PROCEDURALLY GENERATING ECONOMIES WITH GRAPH GRAMMARS (AND MATH)

Mark Gritter (novice Artificer)
Roguelike Celebration
October, 2020

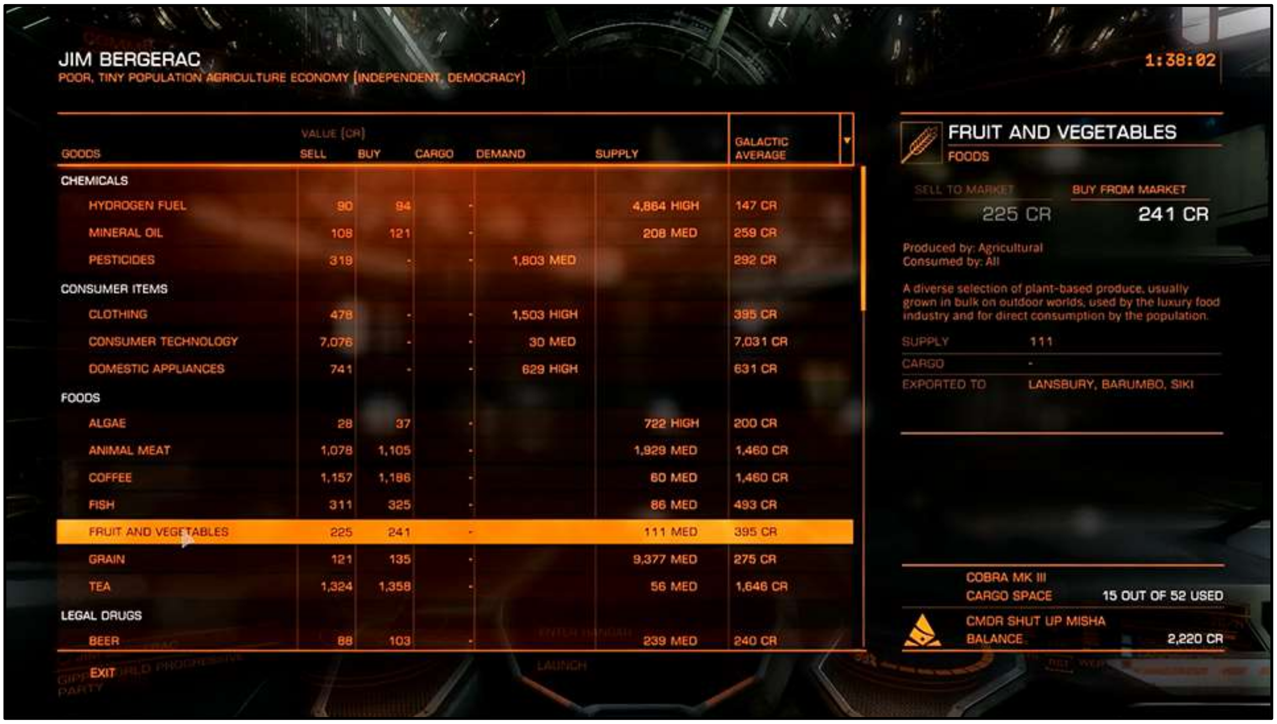
```
Procedurally Generating

  ######  ##### .####. ##   # .####. #    #  ###   ♦##### /###¶#
  #      #      #    # # #  | #    # #\  /#   $    #      #
  #@###  #      $    # #  # # #    # # ## #   $    +####  ######
  #      #      #    # #  # # #    # #    #   $    #           #
  ######  #%### .####. #   ## .####. #    #  ###   #%#### #####/

      o----o     o-o       o
      |             \      |
with  |  o-o o^o o--o o--o o--\  Grammars
      |    | |   |  | |  | |  |
      o----o o   o--o o--o o  o           +-----------------------------
                      |                   | Roguelike Celebration 2020
and                   o                   |
                                          |
 M=\ /=M /aaa\ ==T== |   |                | Mark Gritter
 |  V  | 1   1   t   |   {                # (he/him, novice Artificer)
 |     | 1___1   t   |HHH|                | @markgritter
 -     - 1   1   0   1   }                |
```

My introduction to roguelikes was NetHack (on a PC.) One of the things I loved was the shopkeepers!

Source: http://crpgaddict.blogspot.com/2012/12/nethack-he-coulda-been-contender.html

I played a lot of Elite, too, which has trade as its focus…

Source: https://www.filfre.net/2013/12/elite/



… and still has that as a major component in its current version.

Source: https://www.ign.com/wikis/elite-dangerous/Trading

In an RPG maker called Stuart Smith's Adventure Construction Set I spent so much time building shops I usually didn't finish building the adventures I started.

Image from: https://www.mobygames.com/game/dos/stuart-smiths-adventure-construction-set/screenshots/gameShotId,604603/

Dwarf Fortress has trade, too!

Source: https://df-walkthrough.readthedocs.io/en/latest/tutorials/trading.html

QUESTIONS

Who buys all the junk you sell to shops? Why?

What does the merchant *do* with all the stone tableware your fortress sells them?

How does an entire planet end up as "agricultural"?

Does trade make a difference to the game world?

But I have questions about what's going on here.

…

Most of all, does trade and the broader economy make a difference to the game world?

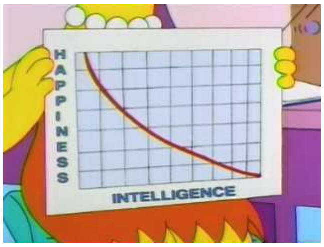
BUILDING SLIGHTLY MORE REALISTIC ECONOMIES

1. Represent the flow of items with a graph.
2. Optimize for maximum happiness (utility).
3. Alter copies of the economy graph to create regional differences.
4. Add trade, which improves the utility by exchange of goods.

I'm going to talk about my experience with a project called Emojiconomy. It builds toy economies through these four steps:

1. Build a graph that represents the flow of items.
2. Apply a utility function that has decreasing marginal utility, and sets the choices represented in the graph to maximize utility
3. Create different regions each with a copy of the graph, slightly altered to "break" (or maybe "specialize") each region.
4. Find trade routes between regions that exchange goods to bring the utility back up.

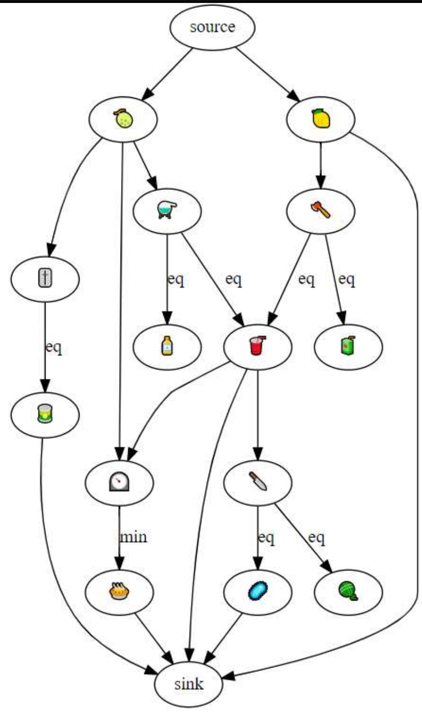
1. I MAKE A LOT OF GRAPHS



Each node in the graph represents either:

- a good (a raw material, finished product, etc.)
- a process (a factory, a workers, etc.)

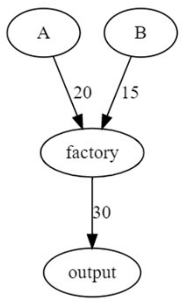
The edges in the graph represent flows of goods.



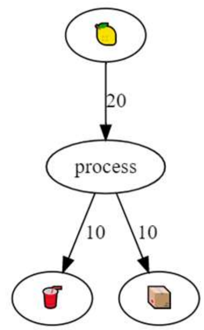
Here's the sort of graph I'm talking about. (Not the one Lisa Simpsons has made.) It has circles (called "nodes") and arrows (called "edges"), and each can have some label attached. Each node represents either a good (like the melon and lemon at the top) or a process that transforms goods (like the alembic or the axe.) From an initial set of "natural" goods like plants, the economy produces a variety of other consumables.

TRANSFORMATIONS

A “factory” requires equal numbers of two (or more) goods and combines them to produce an output good.



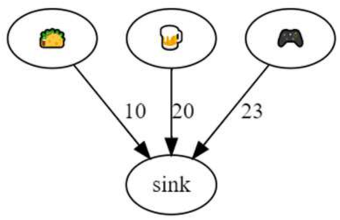
A “processing plant” takes a single input and splits it into different outputs, in equal amounts.



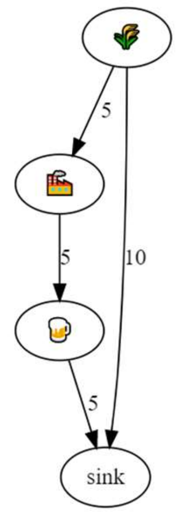
In my prototype, I had just two types of transformation of goods: “factories”, that put two units together to create a single new unit, and “processing plants” that split a single unit into component parts.

CHOICES

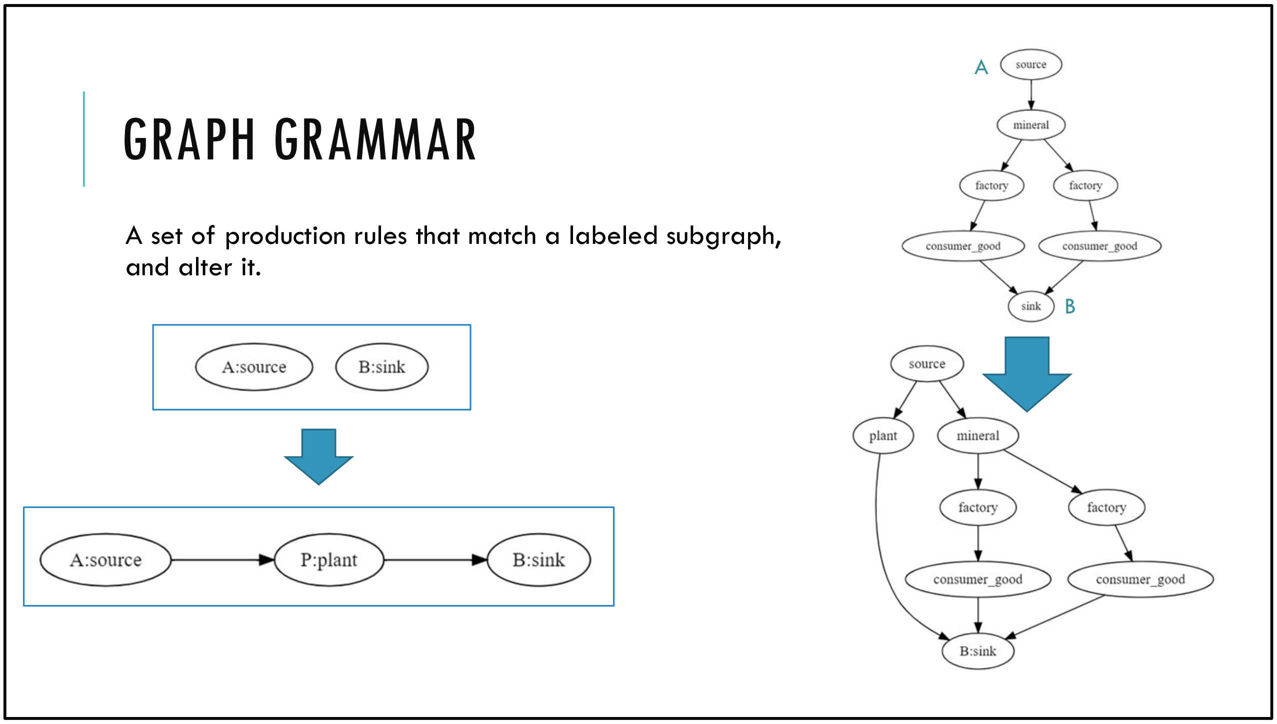
The “sink” represents consumption of goods by the population.



If a good has more than one output, it represents a choice in how it is used.



The sink of the graph represents consumption of goods by the population.
In a graph like this, a good could have more than one possible use: we might eat a plant directly, or choose to process it into beer. This gives us some “play” in the economy, we have to make decisions rather than everything just flowing through the graph.

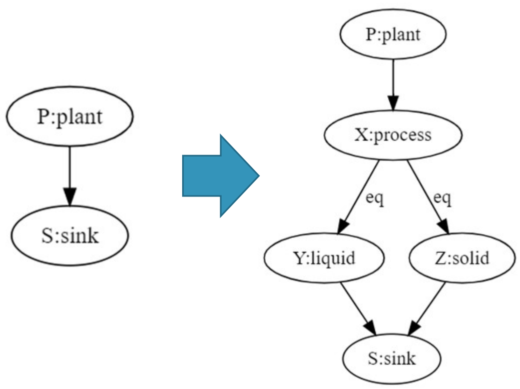


In Emojiconomy, these graphs are produced with a graph grammar.

Graph grammars are a set of production rules. Each matches some labelled nodes and edges within a graph, and then makes am alteration. For example, on the left-hand side is a rule that takes a source and a sink node, and then constructs a new “plant” node in between. The right hand side shows this being applied to an existing graph.

WHY GRAPH GRAMMARS?

They can preserve structure! Like “every good can be used somehow”.



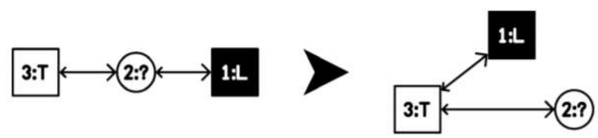


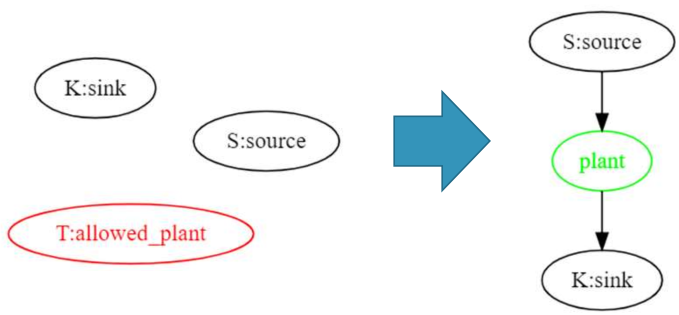
Fig. 9. Rule to move a lock forward. The circular node marked with a question mark indicates any node. Applying this rule will not change the type of this node, only its location and connections in the graph.

From “Generating Missions and Spaces for Adaptable Play Experiences”, Joris Dormans and Sander Bakkes (2011)

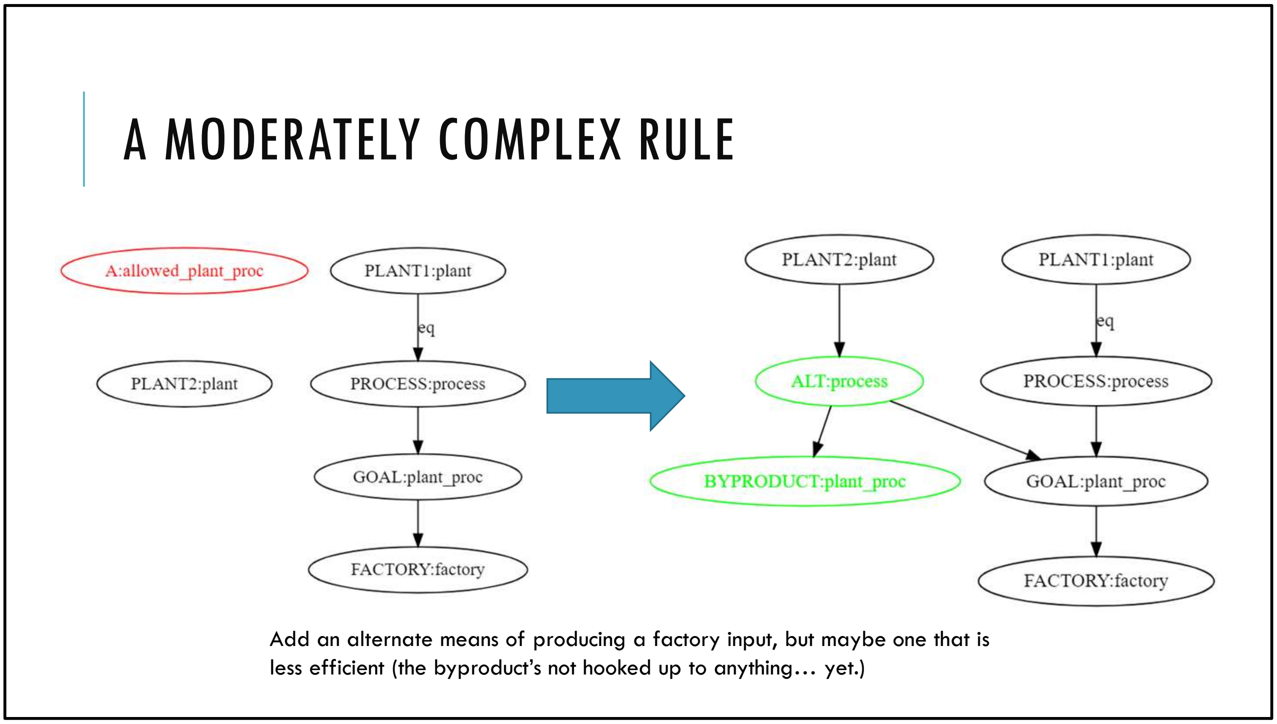
Joris Dormans has used graph grammars to great effect in “Unexplored”. Their strength is that they make it easy to preserve structural invariants. In a dungeon, the graph grammar knows not to move the key behind the door it opens. In Emojiconomy, we can ensure that inputs or outputs remain hooked up, so that there aren’t any consumable items just hanging out being unused.

STUPID GRAPH GRAMMAR TRICKS

Want N of something? Have a rule consume a token, and start with N tokens.



[cut for time] Here's a technique I use frequently: if you want to do something at most N times, add N tokens to the initial graph. Then write your rules so that they delete those tokens while performing the action you want to limit. So, if you want at most 3 plant species in the economy, there are only three allowed_plant tokens in the initial graph.

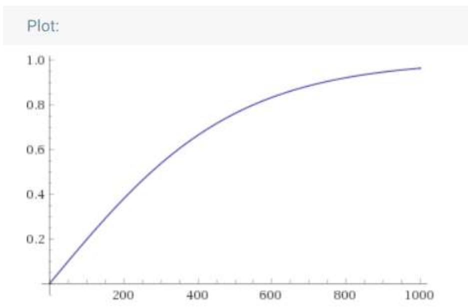


Here's an example of a moderately complex rule. It says if there's some good that's used by a factory, and we have a token that allows another plant product, then create a new process which produces that same good, as well as a byproduct not hooked up to anything. Another rule could use that, or maybe convert it into a garbage type with negative utility.

2. UTILITY

Utility = value or happiness

The important thing for the model is that the added ("marginal") utility of each unit of a good is decreasing!



I picked this sigmoid function

f(x) = e^x / (e^x + 1)

translated and scaled to max out at a utility of "1" and get close to that at about 1000 units.

f(x) = 2 * e^(0.004x) / (e^(0.004x) + 1) - 1

In economics terms, utility is what people attempt to maximize. It's sort of circular. Virtually any real good has "diminishing marginal utility" which just means that the N+1th of it is not worth as much to you as the Nth. The utility you get from having 999 lemons is just about as much as having 1000 lemons, much smaller than the difference between 1 lemon and 2 lemons.
So, I modelled this with a sigmoid function, which asymptotically reaches a limit.

OPTIMIZING

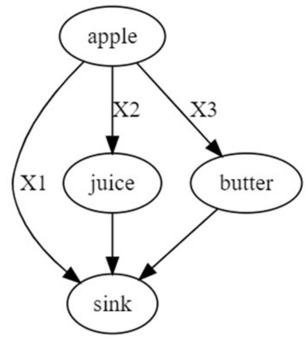
We’ve got a set of choices:

X_1 + X_2 + X_3 = 100%

and a utility function to maximize

f(apple) + f(juice) + f(butter)

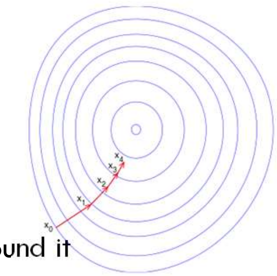
But our objective function is nonlinear!



So what does this give us? We have a set of choices about what to do with goods. In the simple graph here, we’ve got apples which we can eat directly, make into juice, or use for apple butter. The constraint is that that our choices have to add up to 100%.

And we have a function we want to maximize under those constraints. Except our objective function is nonlinear, and so are the effect of our choices, so we can’t use linear programming!

GRADIENT DESCENT



This is actually super-easy* to build, compared with all the math around it

1. try increasing each allocation X_i by a little bit, recalculate utility

2. move some in the combined direction (all X's) that increases utility
   - (if you overshoot, move less next time.)

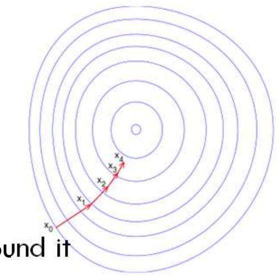
3. project that point back into the allowed space (everything has to sum to 100%)
   - Yunmei Chen and Xiaojing Ye, "Projection Onto a Simplex", 2011 (!!!)

4. continue until good enough!

* *well, no harder than your average lighting algorithm?*

The solution I picked is a standard method called gradient descent. A lot of the math is abstruse, but this is really a very simple idea.
First, measure the "slope" around where you are. We can do this by changing each variable just a tiny bit and seeing whether the utility gets worse or better.
(In some real-world cases you can use calculus instead to compute the slope, but that seems challenging here.)
Once we've done that, take a larger step in the direction that increases utility the most– the direction of steepest upward slope.
That step might make some of our constraints add up to 100%. The solution is to readjust them to the nearest point that does add up to 100%. There's a very cool and simple algorithm that does this, which was only invented in 2011!
Then we just repeat each step until we decide it's good enough.
A lot of the technical stuff in the descriptions of Gradient Descent are (1) how much to move, and (2) how to decide you're done, but really it works fine if you don't get those exactly right.

GRADIENT DESCENT



This is actually super-easy* to build, compared with all the math around it

1. try increasing each allocation X_i by a little bit, recalculate utility

2. move some in the combined direction (all X's) that increases utility
   - (if you overshoot, move less next time.)

3. project that point back into the allowed space (everything has to sum to 100%)
   - Yunmei Chen and Xiaojing Ye, "Projection Onto a Simplex", 2011 (!!!)

4. continue until good enough!

* *well, no harder than your average lighting algorithm?*

The solution I picked is a standard method called gradient descent. A lot of the math is abstruse, but this is really a very simple idea– move in the direction that makes things better. If you end up making things worse, take smaller steps.

One problem is that we might end up with choices adding to more than 100%. The solution is to readjust them to the nearest point that does match our constraint. There's a very cool and simple algorithm that does this, which was only invented in 2011!

REAL ECONOMIES AREN'T PERFECT

So yours doesn't have to be either!

Maybe suboptimal choices are opportunities for players!

Which brings me to a good point, real economies aren't perfect. Yours doesn't have to be either. I think it's cool for players to notice there's a suboptimal choice, if they have the power to fix it!

3. MESS WITH THE GRAPH

Maybe region X doesn't have apples.

Maybe it doesn't have an apple-juice press.

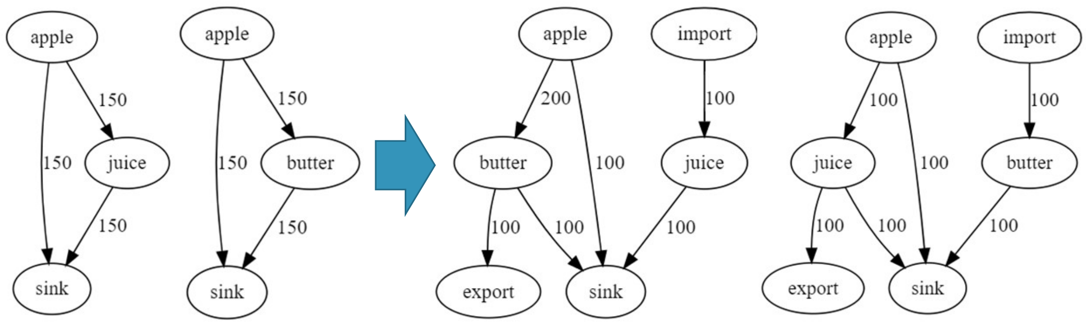
Maybe its residents don't eat raw apples! Or their utility function is different in some other way, like a preference for pears.

Differences can be represented as missing nodes or edges in the economic graph. We can then calculate new utilities for each region.

Now that everything working mostly like it should, we can start breaking it. Our perfect economy might fail in different ways. Maybe some region can't grow apples. Maybe it lacks the infrastructure to make apple juice.

These differences can be simulated by removing parts of the graph. That usually lowers the utility, because there are not as many goods to choose from in each region.

4. TRADE



Utility = 0.5826

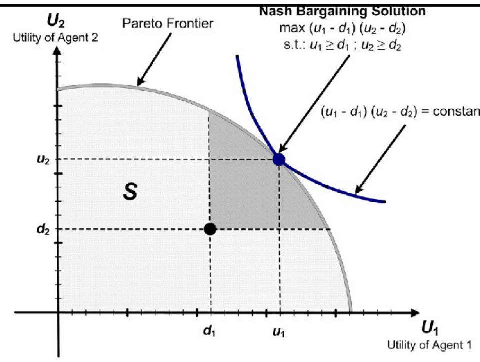
Utility = 0.5921

This is the econ 101 version…

Trade can repair some of the damage. The partially broken economies give a motivation for goods to move between regions. This is the hardest bit, though!

The simple, “econ 101” version looks like this: “A” can make juice but not butter. “B” can make butter but not juice. If they trade equal amounts, both are better off, by about one and a half percent..

POST-101 ECON



Trade provides a surplus. Who gets how much of that surplus?

- Economist answer: “bargaining power!”
- Nash bargaining solution: some point that maximizes **(u(x)-u(d))(u(y)-u(d))**, where **d** is the status quo.
- Other: Kalai-Smorodnisky, “Egalitarian”
- *None* of these models have any consistent experimental support!

Computationally, how do we identify trades that provide a surplus?

But there are two problems, one theoretical and one practical.

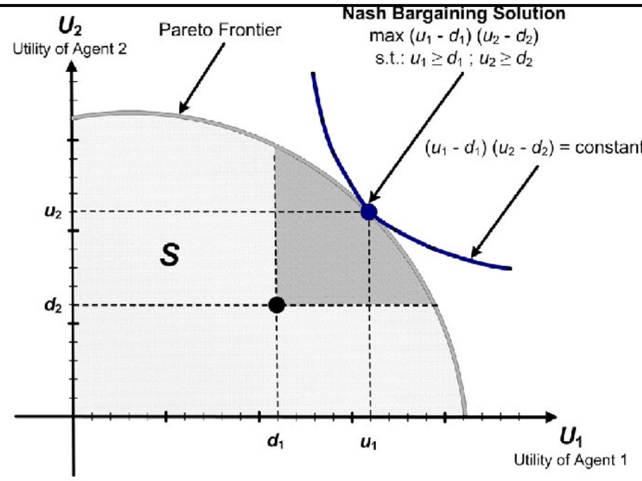
How do two negotiators come to an agreement? For example, how many units of apple butter for how many units of apple juice? Why should it be 1:1, just because it’s “fair”? Maybe region B should dig its heels in and demand a 10:9 ratio, which is still better than nothing for region A.

Economists have looked at this; John Nash proved that under “equal bargaining power” the solution should maximize the expression shown here. Other economists came up with other models, which disagree. The bad news is that studying real humans shows that they don’t behave in a way consistent with any of these models.

Practically, can we even identify all the opportunities for trade? Is there an efficient way of doing so?

(Classical economics says “no”, that’s why we need the free market.)

POST-101 ECON



Trade, how does it work?

- **Who** gets **how much** of the surplus?
- *None* of the standard bargaining models from economics have any consistent experimental support!

Computationally, how do we identify trades that provide a surplus?

But there are two problems, one theoretical and one practical.

How do two negotiators come to an agreement? Maybe A demands more than its fair share--- should B give in, because some benefit is better than none? Economist have developed several models – all of which disagree– and the bad news is that studying real humans shows that they don't behave in a way consistent with any of these models.

Practically, can we even identify all the opportunities for trade? Is there an efficient way of doing so?

AN AUCTION MECHANISM FOR TRADES

1. Proceed in rounds where one region is controlling the trading (like Catan!). Start with large lots, and move to smaller sizes.

2. Seller picks a good whose removal harms them least, and offer it.

3. Each other region makes an honest bid, the maximum they will offer of each of the other goods.

4. Seller picks the bid that increases their utility the most, or rejects if below a threshold.

Downside: lots and lots of optimization runs! Way too slow.

Alternative: pick trades randomly, use them if they are positive utility

Downside: fast, but weird trades

Here's an algorithm I came up with, that produces results that seem OK.
One region becomes the seller, and offers, say, 100 lemons.
Each other region makes a bid (honestly, ignoring their self-interest for now) of the maximum they will pay for those lemons, which might be nothing. They say, for example, 5 video games, 16 fruit pies, or 200 yarn.
The seller picks the bid that maximizes their utility after the trade, or rejects it if the gain is too small.

This works, but each round is very expensive because we run lots of gradient descents to figure out what the economy looks like when we add and remove some goods. It's really way too slow for use in a game.

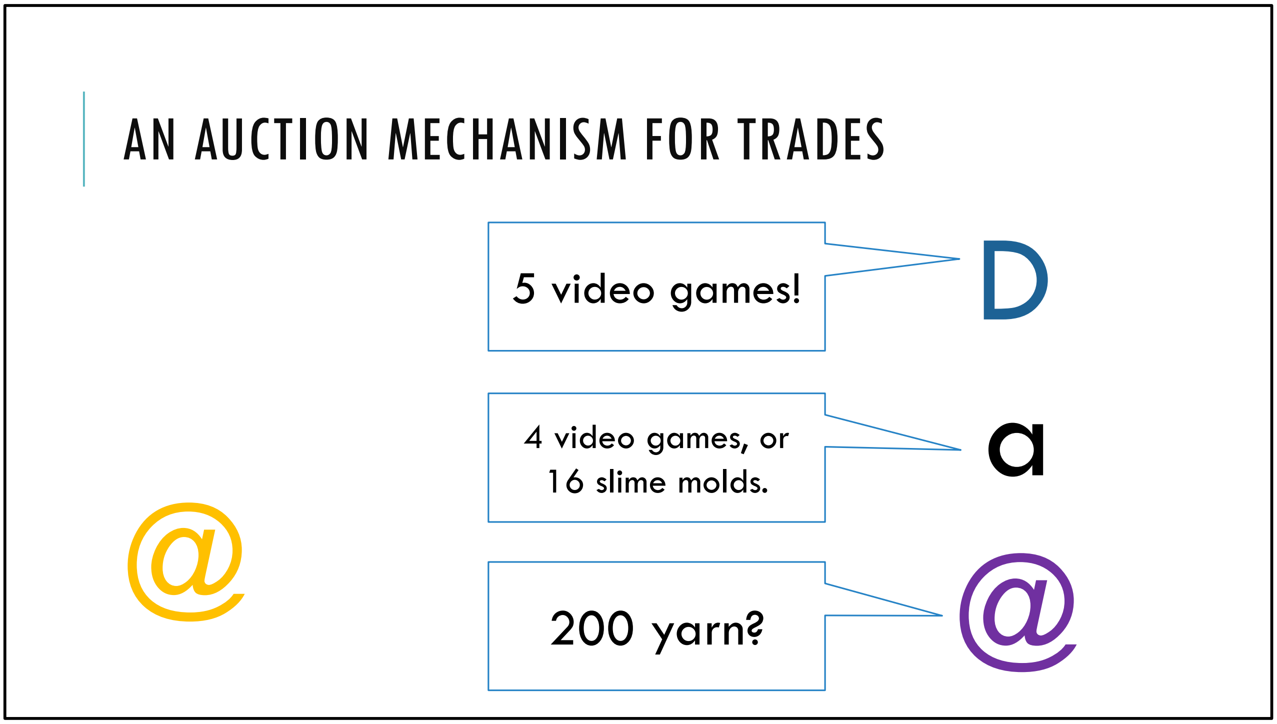
An alternative I played with is just generate random trades and if they're positive for both parties, run them. This produces sort of weird output, though.

AN AUCTION MECHANISM FOR TRADES

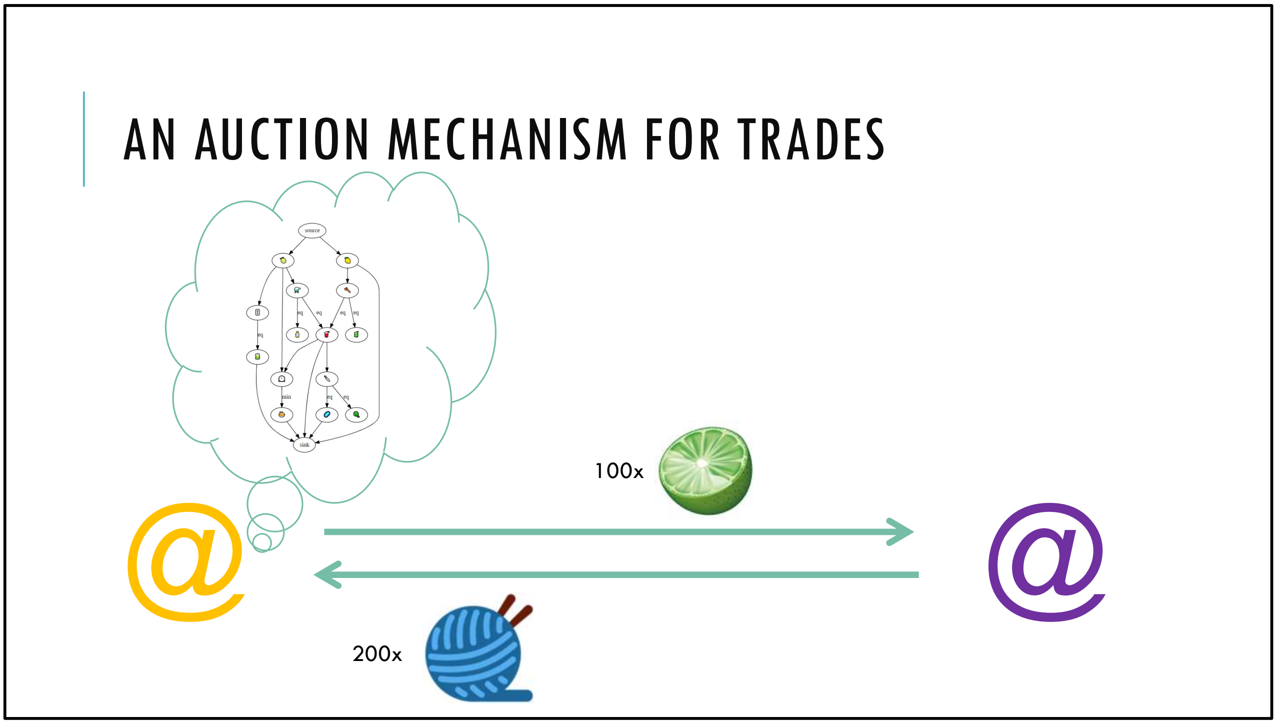
I have 100 limes!

@

Here’s an algorithm I came up with, that produces results that seem OK.
We proceed in rounds. One region becomes the seller, and offers, say, 100 limes



Each other region makes a bid (honestly) of the maximum they will pay for those limes, which might be nothing. They say, for example, 5 video games, 16 slime molds, or 200 yarn.



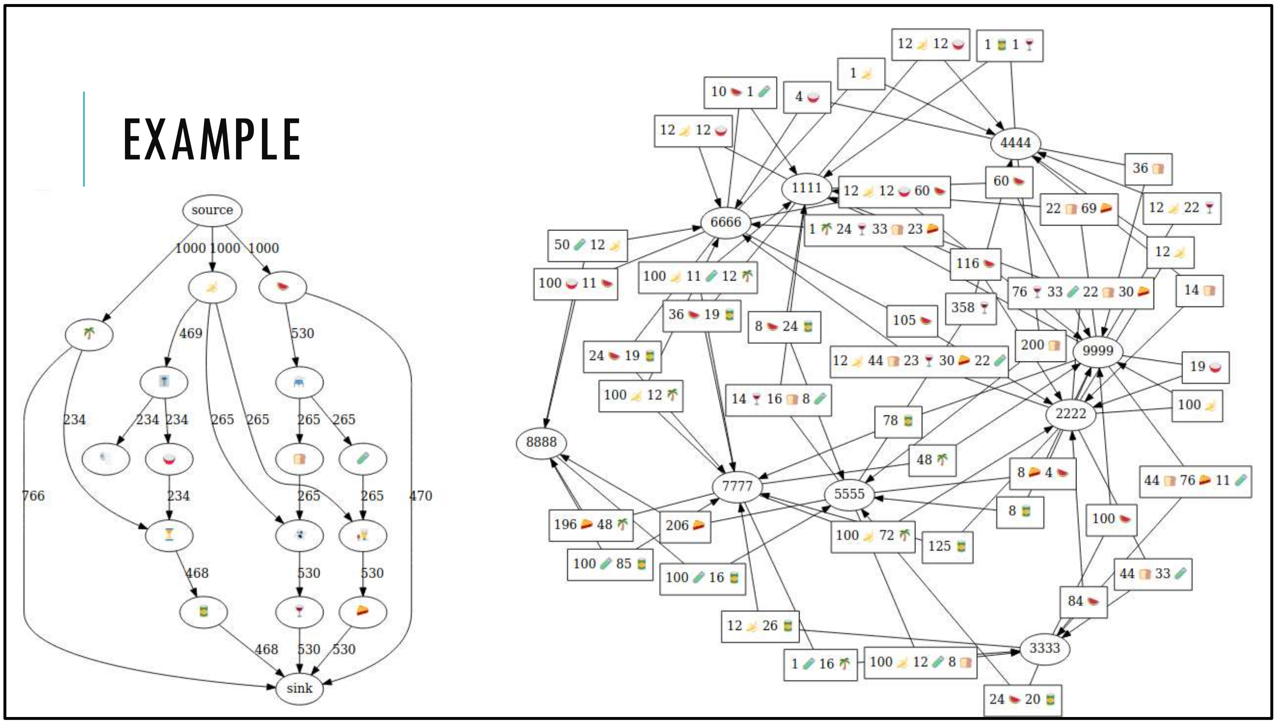
The seller picks the bid that maximizes their utility after the trade, or rejects it if the gain is too small.

AN AUCTION MECHANISM FOR TRADES

Produces mostly-realistic trades, but… lots and lots of optimization runs! **Way too slow.**

Alternative: pick trades randomly, use them if they are positive utility

Downside: fast, but weird trades



Here’s an example run with the “perfect” economy on the left, and the trades on the right. I like that we have a good dense trade network, with a lot of goods involved.

All the trades are bidirectional, though– no triangle trades, though we know they occur in real life. I also ignored any cost of moving goods, or geographic considerations– those might make interesting things that develop more of a hub-and-spoke model.

THANK YOU!

On Twitter: `@markgritter`

On Github:

- `mgritter/soffit`
- `mgritter/emojiconomy`

In RL:

Ideas for future exploration:

- crafting systems
- have players “pathfind” for NPC trade
- procedurally-generated incremental games

The code is available on Github, if you have trouble running it please let me know; my documentation is currently very poor.
I wrote my own graph grammar implementation, called Soffit in homage to Tracery. It’s mainly failed on its design goals, so I’m looking forward to rewriting it– I’ll be giving a talk on it next weekend as part of Minnebar, my local unconference.
In real life I work for HashiCorp on their secrets management system, Vault.