paper like this, make sure you're all relaxed and comfortable, and don't try to get things done quickly so you can get back to your code. THIS is the single most important task you will do during the entire contest. Relax and don't be afraid to express your thoughts. Each of you should be talking, suggesting ideas, bringing up things you need to include, complaining that a sentence 'doesn't sound right.' If only one or two people are doing the talking, then something is seriously wrong. If any sentence sounds even the slightest bit confusing to you SPEAK UP! Each team member will have a very different perspective on things, and only together can you write a really good paper. Every sentence in the paper should be spoken OUT LOUD at least three times before you go on to the next! Also, regularly trade off who holds the chalk – it helps keep everyone involved.

Okay, by the time you all crash on Sunday, you should have the entire paper worked out. You may or may not have the summary done, but the rest should be complete. Yes you might want to gather a little bit more data for your tables, or find a better way to make a picture of some data, but by Monday morning, the contest is just about over – programming should cease!

Monday

Monday morning should be reserved for writing and rewriting the summary. If you agonized over every word of the rest of the paper, here you should be sweating blood! Find a blackboard and put your summary on it. Don't try to do this on a computer screen!! You MUST make the summary community property. Go over it and over it and over it. Make the summary a work of art, and try every possible way of structuring it that you can think of. Spend time discussing what should and shouldn't go into the summary. Remember, the summary is THE most important thing you will create.

The summary should be done by 7:00 on Monday – three hours before the contest ends (adjust this for different time zones). At 7:00, all work on everything must STOP, and you MUST print out THE FINAL version of the paper. Just for the record the printer WILL JAM, the computer WILL CRASH, and everything that can possibly go wrong WILL GO WRONG on Monday afternoon. The danger of losing the competition just because you could not get the paper printed out in time is VERY REAL. At 7:00 on Monday, drop everything, and work as a team to get a final version of the paper printed out. After you have a final version, if there are one or two things you want to fix – great, work right up until the end. See if you can make another final version (or fix a couple of pages), but do not even think about it until you have a paper which you could seal in an envelope!

The Time Trap

So far we've focused on the right things to do – now let's look at the other side of the coin. I've been on one successful participant team that really crashed and burned and one very near miss. The fact is that half of all papers are ranked successful participant, and most of them deservedly so. There are of course many ways that a team can fail, but there is one pattern that I've seen repeated, both in my own teams and in others.

Let's look again at the tasks your team must complete during the contest.

1. Research
2. Come up with strategies to:
 - Model the system.
 - Solve the problem.
 - Quantitatively compare your solutions.
3. Impliment these strategies on the computer.
4. Use your programs to generate data, and figure out how to display your data.
5. Write the paper.

The most important of these goals is the last one. A well written paper with so-so models/solutions/data will usually trump a so-so paper with well developed models/solutions/data. So how do you make sure to write a good paper? I can give you lots of suggestions and ideas, but the reality is that it's mostly an issue of TIME. If you can sit down as a team with your paper, and spend 15 minutes or so on each paragraph – read it out loud, look for things that aren't completely clear, talk about ways to rearrange it – then you will end up with a top notch paper.

Let's talk about time a little more. When are you likely to waste time? When will you be rushed? Typically, on Friday and Saturday the contest feels very long and it's easy to waste a lot of time. The pressure will grow as Sunday passes way too quickly, and you will be incredibly rushed on Monday afternoon. You end up hurrying through the one task you should spend the most time on!

The roadblock that wastes all your time can often be task #3 – trying to impliment your ideas on the computer. My team spent most of Friday, Saturday, and Sunday on that task when I did the submarine problem. When we finally did get something, it was already Monday. Even if we'd got glorious results, by Sunday night it's too late. We did not have time to spend on the paper, which was lousy as a result, and that's the trap: You spend all your time fighting the computer, so that when you do go to write the paper, you don't have enough time to write a good one.

So how do you beat the trap? In a way I was lucky that we beat it when we did the MRI problem – I happened to know exactly how to impliment our ideas (thanks to Numerical Recipies), so that we had almost all the data we needed by the end of Saturday. The problem clicked with what I knew, so we had a huge amount of time to throw at the paper. You can't count on being so lucky.

In order to guarantee that you will have lots of time to spend on the most important goal, you need to set up harsh time limits on the other tasks, then stick to them as closely as you can. Make yourself stay up Thursday night and Friday night in order to meet the deadlines, so you don't have to stay up Saturday or Sunday nights – which can be deadly.

To do well, you team needs to make Friday the most productive day of the contest by DEMANDING that things get done. One simple way to do this is to come up with your strategies as soon as possible, and insist that they be working on the computer by midnight Friday. If your programmer doesn't think he/she can get the code finished and running by Friday midnight – then you need to make things simpler. This contest is about spending 96 hours on an 96 day problem. Make your programs quick and dirty, but get them working! Stay up Friday night and make sure the code is DONE before Saturday rolls around. If it's friday at 6 pm and you're not going to get the code done within the next few hours, then you're trying to do something too complicated. Regroup, do something simpler, but get that code working as fast as possible.

The trap is an issue of time, and if you're aware of it, you should be able to beat it. Do whatever it takes to make sure that all the research, the programming, and the data collection is DONE by Saturday night at the very latest, then get a good night's sleep. Even if it means rushing through things at the beginning, it's worth it to save the time for the paper. If your team can devote all of Sunday to the paper, then you've already defeated the vast majority of the other teams.