Jethro Update, September, 2026

"Play Of The Hand" Notes, in Q & A format

Q.: The ZBridge.Club website mentioned a new "play of the hand" algorithm for Jethro was released in August, 2026. What's new?

A.: There is a lot going on. This is going to get fairly technical in some places.

Let's start with describing the previous algorithm, and how it was implemented. The algorithm is called "Perfect Information Monte Carlo Simulation", or PIMCS for short. The concept was pioneered by Professor Matthew Ginsburg, the original inventor of GIB (Ginsburg's Intelligent Bridge player). This is now considered a fairly standard way for bridge bots to choose the next card to play; many bridge playing programs use some variation of this.

The idea is relatively simple. The devil is in the details. When choosing a card to play, it goes something like this:

- Create a lot of different "worlds" – different complete hand layouts – which are plausible, given all the information available to the player choosing. Call that player the "searcher." Information available is their own hand, dummy (visible after the opening lead), previous cards played, and shape/strength information advertised by each player during the auction.
- For each world, run a double dummy solver (DDS). Simulate playing each of the searcher's cards which are legal to play. The DDS calculates exactly how many tricks can be made if every hand plays perfectly from the current point in time to the end of the hand.
- Accumulate the information for all of the worlds. You might get information which says for this set of worlds, if I play the ♠K, I will average, say, 3.4 tricks. But if I play the ♦2, the average is 3.7 tricks.
- A simple PIMCS algorithm just picks the card which maximizes the trick count across the set of worlds tested. A somewhat more sophisticated algorithm knows that some tricks are more important than others, and give extra weight to cards in specific worlds which make or break the contract.

- Once the card is selected, there may be some other considerations, such as choosing either the highest or lowest card from “equals,” or choosing the correct card to lead according to partnership agreements.
- At no point, ever, is any “peeking” into the actual hands allowed. That would be cheating. Where’s the fun or the challenge in that?

There is more to it, but that is the general idea.

Q.: That sounds a lot different from how a person decides which card to play.

A.: And it is. As far as I know, no one has ever successfully created a “single dummy” solver. I.e., looking at a single hand and deciding which card to play as part of a unified overall plan. The PIMCS method is much more brute force, leveraging the things a computer does well – super fast calculations and never forgetting anything – in an attempt to simulate what goes on between an experienced player’s ears. But it is not intuitive, which can, on occasion, lead to some weird looking (and bad) plays.

Q.: So PIMCS isn’t perfect. That’s hardly a surprise. Overall, how well does it work?

A.: As a first pass, I thought it was great. It really does do remarkably well.

Q.: I sense a “but...” coming.

A.: Right. There are limitations and several well documented issues. Some of these issues are referred to in the literature as “strategy fusion” and “non-locality.” It often comes down to things like this:

- In any specific world, the algorithm will never “mis-guess” since it is playing double dummy, and knows with certainty the location of every card. The algorithm always finds a missing queen, for example. “Two-way finesse” positions will win in every world.
- It also knows that its opponents will never mis-guess, so there is no reason to try to put an opponent on a guess. For example, say declarer, on your right, leads a low club. You have the Ace, and can see the King and Jack in dummy. You don’t know where the Queen is. Early in a hand, most good

players will duck smoothly, unless they have a very good reason to play the Ace. It won't always be the winning play, but it gives declarer a chance to go wrong. With PIMCS, however, declarer is playing double dummy in every world. Accordingly, declarer will never mis-guess the location of the Ace or the Queen, so a strict PIMCS algorithm is much more likely to play the Ace, solving declarer's potential problem; there are usually a few worlds where declarer led a singleton, and ducking will cost a trick. Similarly, there is no reason for a defender not to lead a singleton King, even if the Ace may be on their right, since the algorithm "knows" declarer can just pick up their King by banging down the Ace.

- Short suit (singleton or doubleton) leads are favored on defense, perhaps to an unreasonable degree. Playing double dummy, the short suit holder always "knows" their partner will get the timing right for a ruff. In practice, their partner is typically less certain to do so successfully.
- Unlike an experienced player, there is no concept of "making a plan" and following through. Every decision about which card to play *right now* is made anew, without paying any attention to what came before. Any time it appears a computer player made a plan and followed through is likely an illusion.
- For example, if a winning finesse is taken, there is no additional weight given to repeating the finesse. Yes, sometimes players correctly duck a winning card, but far more often if a finesse wins once, it makes sense to place the missing honor in the appropriate hand.
- In the same vein, any board where a player has choices to make about which order to try different strategies will be a crap-shoot at best.
- Discovery plays and anti-discovery plays are not a consideration.
- There is no "look-ahead" where playing more than one card is simulated. After the initial test card is chosen, everything is run through the DDS.
- "Safety plays" are very difficult to simulate. Everything is based on maximizing trick count. Sacrificing an entire trick for a 5% gain in the likelihood of making the contract is a difficult decision for the algorithm to make.

- The algorithm doesn't know anything about "bridge logic." For example, "second hand low", as a general principle. As noted above, it is not always correct, but there is a reason it is a maxim.
- Making correct discards can be difficult. If the algorithm thinks it doesn't matter (since the opponents are playing double dummy) a discard that helps an opponent is more likely.
- Perhaps most importantly: The algorithm doesn't know anything about defensive signaling. All it knows is which cards have been played, but does not assign any special meaning to them.

Q.: How did you address those issues?

A.: Before getting into those details, let's go back a bit. I did a bit of hand waving when I said, "create a lot of different plausible worlds." This is not quite as easy as it sounds, and is absolutely critical to creating realistic simulations. One of the truisms about any algorithm is "garbage in → garbage out." If you don't get a good representative set of worlds, you should not expect your results to be very good.

My first attempt at world creation was straightforward:

- Create a very large number of worlds by assigning known cards (seen and previously played) and randomly dealing the unseen cards to the unseen hands.
- Calculate a "similarity score" between each unseen hand and the advertised shape/strength criteria for each unseen hand.
- Rank the worlds based on those similarity scores.
- Use the "most similar" worlds for DDS simulations.

This approach isn't terrible, but there are some problems. It works best when the unseen hands have "normal" shapes. It works less well when the unseen hands have extreme shapes. An example might help to clarify: your RHO deals and opens 3♦. The auction proceeds P-P-P, so you are on lead. Your hand is nothing special: ♠J93 ♥A754 ♦A965 ♣J8. RHO has advertised 7+ ♦, but if you hold your hand constant, and deal a few thousand completely random worlds, there just

won't be all that many where RHO has 7+ ♦. There are also other problems, which I will get to later.

I decided to try to solve the "shape" problem using "combinatorics". Combinatorics is a way of putting all of the 100% known information together with the advertised information (but remember, they might be lying sometimes!) and creating shape probabilities for each of the unseen hands. North advertised 7♦, but they won't always hold exactly 7. Sometimes they lie, and have only 6. Sometimes they might have 8. Deciding how often the players might be lying is something you have to estimate.

The calculated shape probabilities are used as a constraint when dealing out world hands. In the 3♦ opener example, the single most likely hand shape for everyone around the table according to the combinatorics is calculated as N:2=2=7=2, E:3=4=4=2 (your hand, so it is constant), S:4=3=1=5, W:4=4=1=4. This exact shape occurs in 0.73% of worlds with this set of shape constraints.

Looking only at the North hand, the combinatorics give North exactly 7♦ ~82% of the time. Exactly 6♦ ~13%, and exactly 8♦ ~5%.

The combinatorics data is used as an input to the world generator. The exact number of hands with each calculated shape will be generated. In the above example, if you create 10,000 worlds, you are assured that exactly 73 of them will have the shape noted above. So combinatorics gives us many more likely hands to choose from, instead of just hoping for the best with completely random shuffles.

Shape is really important, but is not the only consideration. Strength is also advertised with every bid, sometimes given as HCP (for NT bidders) and WP (working points) for everyone. On preempt hands, the number of "expected playing tricks" is also advertised. If you are interested, my other monographs have very detailed information about how these values are calculated. Once the shapes are set, the strength is calculated for each hand in a world, and compared to the advertised strength information. Then creates a "similarity score" that can be used to rank the hands.

Q.: So now we are good, right?

A.: Not quite. Even when information is advertised, it can be pretty broad. For example, if a player opens 1♦, the information advertised is ">= 4♦, 12.7-21.5 WP". This is likely (but not 100% certain) true information. Even so, there are many, many hands satisfying these criteria which are quite improbable. To try to mitigate this problem, I added a number of "inferences" to alter the "similarity score". These are the sorts of things an experienced player typically does without conscious thought when building a mental of image of the unseen hands, both during the auction and the play. For example, if a player has bid exactly one suit, and there is a major suit which no one bid, the player is unlikely to hold 5 or more cards in the unbid major.

Inferences from the play abound as well. If a player leads a low card in an unbid suit vs. a suit contract, they are unlikely to have touching honors in that suit. Underleading an Ace vs. a suit contract is unlikely. Etc. I am sure I have barely touched the surface of the available inferences. It is an area I need to explore more. "Negative" inferences are especially difficult to diagnose and quantify.

There is another issue with world generation worth mentioning. It is entirely possible to "overfit". For example, if a player makes a bid which advertises, say, 5-5 in the majors, they might not necessarily have that shape. Even Jethro will fudge on occasion, if it seems appropriate. The point is that if every simulated world has that player holding 5-5, it can lead to some very silly plays, where the bot believes it doesn't matter what they play. Except it does sometimes…. To try to handle this, a few "not very likely" worlds are slipped into the sample. But there is no guarantee any particular flaw will be represented.

Finally, there is one more elephant in the room which I would be remiss not to mention. When a human player makes a bid or plays a card, Jethro assumes the action means what it would mean if Jethro had done it. But people don't always do what Jethro would do. This can lead to all sorts of confusion, and incorrect inferences being drawn by the software. This is a problem both in the bidding – was that 3♥ bid a limit raise, or was it preemptive? – and in the play. In particular, if your partnership agreement is to lead 3rd and 5th from length, but you forget

and lead 4th best, the highest ranked simulated worlds are less likely match the actual hands you are attempting to mimic.

The bottom line is that humans are a lot better at figuring out when something unusual has occurred, and dealing with it. Often by just ignoring it.

Q.: It appears you made some likely improvements to your original PIMCS. But it sounds like you still thought it was lacking in some ways.

A.: Correct. I was very much troubled by the fact that the program had to choose a card to play without doing any "look-ahead" to see what might happen after that specific card was played. This led me down an enormous rabbit hole for many months. I still haven't fully emerged.

I was particularly interested in looking at a bridge playing algorithm called "Alpha-Mu." You can find an article describing it here:

(PDF) The {\alpha}{\mu} Search Algorithm for the Game of Bridge

or here,

https://www.researchgate.net/publication/337386923_The_alphamu_Search_Algorithm_for_the_Game_of_Bridge

I am not going to even pretend to say I begin to understand the article. Nonetheless, I did take a couple of things away from it.

First, it creates a game tree, consisting of nodes which represent potential cards played. This has similarities and differences to game trees created for games like chess. It is similar in the sense that every possible move is considered. Subsequent moves create another level of the tree. The tree can get extremely large, as possibilities are explored.

A critical difference between chess and bridge, however, is that chess is a game of complete information. The chessboard is available for both players to see. Bridge is a game of hidden information; you can only see your hand and (after the opening lead) the dummy. Hence the need for "world creation" where you try to get a representative sample of likely worlds.

As “Alpha-Mu” builds its tree, it calculates a value for something called a “Pareto front” at each node – a node represents a potential card to be played -- in the tree. “Maximizing” the Pareto front is how the algorithm decides which parts of the tree to explore further. You will have to talk to someone a lot smarter than I to get a good explanation of what exactly a “Pareto front” is, and how it is used in this context. I didn’t go down that path.

The idea I had instead, was to create a game tree and use the DDS at each node in the tree to determine which play to explore further. The first step is to run a PIMCS simulation and see how that goes. I already knew how to run the DDS, from the work described above.

Suppose you are on opening lead. You typically have about 10 different distinct choices to choose from. (Possibly as many as 13, but most of the time you will hold some “equivalents” such as KQ or 987 in some suit.) Run your worlds through the PIMCS, and see how well each card does. For each card I calculate a value called a “utility”; it is a combination of expected DD tricks and other factors. Then, for the “best” cards, I do a “look-ahead” and let the PIMCS choose a card for my LHO in every simulated world. The card chosen will be the one in each world that the player (who may be an opponent) thinks is “best” for them at that point in time. Rinse and repeat, for each player around the table. If a trick finishes, the winning player leads to the next trick. This is how the tree is built. Build the tree for as long as you want (and your time budget can afford). At each step of the way, the utility of the “best” card is propagated back up the tree to the original searcher’s potential choices. When you decide you are done, pick the card at the top level with the highest accumulated utility.

That is the idea in a nutshell. There is a lot more to it. I have glossed over innumerable details, many of which drove me crazy.

Make no mistake, this is **not** an implementation of the Alpha-Mu algorithm described in the paper I cited. But since it was inspired by Alpha-Mu, I call it “Alpha-Mu Lite”. Technically, I suppose the most correct description is “Single Observer Monte Carlo Tree Search,” which doesn’t exactly roll off the tongue. I think “Alpha-Mu Lite” sounds better, so that is what I am calling it.

Q.: As you build the tree, how do you decide which path to go down? For every look-ahead card, there are usually choices to be made.

A.: Correct. There is no point in closely examining plays which are judged to be obviously bad. You want to look more closely down paths which might be fruitful. You also need to assume the opponents are making their best plays at each step. I used ideas found here:

https://eprints.whiterose.ac.uk/id/eprint/75048/1/CowlingPowleyWhitehouse2012.pdf

Once again, that is an academic paper which is difficult to wade through. My algorithm makes some critical changes to the algorithm presented.

An important part – perhaps the single most important part -- of Cowling's algorithm is to decide the extent to which you want to "explore" the tree by going down new paths, vs. "exploit" the paths you already believe are good. Tuning those parameters is non-trivial. I am sure improvements to my algorithm can be made here!

Q.: And that's it, eh?

A.: Um. Not so fast. Literally. Calculation speed is a big issue. There are several different things it is desirable to maximize:

- The number of worlds tested
- The number of "look-ahead" cards you play
- Speed of the DDS

The DDS I am using is open source C++ code from Bo Haglund and Soren Hein. See http://privat.bahnhof.se/wb758135/index.html for details.

This is a fantastic piece of freeware. I have never seen it fail. The CPU time needed to solve a single board depends on the complexity of the hand, but averages somewhere around 200 milliseconds on my computer. So solving 100 boards one at a time might take 20 seconds. That is way too long to expect anyone to wait. The DDS can be set up run in "batches" so instead of solving boards in succession, you can send in whole bunch to be solved simultaneously.

(Let me repeat: This is *really* good software!) The speed in batch mode improves by a factor of about 7 (in my experience) but even that really isn't fast enough to do what I want.

What I was finding was that even with a relatively small number of worlds (e.g. 80), and only a couple look-ahead cards, it just wasn't fast enough to build a very deep tree. My goal was that it should never take more than 5 seconds to choose a card to play, and I really wanted the typical time to be in the 3-4 second range or (hopefully) less.

I didn't have any way to know for sure, but it seemed to me that my Alpha-Mu Lite algorithm would not be particularly useful unless I had at least a 5 card look-ahead horizon, and I really wanted more than that if possible. But as more look-ahead cards are played, the tree becomes deeper, and the number of plays you actually have in the ultimate "leaf" nodes can get quite small. A critical mass of examined hands at the "leaf level" is necessary to have confidence in the path chosen.

Q.: So you were stuck?

A.: Just about. But Anik came to the rescue! When I explained the problem to Anik Mitter, the creator of ZBridge.club, he proposed an ingenious solution. Instead of using the DDS, which calculates exact trick counts, Anik said he could train a customized neural network, using millions of hands, to simulate the DDS calculations. The results from the neural network wouldn't be the exact same results as the true DDS, but should be close enough to be useful. The loss in accuracy would be offset by a speed improvement of at least an order of magnitude.

Well, hot damn! That was definitely worth a try. I spent a lot of time whining that I wanted the model to be more accurate. Anik patiently explained how accurate the model actually was, and politely told me that that was about all I could expect. There were some bumps along the learning curve, but we got it going, and I implemented it inside my Alpha-Mu Lite algorithm.

Currently I am running with a batch size of 512 worlds, and a look-ahead horizon of 13 data batches. All within the five second time budget. I have experimented

with many different combinations. For now, that is the one I am using. More testing might suggest a different configuration would be better.

Q.: You are looking 13 cards ahead?

A.: Technically, yes, that is possible. In practice it doesn't work out that way. Every batch *might* expand the look-ahead another level, but you don't really want to do that. You need to fill the tree in enough to have some confidence in the choices made. Deciding exactly when the choices made have accumulated enough data to be "trusted" is one of many, many design decisions. Again, I suspect this is an area where I could make improvements.

Q.: And it consistently finds the best play?

A.: I wish. First, it is often not clear what the "best" play might be. When analyzing a hand, we tend to look at all 52 cards, and often say, "Obviously, the right card to play now is <whatever>." When the cards are hidden, it is often a lot more difficult. We can be blinded to the alternatives, since we are looking at the actual layout. A card that is obviously best for the actual layout may not be best for other plausible layouts.

Second, even if there is a technically "best" play, the margins are often very small. Between the "worlds" created, modeling the DDS instead of having a true DDS, and all the different levers I have when making algorithmic decisions, there is a not-insignificant amount of noise introduced. For example, just creating a different set of equally plausible worlds often leads to different outcomes.

For example, suppose the true "best" play is, in reality, 0.1 tricks better than other plays. It is not unusual for random "noise" to overwhelm those 0.1 tricks, leading to a different choice.

In this respect I think of the algorithm as "fragile" or "brittle" instead of "robust." Surprisingly small adjustments can lead to different outcomes.

Q.: Is this how other modern bridge bots choose which card to play?

A.: Honestly, I have no idea. Despite diligent digging, I really could not unearth much information about how exactly other bots do this job. The description

above is more than I found. What I described might be a brand new method. Or maybe everyone is already doing it. I just don't know. (Either way, I claim dibs on "Alpha-Mu Lite" as the name for this algorithm!)

Q.: How well does your "Alpha-Mu Lite" work, compared to the original PIMCS?

A.: I don't have hard data, but my sense from examining countless hands is that the algorithm seems to alleviate some of the strategy fusion issues associated with PIMCS. As an added bonus, I believe it also makes the play seem a bit more human-like. There are the occasional "clunk" moments – why the heck did it do that? – but all in all I find it to be eminently playable. It is far from world class, whatever that means. Michael Rosenberg -- generally acknowledged as one of the best declarers in the world -- has nothing to fear.

Q.: Are you done tinkering?

A.: Not at all. I feel like I have really barely scratched the surface. There are dozens of parameters to play with. I have no doubt the algorithm can be further improved. I also haven't spent much energy trying to make the algorithm faster. For example, cutting off the search ahead of time, when the likely best card emerges.

Q.: Does the algorithm "make a plan" now and follow through?

A.: No. Having a true "single dummy" solver remains the holy grail for bridge bot play simulation. This doesn't do that. What it hopes to do, is look ahead far enough to be skeptical of the original PIMCS suggestion when appropriate.

Q.: What about defensive signaling?

A.: Still a pipe dream. Very difficult, imho. But definitely near the top of my list of places where I want to take my next deep dive. Even for human players this can be difficult. First, the potential "signaler" has to decide if it is appropriate to signal. If so, what kind of signal: attitude, count, or suit preference? Then, the other defender has to decide if (a) a signal was actually given, and (b) what to do about it. The declarer can also see which cards have been played, and alter their

strategy appropriately. This is a big chunk to bite off. There is a good reason bridge bots have struggled with this!

Q.: How about an example?

A.: Sure. Let's look at one entire hand in excruciating detail, from the bidding all the way to the end of the play.

Dlr: N Vul: E/W	♠ A742 ♥ T632 ♦ A6 ♣ AT8	
♠ KQ6 ♥ 87 ♦ K743 ♣ Q952		♠ T3 ♥ K954 ♦ QT952 ♣ 63
	♠ J985 ♥ AQJ ♦ J8 ♣ KJ74	

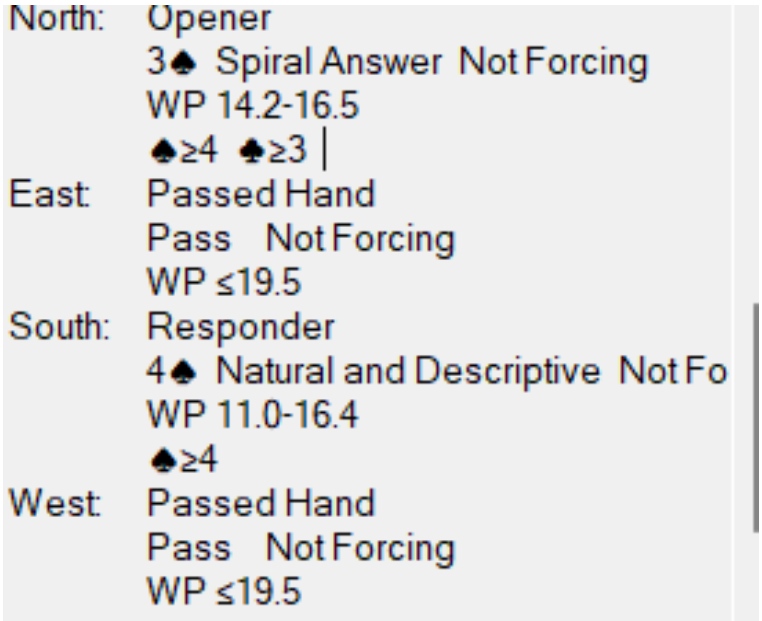
West	North	East	South
	1♠	Pass	1♠
Pass	2♠	Pass	2NT
Pass	3♠	Pass	4♠
Pass	Pass	Pass	

A note about the auction: South's 2NT bid is an optional treatment often called "spiral" which Jethro plays by default. (Or, if you prefer, you can turn it off; personally I find Spiral to be extremely useful.) 2NT is an artificial bid which asks North for more information about their 2♠ bid.

Jethro really likes Aces, and the North hand has 3 of 'em. The 3♠ bid describes a hand with 4 card spade support, and maximum values for their previous action. If you think the North hand is a minimum because it has only 12 (traditional) HCP, well, Jethro disagrees with you. Before the auction, Jethro initially evaluates it as worth 14.06 "Working Points." In support of Spades, the North hand re-evaluates to 14.71 WP. The strength cutoff for the 3♠ spiral response is 14.2 WP, so Jethro is strong enough to show maximum values.

Across from an opening hand South Jethro was never going to stop bidding below game, but they wanted to know how many Spades their partner held. With a different hand, say 3=4=1=5 shape, North's 2nd bid might also have been 2♠ despite having only 3 Spades. If N/S were not playing Spiral, South would just jump to 4♠, when they might have preferred playing 3NT instead of 4 Spades with a 4-3 fit.

At the end of the auction, this is the public information which has been advertised by each player around the table:



This is the information which each player uses when constructing possible worlds to help them choose a card to play. It is augmented, of course, as cards are actually played and hit the table. As you can see, the information is fairly sparse. The E/W hands passed throughout, and all this information really says is that

neither E nor W had a had strong enough hand to insist on doing “something.” There is no information to indicate, for example, that East had an opportunity to make a 1 level overcall in a suit but failed to do so. As mentioned above, negative inferences are very difficult to assimilate.

But enough about the auction. This is supposed to be an article about card play.

West has an unattractive hand to lead from. A Spade lead is likely to blow a natural trump trick, and leading away from unsupported honors in the minor suits has no appeal either. East might have a few points, but it is highly unlikely to be more than a few. With unattractive options in three suits, West leads a ♥. The computer (correctly) sees the 8 and 7 as equivalent. Additional “bridge logic” chooses the 8 as the correct card to lead from this holding, according to the E/W partnership agreement. (No, that is not an agreement you can change; Jethro will always lead high from doubleton.)

North (dummy) has no reason to do anything other than play a low card.

What about East? Given the cards that East can see, East can draw an inference that the 8 is likely West’s highest card in this suit, and that West likely has no more than 2 Hearts. That information is used when East creates worlds to simulate the hands they cannot see. The simulated worlds are run through the Alpha-Mu Lite algorithm, which indicates playing the King is an inferior play. So East plays low.

South cannot draw the same inference about West’s ♥ holding. There is just room for West to have led the 8 from an original holding of K98. Not that it matters right now, when deciding which card South should play. South wins trick 1.

What should South do at trick 2? A person would look at the hand, and likely come to the conclusion that there is no particular reason not to start drawing trump, and would try to figure out how to best do that. To simulate those thought processes, once again Jethro creates plausible worlds and runs simulations where a total of 13 cards are played in order. To help make this particular decision, a game tree is built with almost 12,000 nodes needed to represent the possible paths examined.

After all that, South Jethro emerges by leading the ♠J. Is that the "best" technical play? I dunno. Maybe, maybe not. But it is not entirely implausible. Maybe we squash a stiff ♠T.

I tried making small adjustments to the algorithm parameters, to see if other choices might be made. And indeed they were. Sometimes South led the ♠9, and let it run when West doesn't cover. Sometimes South led the ♠5, and finessed the ♠7. None of these choices seems entirely implausible to me.

Back to what actually occurred: West covers the ♠J with an honor, and North wins the ♠A. East plays low; the ♠T did not drop.

Now North has to lead a card to trick 3. There are two confirmed natural spade losers. There is no "memory of a plan" which South had before leading to trick 2. Once again, new worlds are created, and simulations run. North continues by leading another Spade. Was there a "better" play? Once again, I don't really know, and I can see all four hands.

And so it goes. At every decision point.

In any case, East wins trick 3 with the ♠T, and has to lead to trick 4. There are undoubtedly some inferences which East should be able to draw from declarer's previous plays, but they are beyond Jethro's (and my) ability to figure out.

For better or worse, East leads back a ♥ at trick 4. Maybe West can ruff. Or maybe it would have been better for East to lead a ♦. I really don't know, and I am looking at all four hands. East can only see their own hand and the dummy, and all of the worlds they simulate.

Should South finesse the ♥J? South still does not know for sure that West doesn't have the ♥K. South eschews the finesse, and plays the ♥A, winning the trick. This may look a bit surprising. We can see that the finesse would have won on the actual layout. Was the finesse the better play? If West ruffs the ♥A, South will likely be wishing they had finessed.

But look what happens next. South won trick 4 with the ♥A, and immediately leads back the ♥J. East will win this trick with the ♥K, but now the ♥T in dummy will eventually let declarer pitch a losing ♦ from their hand.

This is the layout when East is on lead at trick 6:

Dlr: N Vul: E/W	♠ 72 ♥ T ♦ A6 ♣ AT8	
♠ K ♥ - ♦ K74 ♣ Q952		♠ - ♥ 4 ♦ QT952 ♣ 63
	♠ 98 ♥ - ♦ J8 ♣ KJ74	

Leading to trick 6, East still doesn't want to try a ♦, or break the ♣ suit. So East leads back their last ♥. Declarer pitches their losing ♦, West also pitches a ♦, and North wins the ♥T in dummy. Should West have ruffed in with their ♠K instead of pitching a ♦? Or should West have pitched a ♣ instead?

If West does ruff in, they will have to decide what to lead next. Having run this simulation many, many times, I can tell you that West always leads back a ♣, solving declarer's guess in the ♣ suit. We can see that a ♣ is wrong on this layout, but the simulations really don't like leading away from the K♦ here, not knowing South's minor suit holdings. We will come back later, and force West to ruff in and lead a ♦, giving declarer some different decisions to make.

In the meantime, what happens when West discards, and North wins the ♥T?

Is it time to try to guess the ♣? No, declarer goes down a different path. At trick 7, North cashes the ♦A, and leads another ♦ at trick 8, ruffing in hand. This leaves West with these cards:

♠ K

♥ -

♦ -

♣ Q952

South exits with their last ♠, and West has to lead a ♣ after winning the ♠K. A strip and endplay! Yeehaw! South will easily take the rest of the tricks now. The bridge programmer geek in me finds that pretty exciting. To be clear: there was never any "plan" to do a strip and endplay. The simulated play led to this result nonetheless, based on the accumulated statistics.

As promised above, however, let's go back. Suppose West ruffs in at trick 6, and cruelly leads a ♦ to trick 7. Now what does declarer do? They win the ♦A, and then ruff their last ♦. They cash another ♠. Now they really do have to guess the ♣. And they get it wrong, playing East for the ♣Q. Well, that is kind of a bummer on this particular layout. Was that a good play or a bad play? To be honest, I have looked at this hand far too long to be sure. I have tried other subtle variations on the play sequence, where South successfully played West for the ♣Q.

At the end of the day, there was some good and some not so good.

Q: That was … a lot. Maybe more than was really necessary….

A: Yes. If your eyes glazed over and you skipped to the end, I won't be offended.

One of the reasons I went through this hand in painstaking detail was to try to convey the idea that nothing is as straightforward as it might seem. The algorithm used is not at all similar to the kind of thought processes a person might employ. On those occasions when the program does something strange (and bad) the goal is to figure out why, and see if the problem can be mitigated using the tools available.

I suppose one of my motivations behind that detailed explanation was to respond to a hypothetical, “But it is obvious to do <something>. Why didn’t it do that?” There is a good chance I agree with you. But getting the program to find that apparently “obvious” play may be a completely different issue.

Q.: What’s next?

A.: As noted above, there are plenty of places to continue to explore. This remains a work in progress, and will likely never be “done.” If you want to continue the discussion, drop me an email: JethroTBW@gmail.com

♣ ♦ ♥ ♠