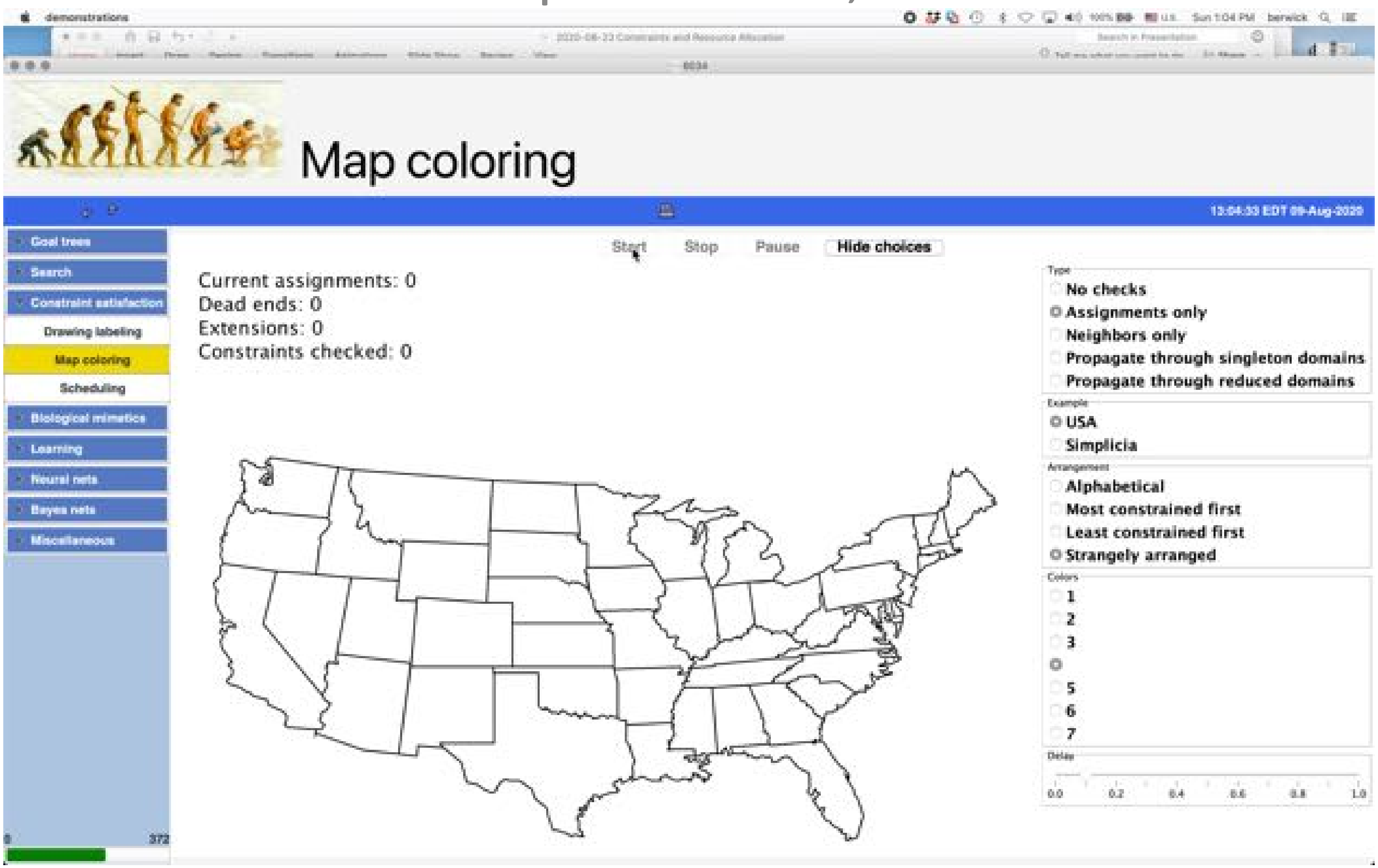
6.034 Artificial Intelligence: Lecture 6
Constraint Satisfaction

Professor Robert C. Berwick

September 16, 2020



AI Methods

- **Problem solving**
  - G+T, *search, optimal search, games, constraint satisfaction*
- **Inference**
  - *rule-based systems, Bayesian inference*
- **Machine learning**
  - *k-nearest neighbors, id trees, neural nets*, deep neural nets, *support vector machines*, genetic algorithms, near miss/one-shot
- **Communication, perception, action**
  - natural language processing, vision, robotics

Constraint satisfaction

- Search, exploiting constraints
  - Motivation for constraint satisfaction: Babies vs. Google
  - Constraint satisfaction & the trouble with Texas
  - The Domain Reduction Algorithm – what works best?
  - Propagation choices & Ordering choices for the Domain Reduction Algorithm
  - Applying constraint satisfaction to other problems

Constraint satisfaction: how does it fit in to our picture of intelligence?

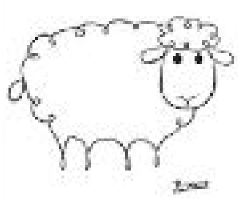
Constraints exposed by

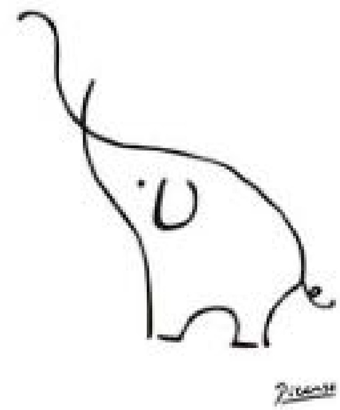
Representations that support

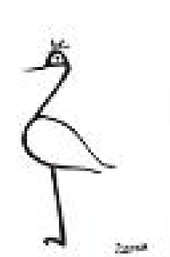
Models of perception, thinking, and action

- Right representation = problem almost solved
- More constraint, less search

Motivation
What babies can do easily







The American Journal of Psychology, Vol. 75, No. 4 (Dec., 1962), pp. 624-628

Picture perception in infancy

Judy S. DeLoache, Mark S. Strauss, Jane Maynard

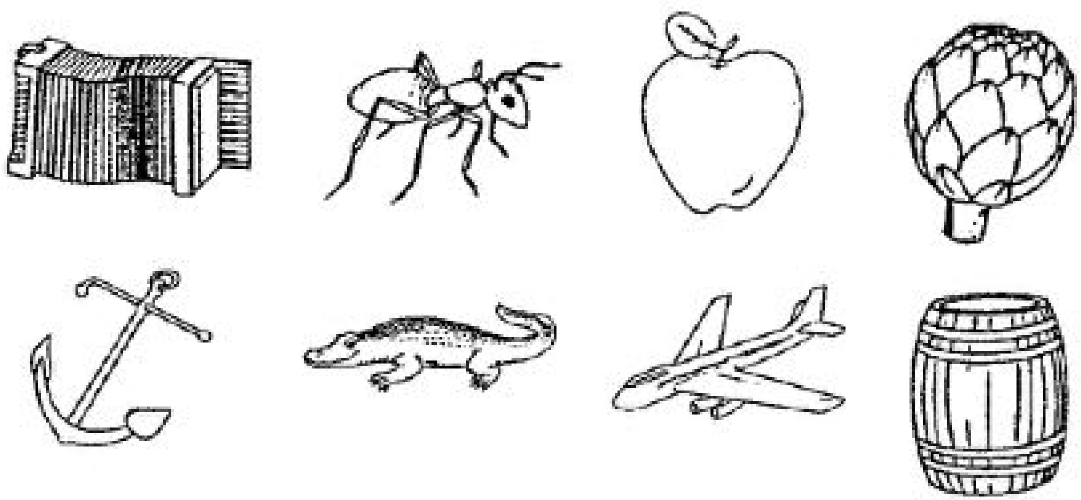
Infant Behavior and Development, Vol. 2, January 1979, 77-89

Slide Courtesy of Pawan Sinha

Google knows best?

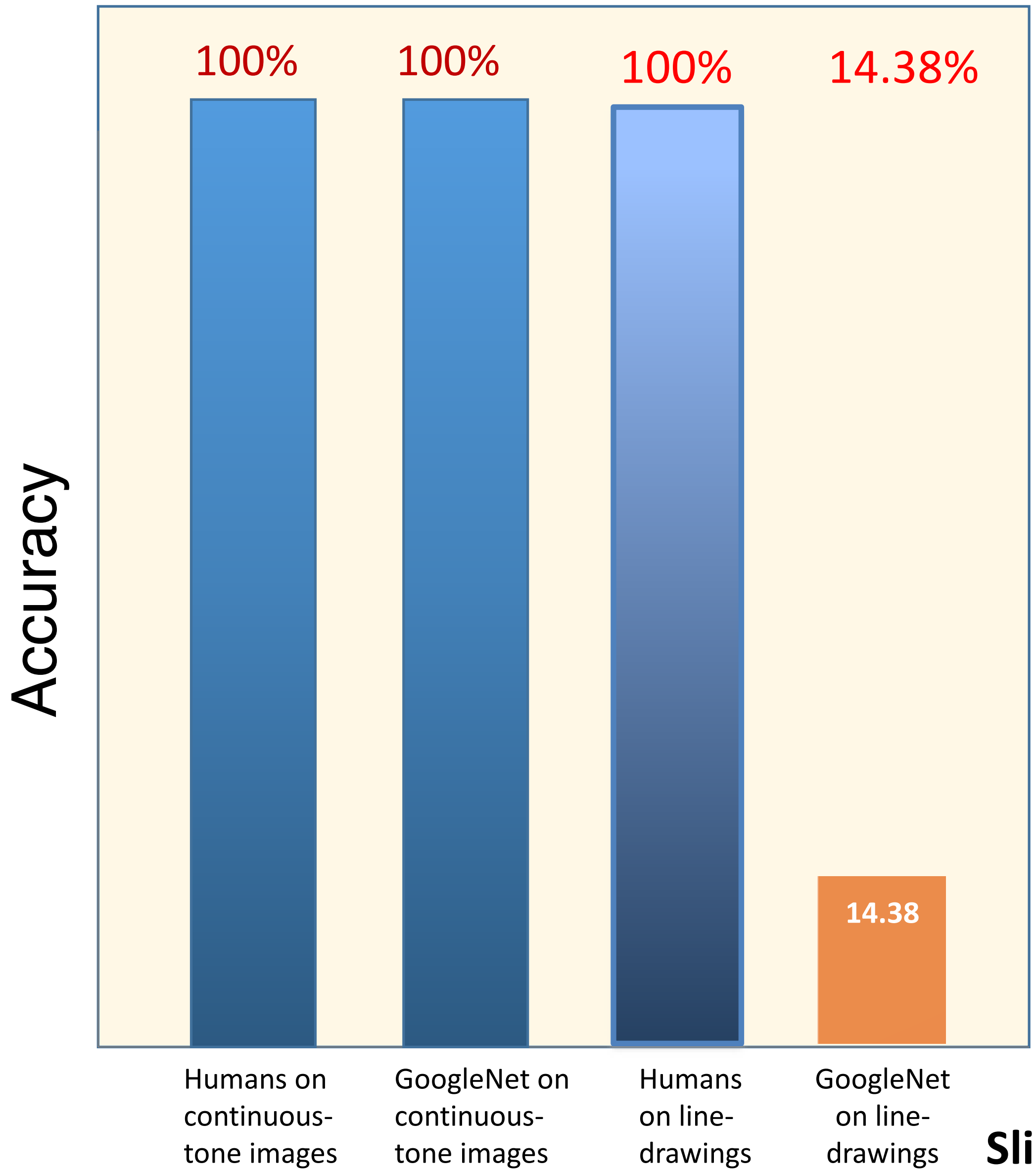
ImageNet – 14 million images

Line drawings of common objects in ImageNet



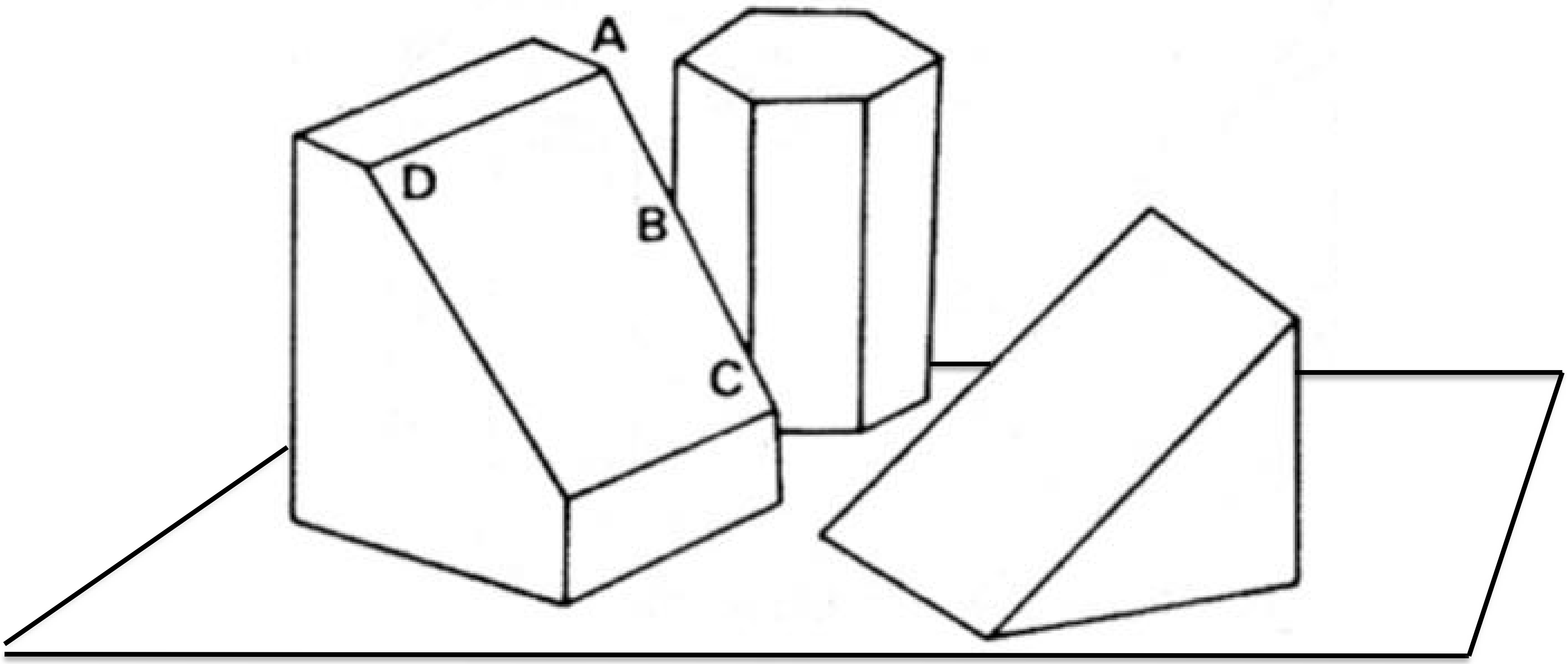
Slide courtesy of Pawan Sinha

Line drawings & pictures: Babies vs. Google



Slide courtesy of Pawan Sinha

2-D line drawings from 3-D scenes



Q: How do we figure out what these lines mean in terms of edges & junctions, and what are the separate blocks as projected from the real 3-D world?

Ans: To talk about this, first we need a library to describe the possible lines and junctions

How might one do this labeling?

- Lines in a drawing can meet up in a few different ways (these are the “constraints”)
- Places where lines meet up are called junctions
- Not all junctions are physically realizable

Let’s see how this works by <u>constraint</u> <u>satisfaction</u> aka <u>constraint</u> <u>propagation</u>

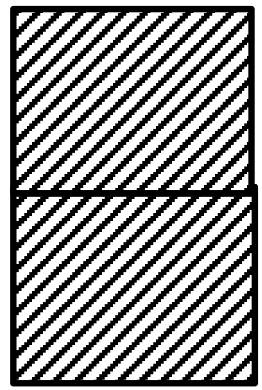
First, we need a description of lines and junctions between them…and three assumptions

The representation language or “combinatorial chemistry” for 3-vertex line drawings

Assumption 1: no “screw cases”

X

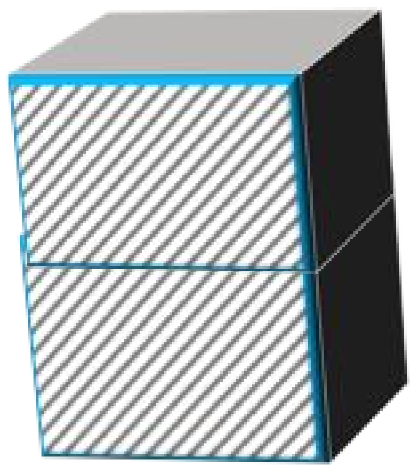
not allowed



ASSUMPTIONS

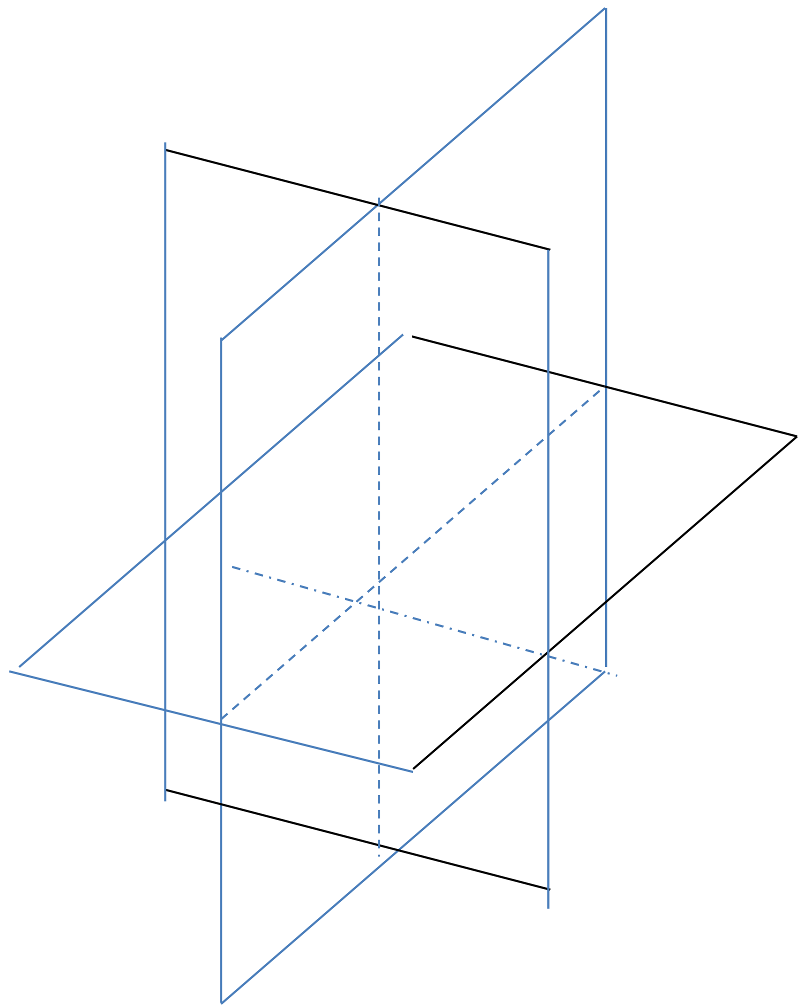
1. General position

OK



Real World	**2-D line drawings**
Vertices	**Junctions**
Edges	**Lines**

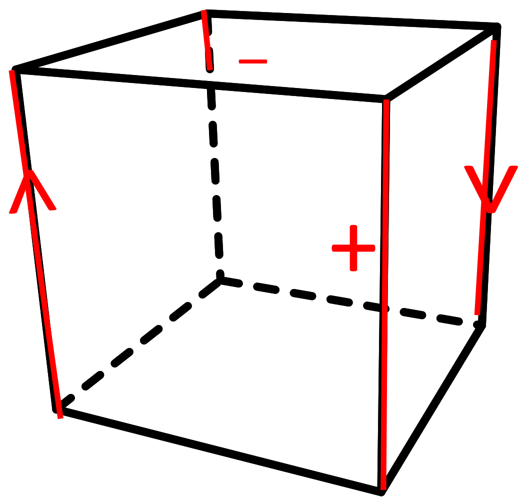
Assumption 2: 3-vertex (trihedral) junctions

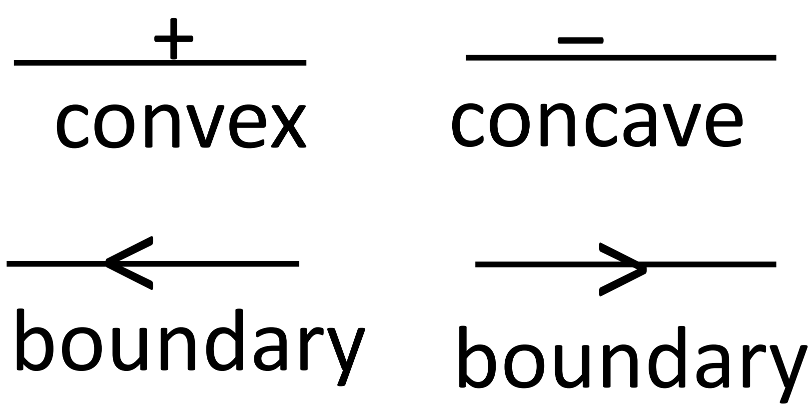


ASSUMPTIONS

1. General position
2. Trihedral vertices

   (all line junctions formed by intersection of 3 planes)



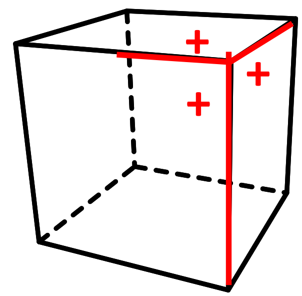


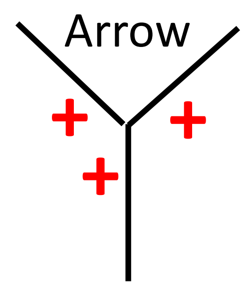
ASSUMPTIONS

1. General position
2. Trihedral vertices (all junctions formed by intersection of 3 planes)
3. Four line labels

Note: also by convention, we assume boundaries are traversed “clockwise”

Given these assumptions, there are only a few ways to label junctions of line drawings

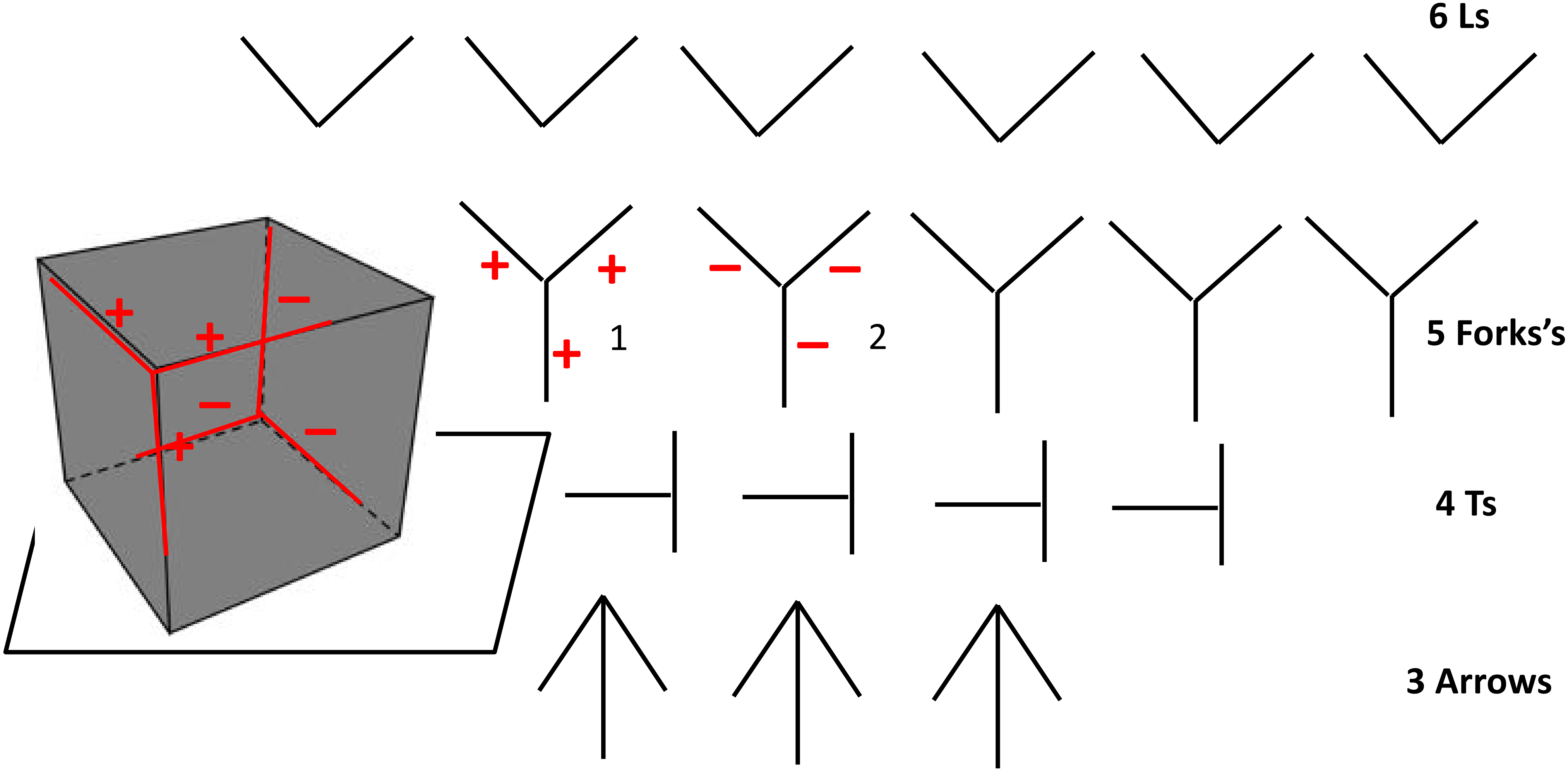




4. 18 atomic junction (vertex) types (where 2 or 3 lines meet), assuming 3-faced vertices:
   (i) 6 Ls
   (ii) 5 Forks
   (iii) 4 Ts
   (iv) 3 Arrows

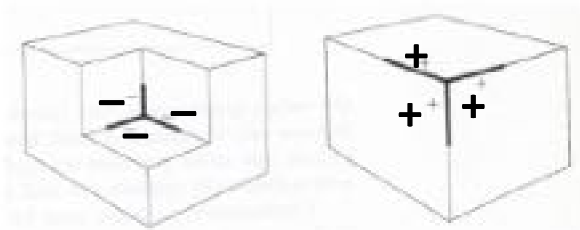
But where do these 18 primitive junction types come from?

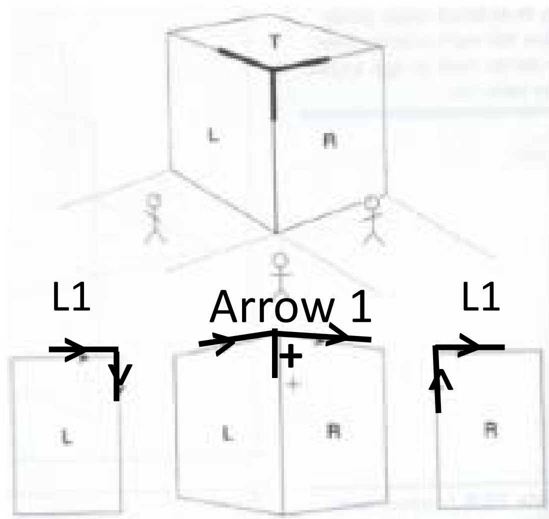
Vocabulary of junction types…come from difft views of objects

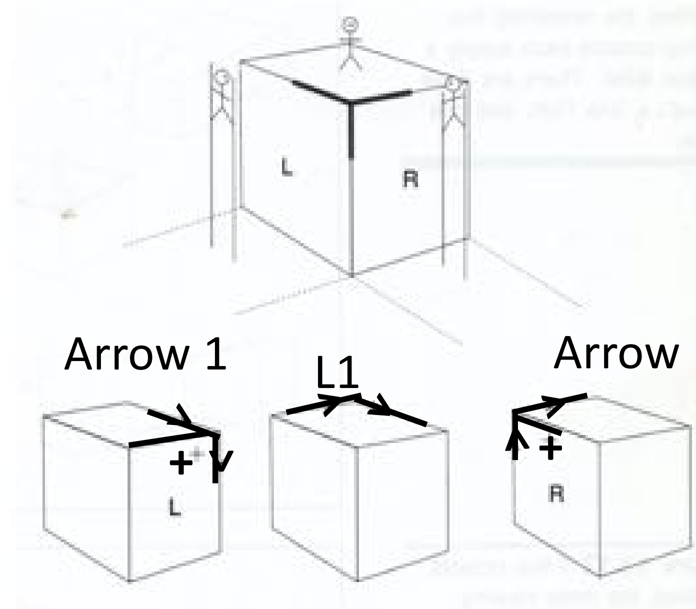


Here are the first two fork type junctions

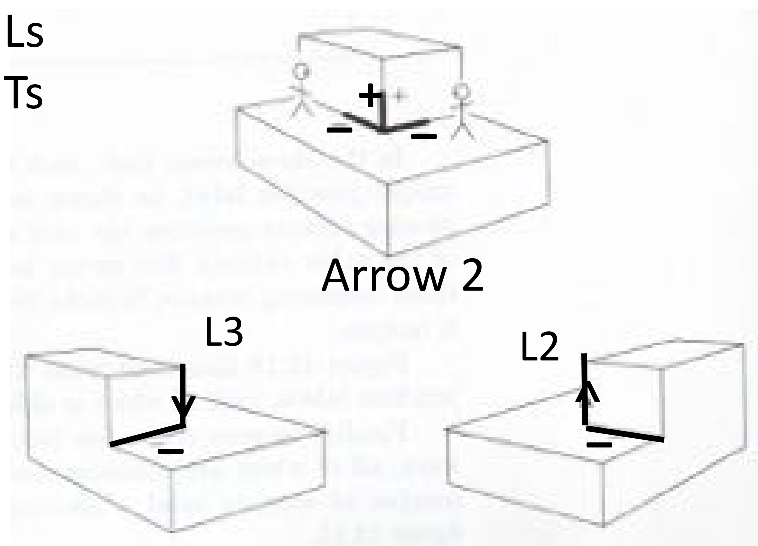
The full library of junction types – just 18 labels

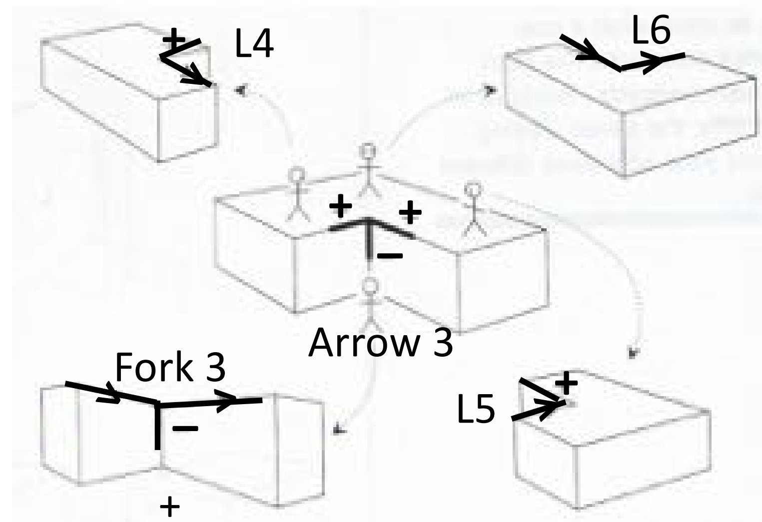






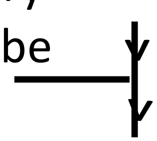
3 Arrows
5 Forks
6 Ls
4 Ts



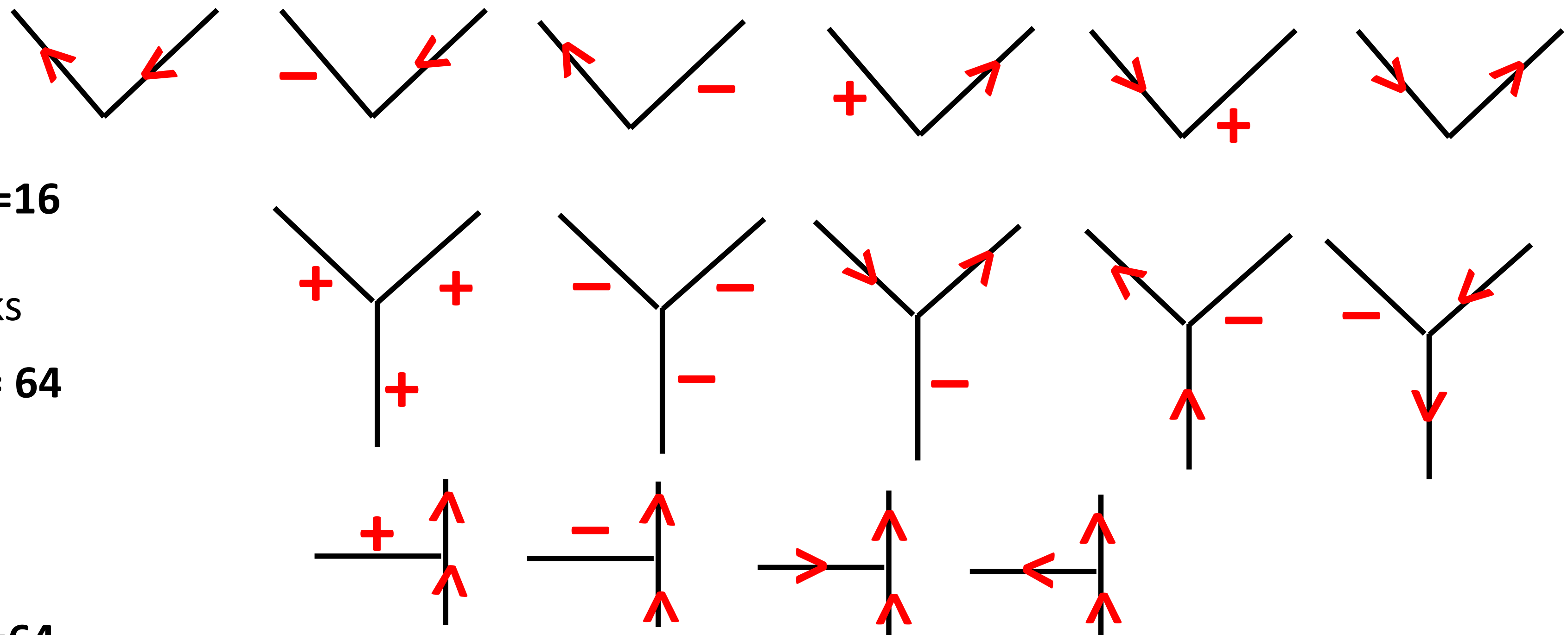


+
Rotations
= Forks 4, 5

And 4 T's, since the "T" is always a boundary line, and the other line hitting the "T" can be any of the other 4 labels, +, –, >, or >



The “periodic table” of 18 “elements” that can label trihedral junctions



L’s

4^2 =16

6 L’s

Forks

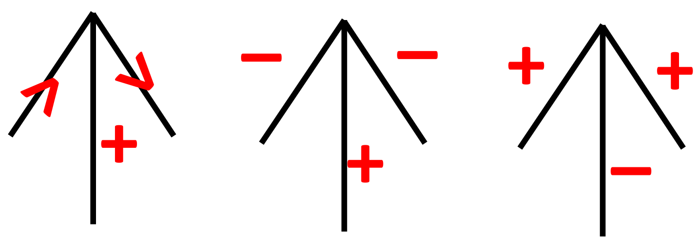
4^3 = 64

5 F’s

T’s

4^3 =64

4 T’s



Arrow 1 Arrow 2 Arrow 3

3 A’s

Arrows

4^3 = 64

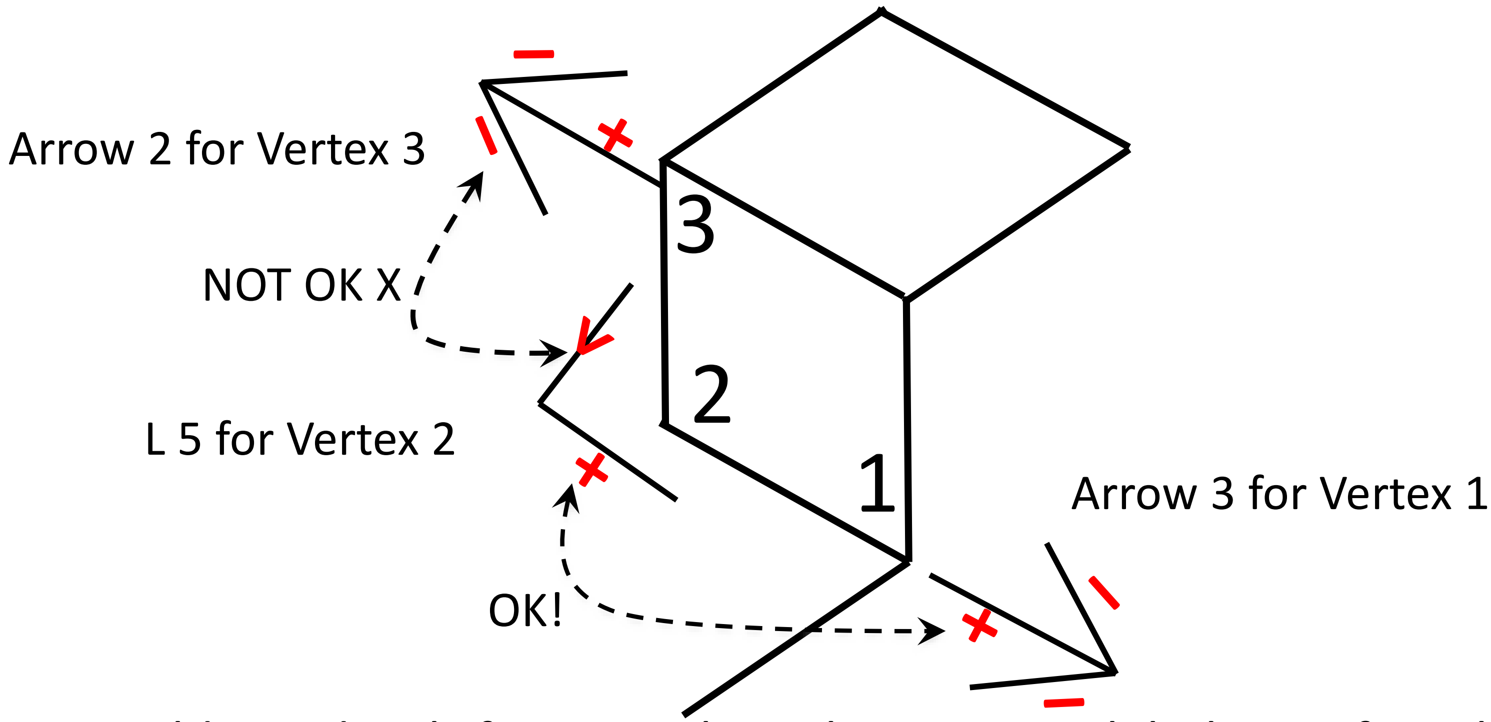
Q: How many junction types <u>logically possible</u> if no real world 3-vertex constraint?

Total: 208

18 physically realizable out of 208 logically possible – constraints

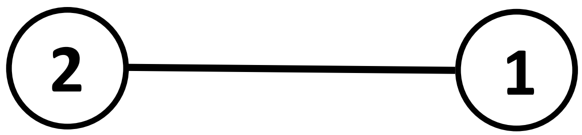
Now let’s see how to do “constraint satisfaction” using these constraints

Given a 2-D line drawing, how do we label it with these junctions? DFS?

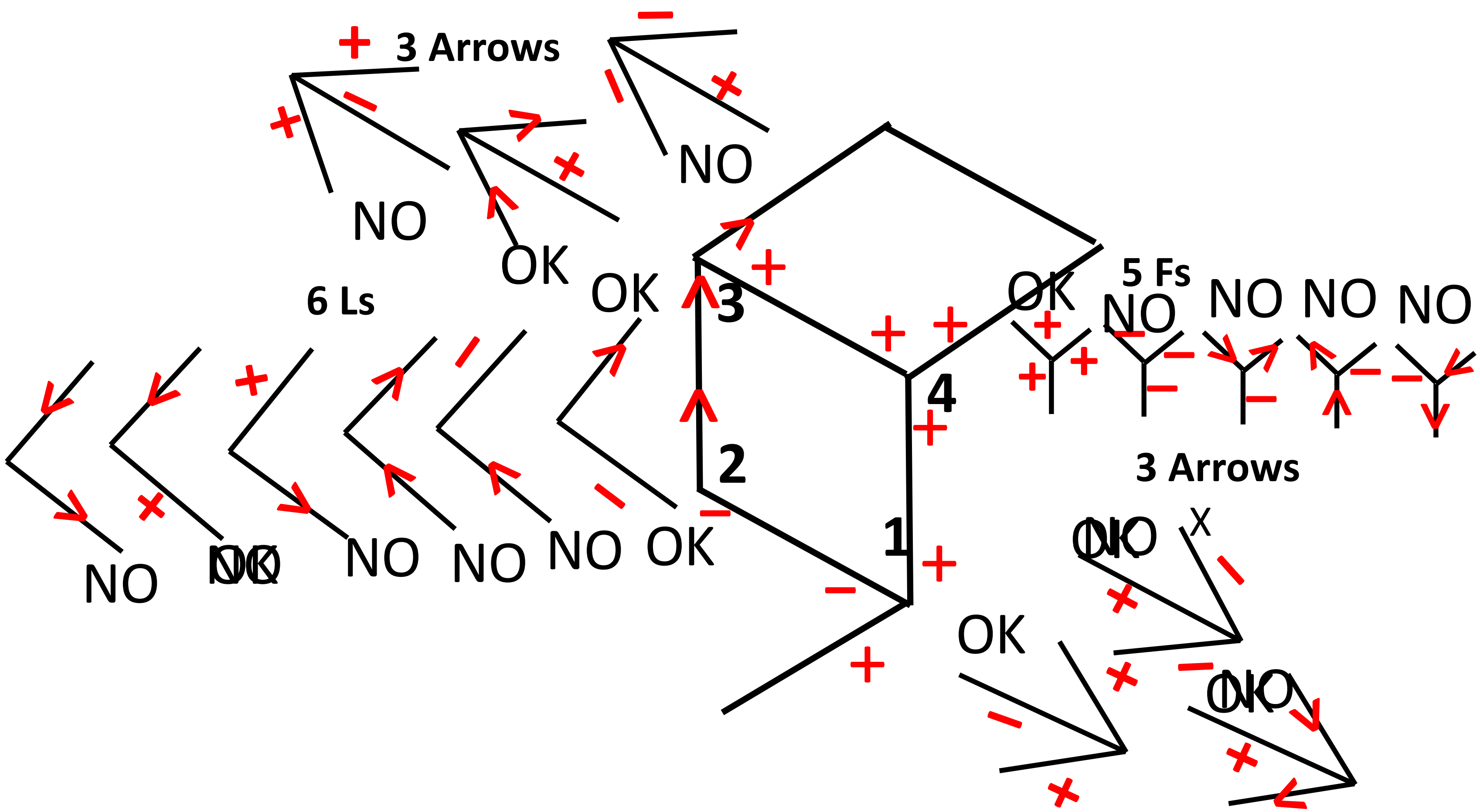


- We could use depth-first search: pick a junction label out of our library of 18, for some vertex, #1; then picking a junction label for vertex #2, and so on, backing up if the labels are not “compatible”
- E.g., pick Arrow 3 for Vertex #1, and then L5 for Vertex #2. The line connecting vertex 1 to vertex 2 must be marked “+” at both ends: **this is the key constraint that must ”propagate” & defines the notion “compatible”**
- But this choice is not compatible with Arrow 2 for Vertex #3, because the < on L5 ≠ the – on Arrow 2; we need to back up & make another choice of junction label

Key idea: instead, we can "pile up" <u>all</u> the labels at each junction and filter them out one against the other all at once ("Waltz's algorithm")

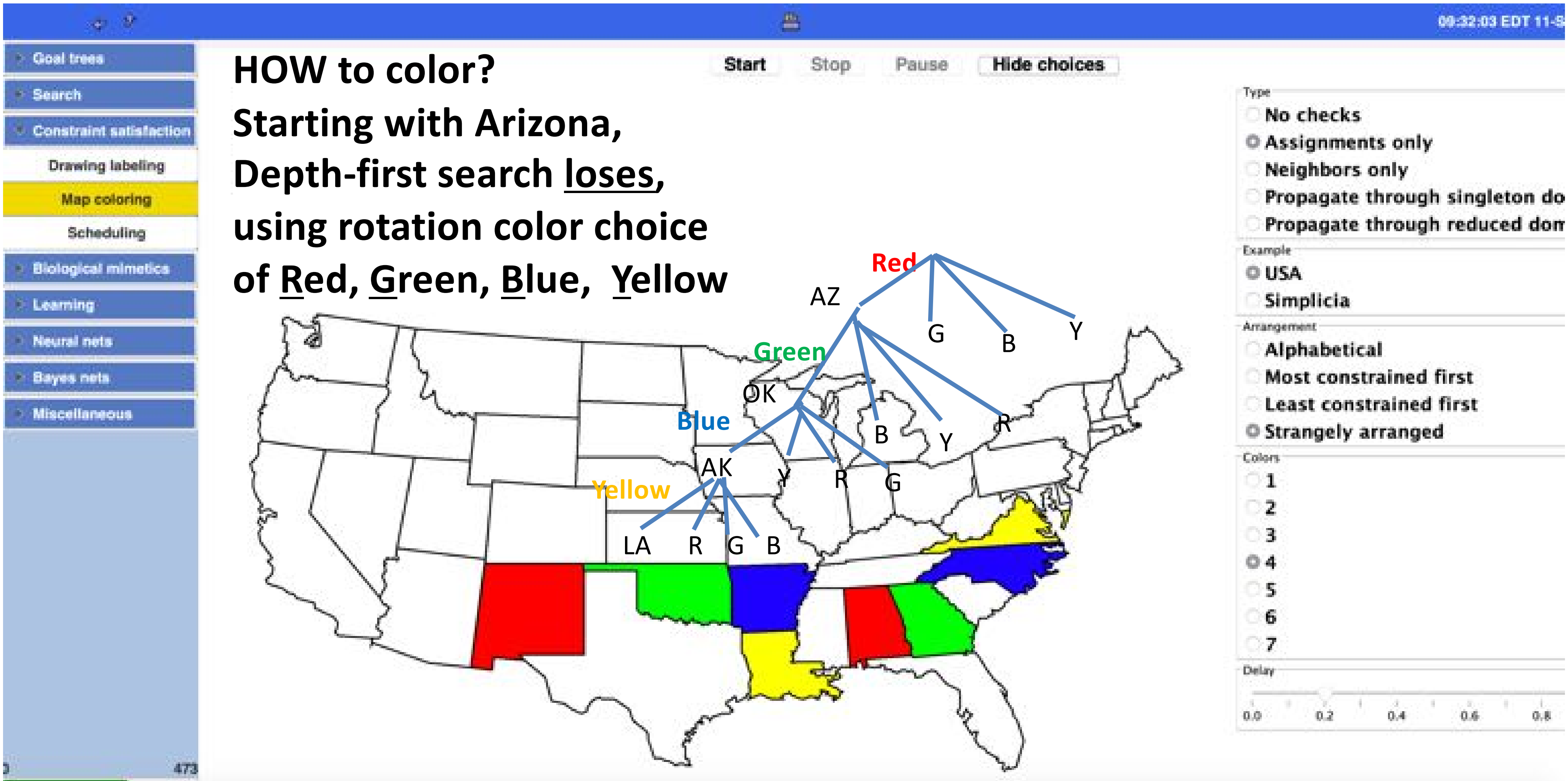


Constraint: labels @ line joining vertices must match!
(constraint "propagates" from one vertex to another)



Constraint satisfaction for map coloring: No adjacent states w same color

4 colors – Red, Green, Blue, Yellow



Will constraint propagation as in line labeling work out here?

{Red, Green, Blue, Yellow}	{Red, Green, Blue, Yellow}
{Red, Green, Blue, Yellow}	{Red, Green, Blue, Yellow}

We need to do something a bit more general…

Vocabulary for a general method: the Domain Reduction

- Variable V: something that can have an assignment
- Value x: something that can be assigned
- Domain D: a bag of values
- Constraint C: a condition that must be satisfied among variable values

Systematic Idea for Map Coloring: *Domain Reduction Algorithm*

- For each depth first search assignment
  - For each variable V_i *considered*

    we have choices here

    - For each value x_i in D_i (domain of V_i)
      - For each constraint C between V_i and other variables V_j

        we use binary constraints (e.g., Y/N)

Domain Reduction Algorithm

        - If $\nexists x_j \in D_j$ such that $C(x_i, x_j)$ is satisfied
        - Then remove x_i from D_i

Systematic Idea for Map Coloring: *Domain Reduction Algorithm*

- For each depth first search assignment
  - For each variable V_i <u>*considered*</u>

    we have choices here

    - For each value x_i in D_i (domain of V_i)
      - For each constraint C between V_i and other variables V_j

        we use binary constraints (e.g., Y/N)

        - If $\nexists x_j \in D_j$ such that $C(x_i, x_j)$ is satisfied
        - Then remove x_i from D_i
        - If D_i empty, then backtrack

Domain Reduction Algorithm

Let’s see domain reduction in action with map coloring – we can “consider” variables in several different ways…

1. Check nothing (no constraints); or everything
2. Check assignments only
3. Check neighbors
4. Propagate constraints through reduced domains
5. Propagate through singleton domains

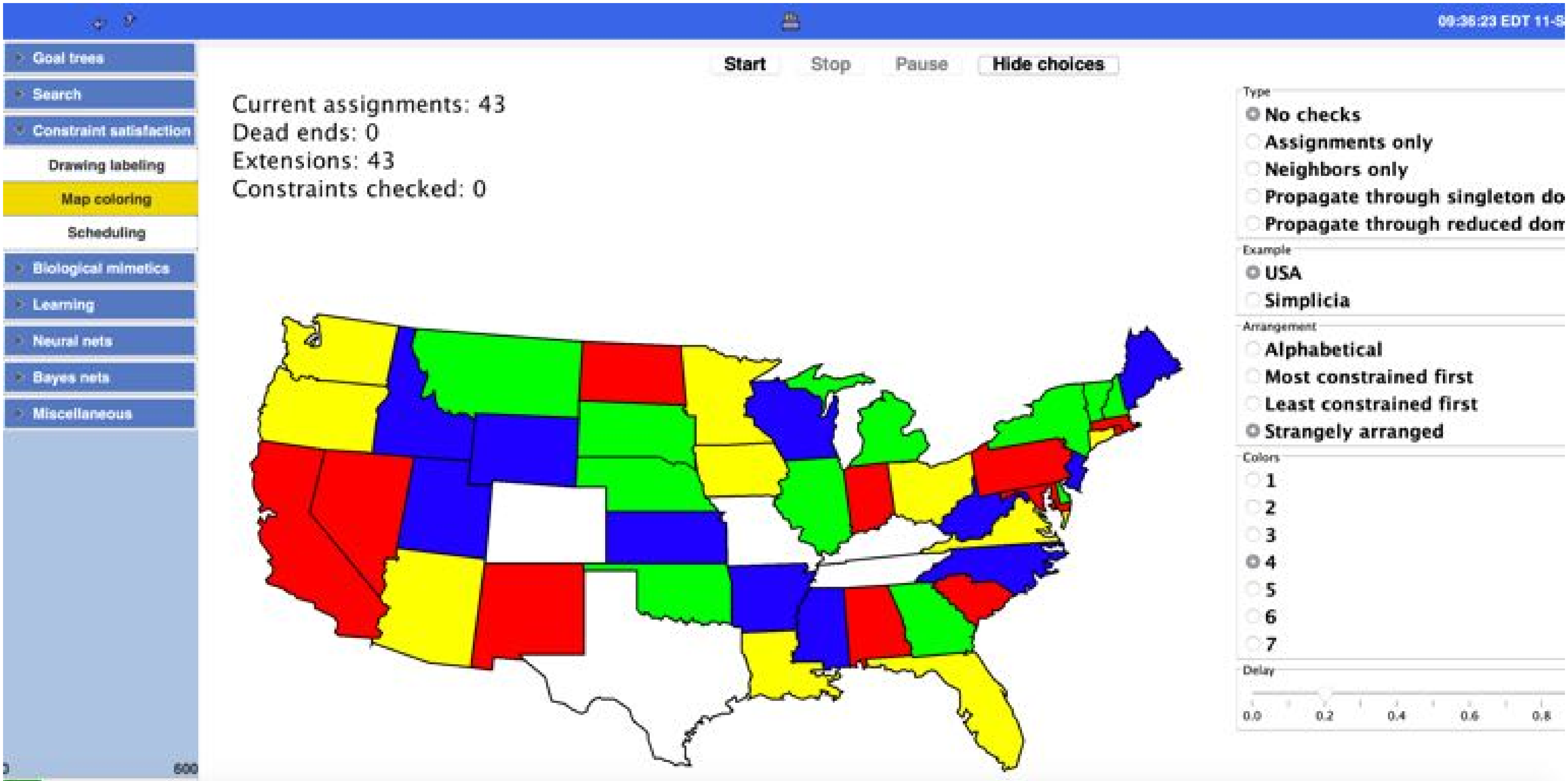
In addition, 2 flourishes:

Try most constrained variables (e.g., states) first;

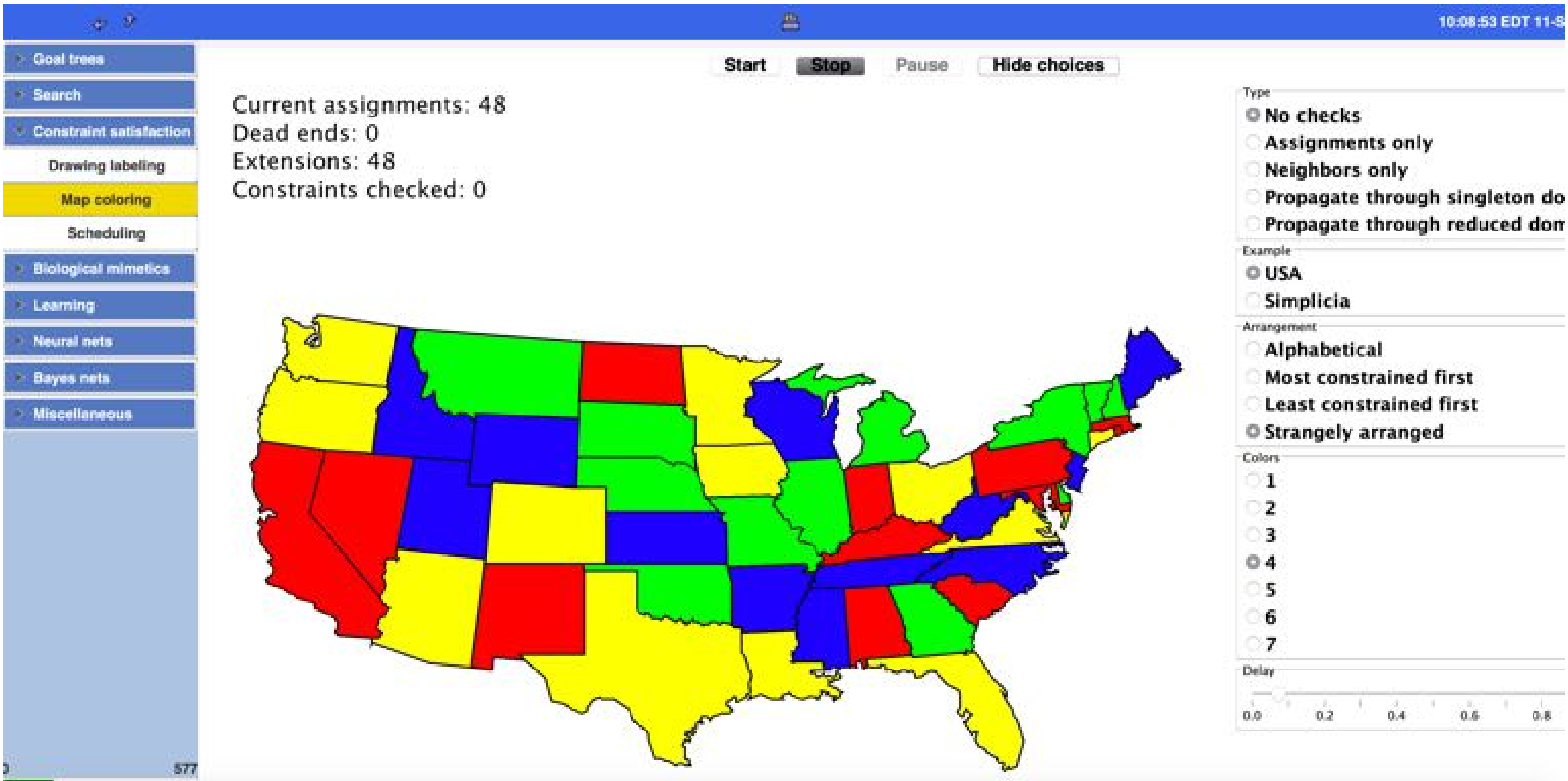
Try least constrained variables (e.g., states) first

Which method works best for USA map coloring?

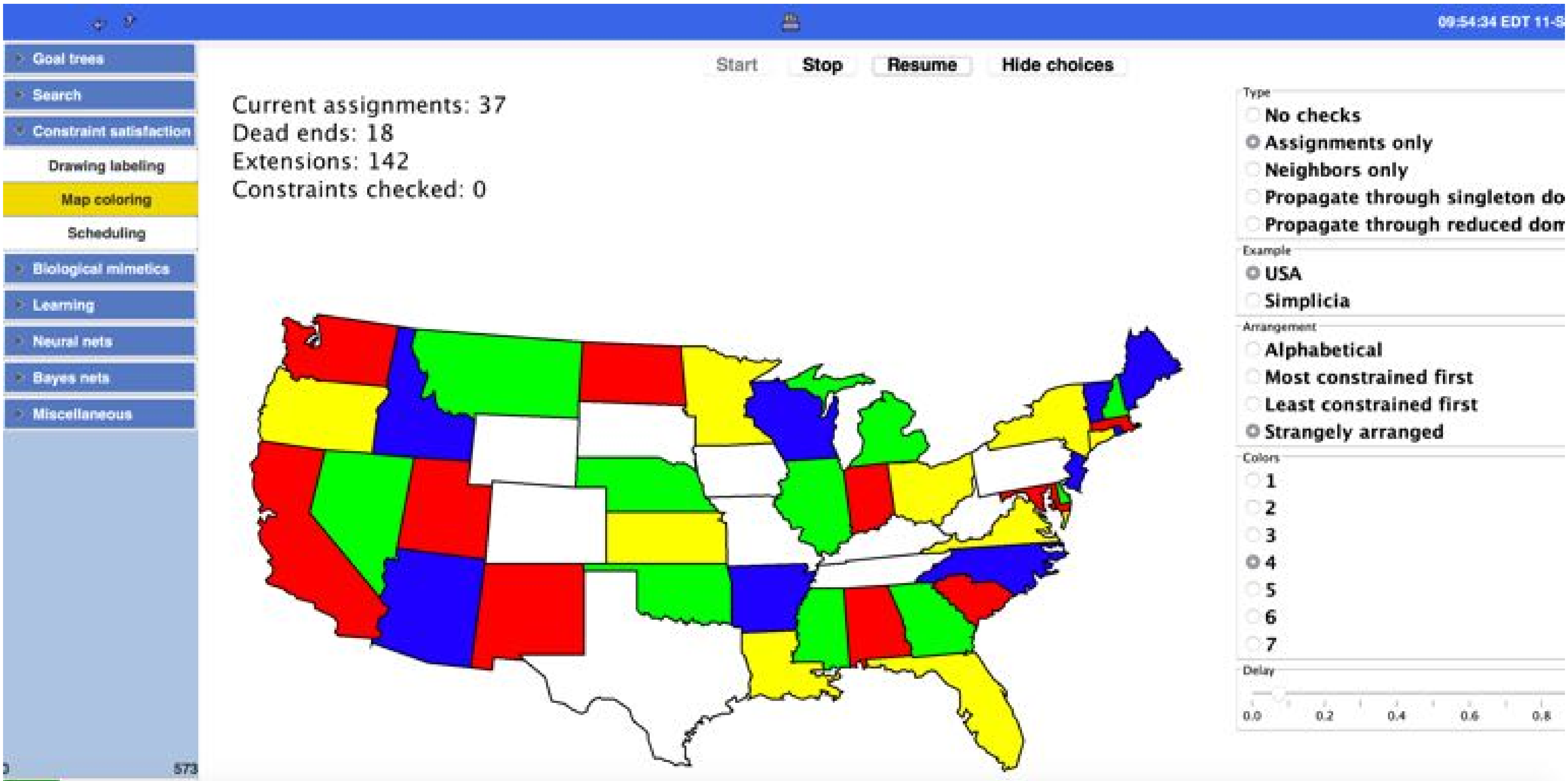
1. Check nothing



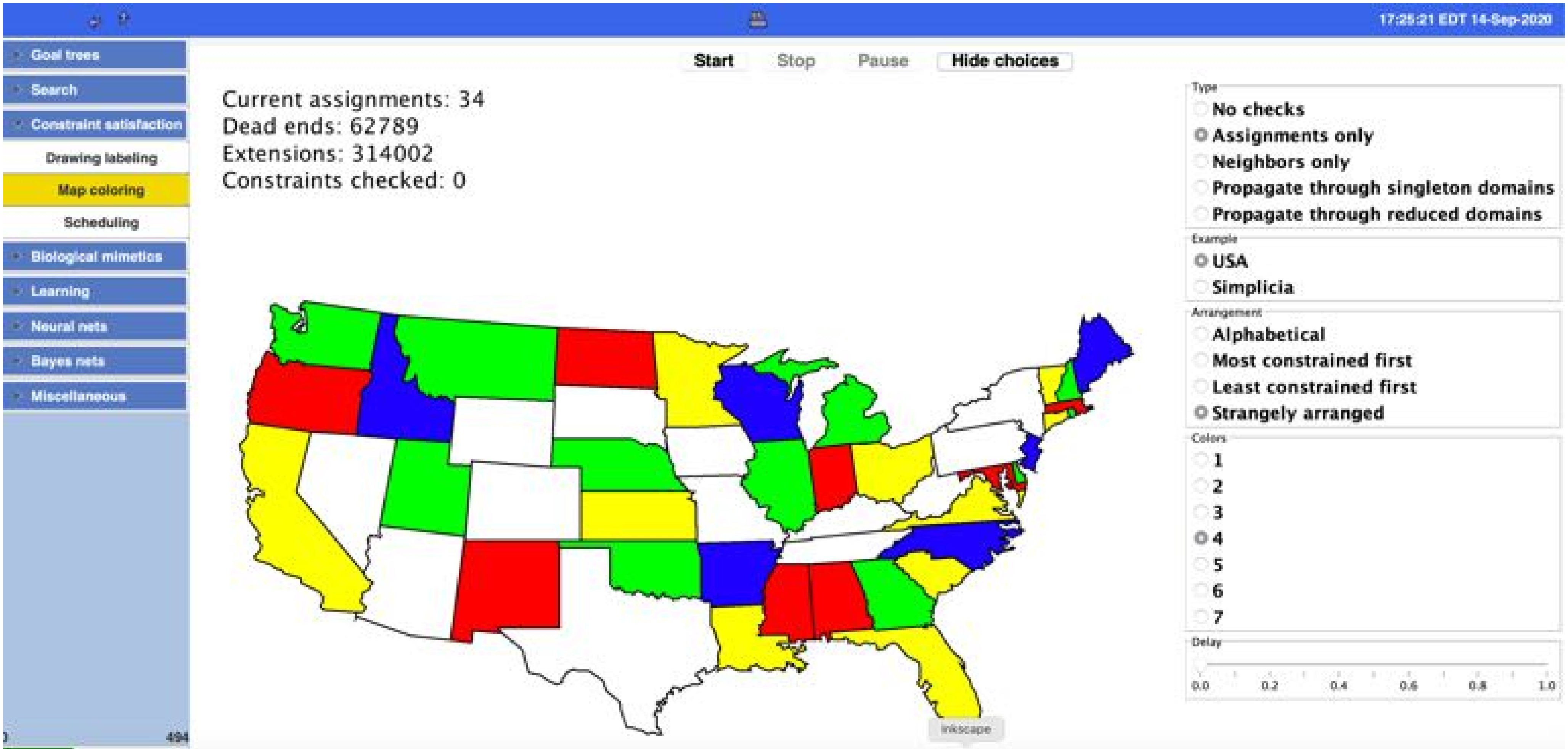
1. Check nothing – end state



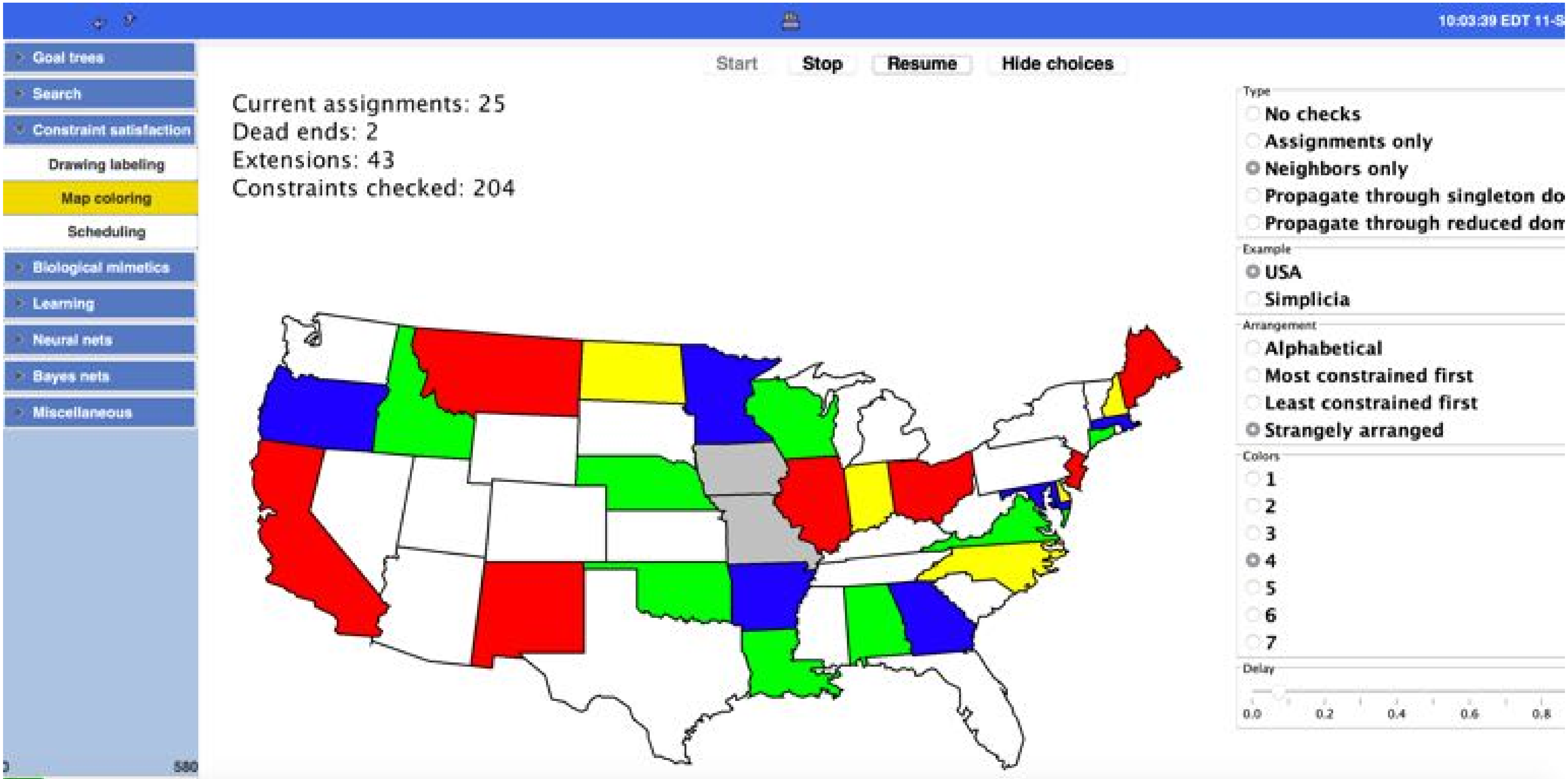
2. Check assignments only



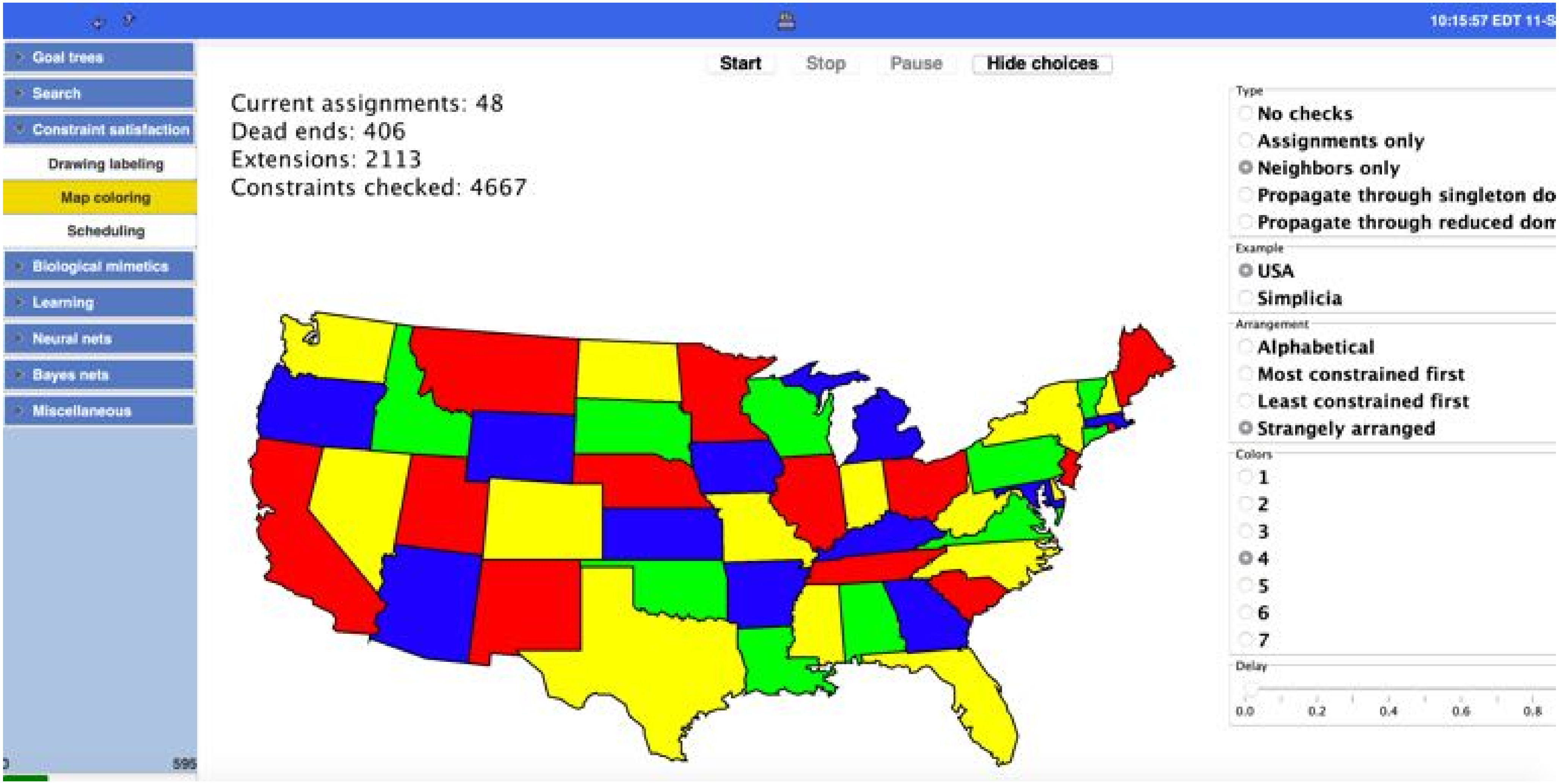
2. Check assignments only – end state



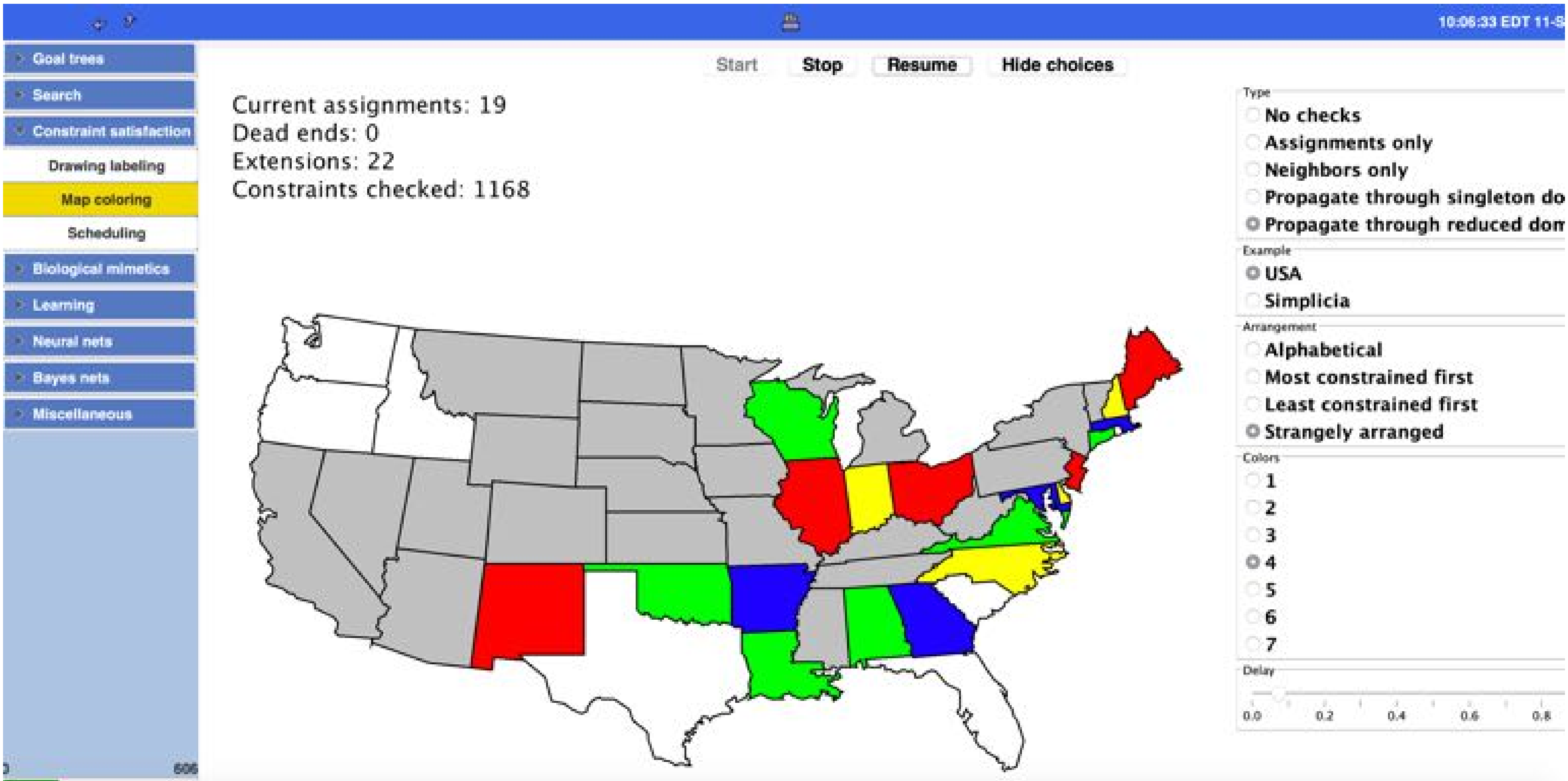
3. Check neighbors only



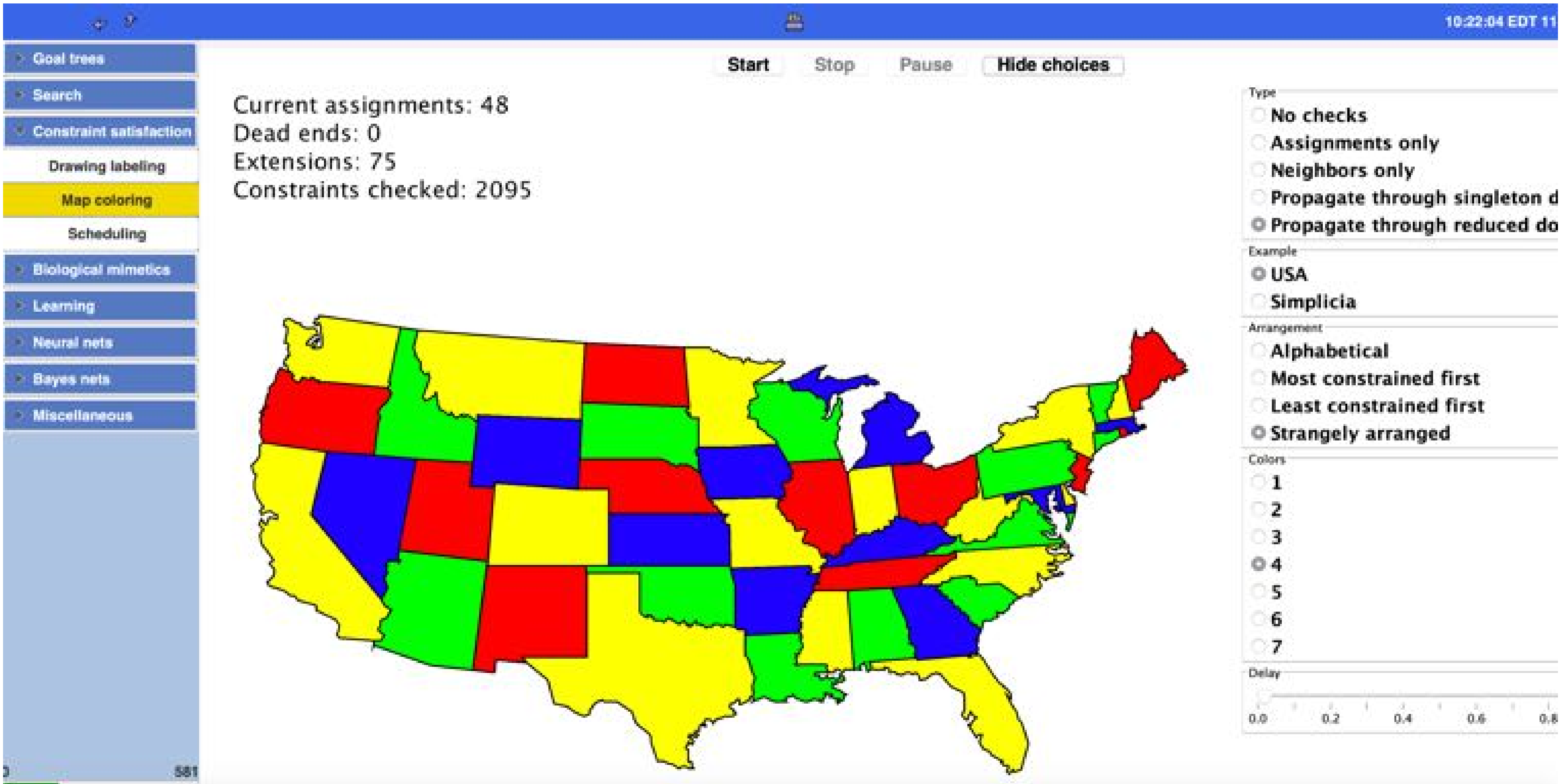
3. Check neighbors only – end state



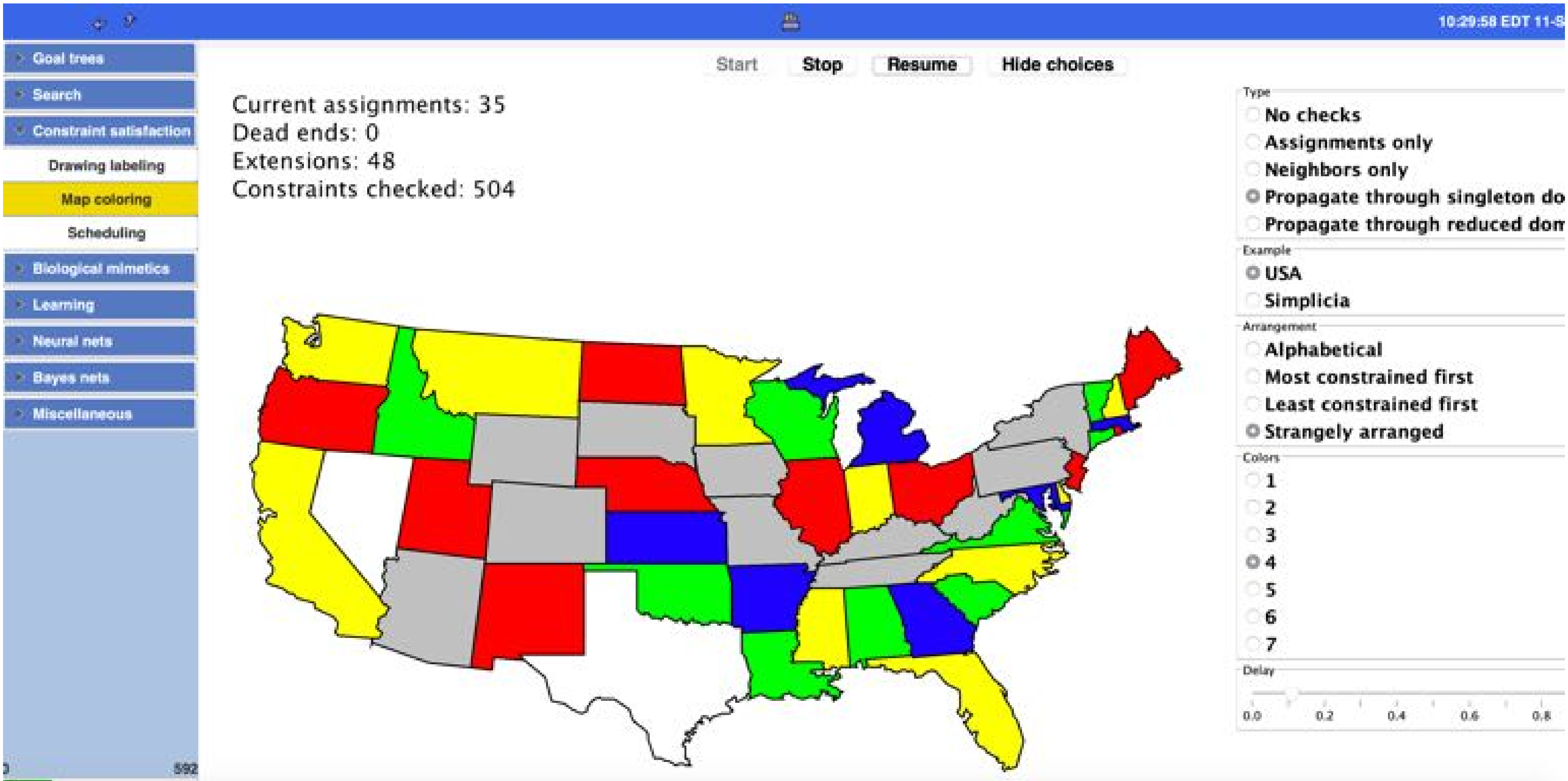
4. Propagate constraints through reduced domains



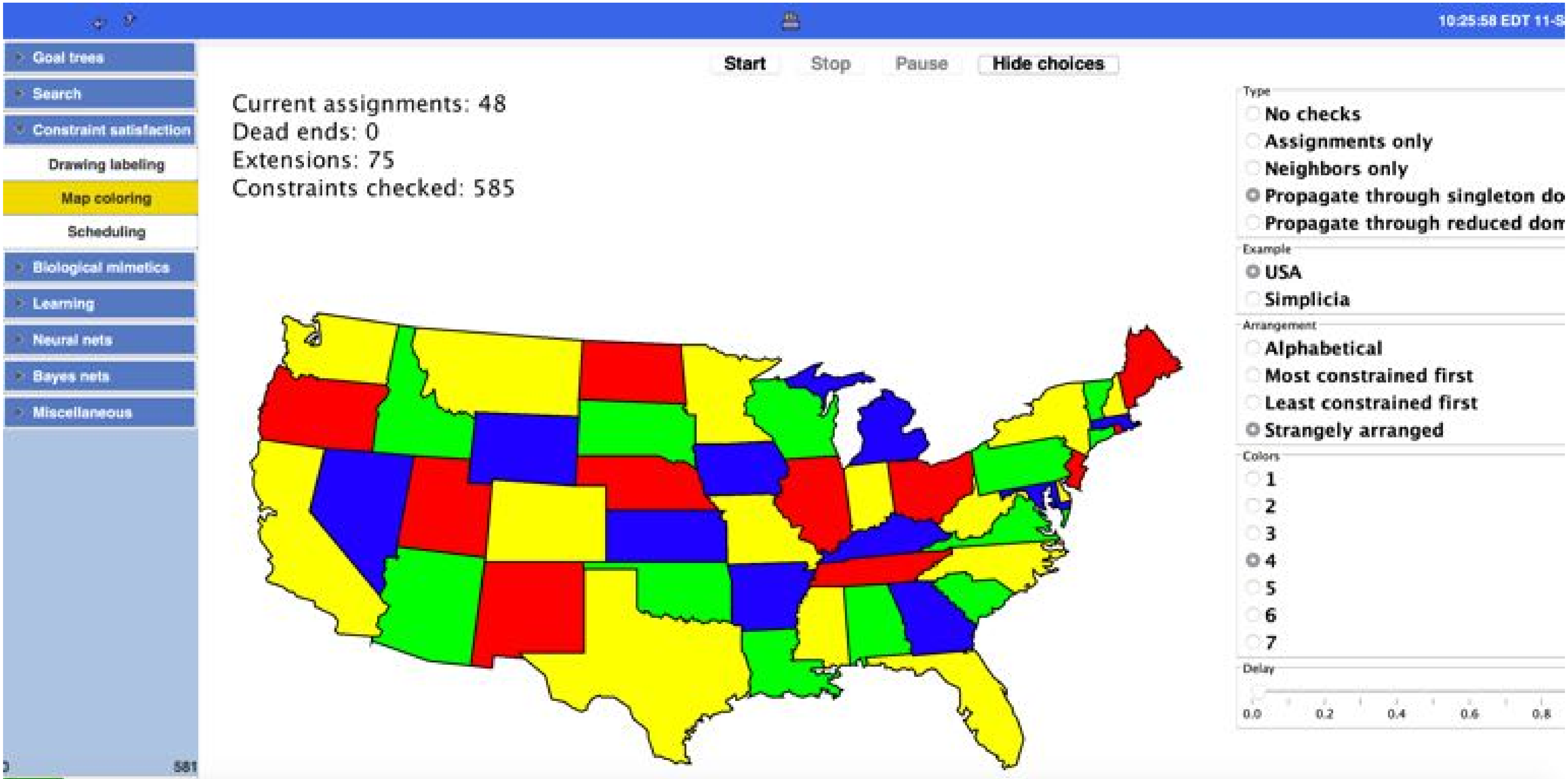
4. Propagate through reduced domains – end state



4. Propagate through singleton domains

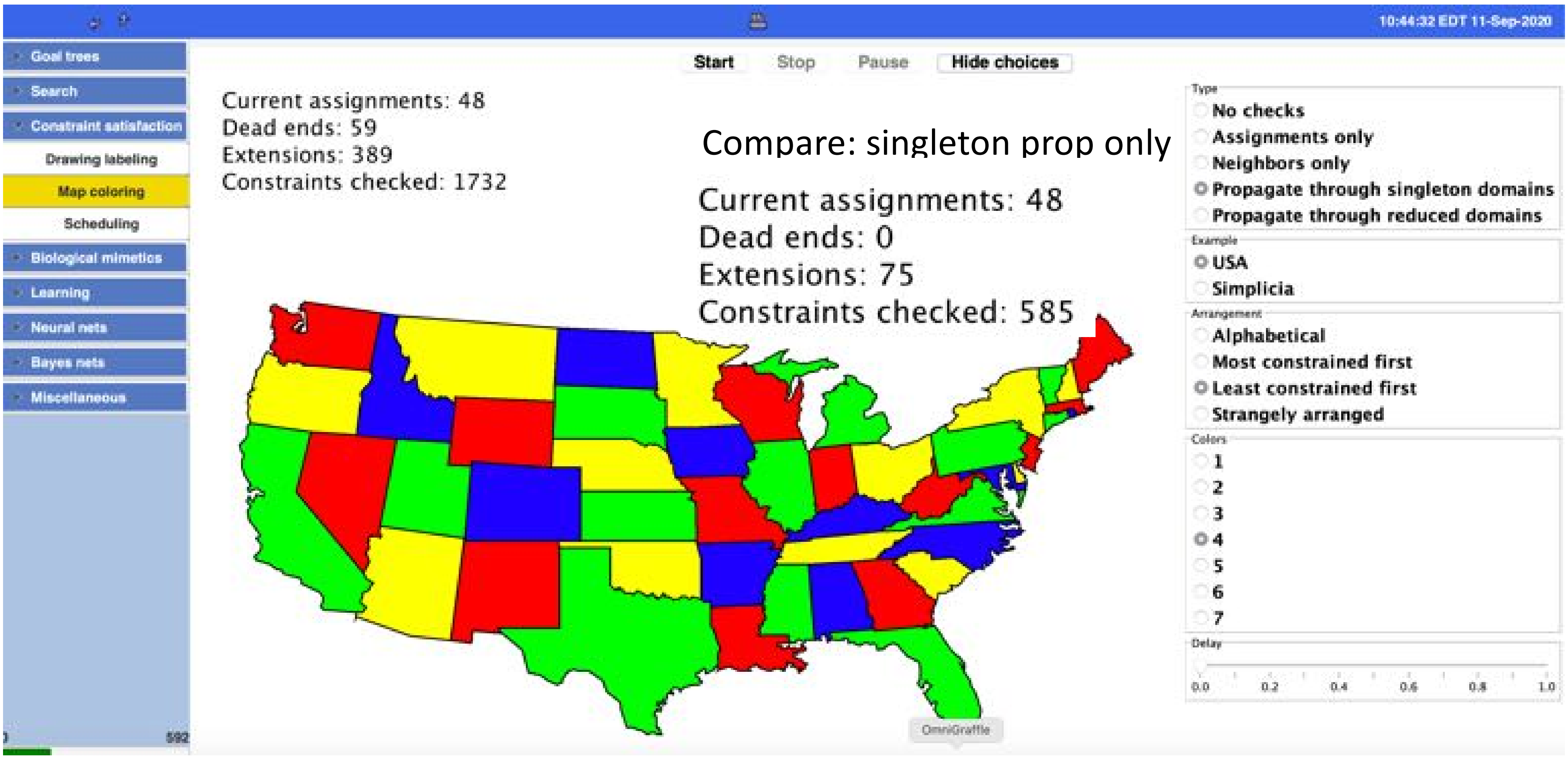


5. Propagate constraints through singleton domains – end state

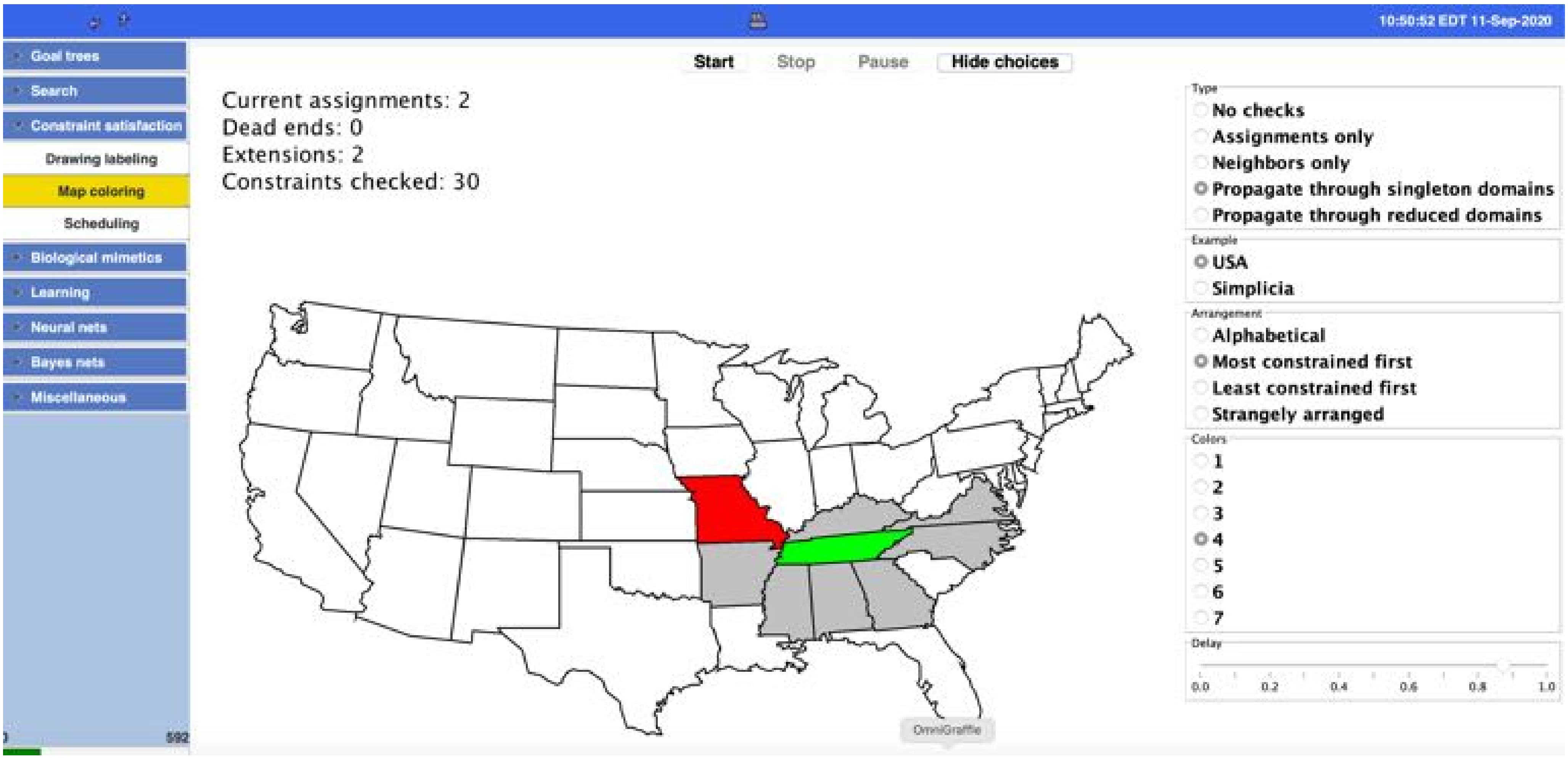


Flourish 1: color least constrained state first

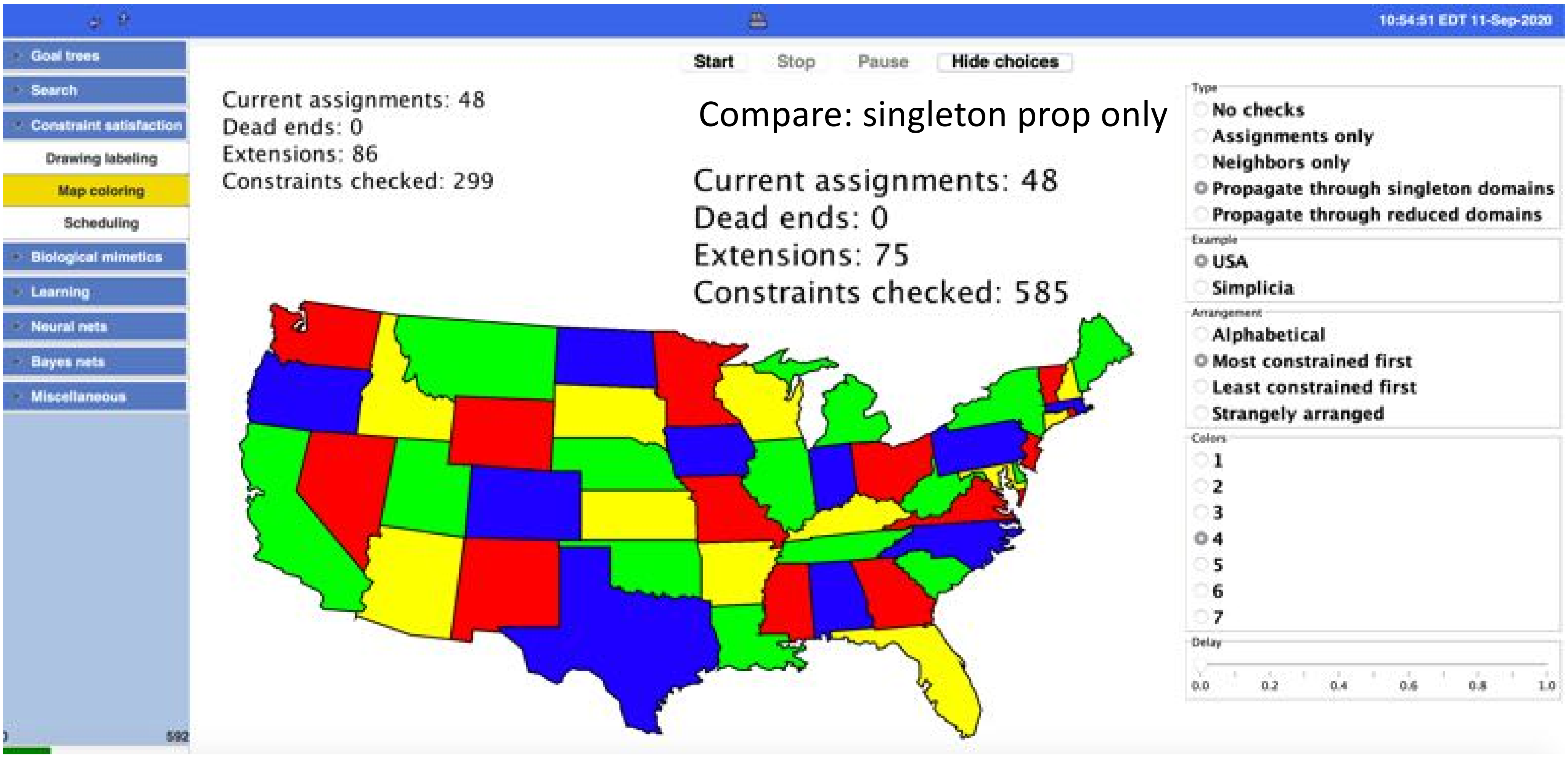
Flourish 1: color least constrained state first – end state



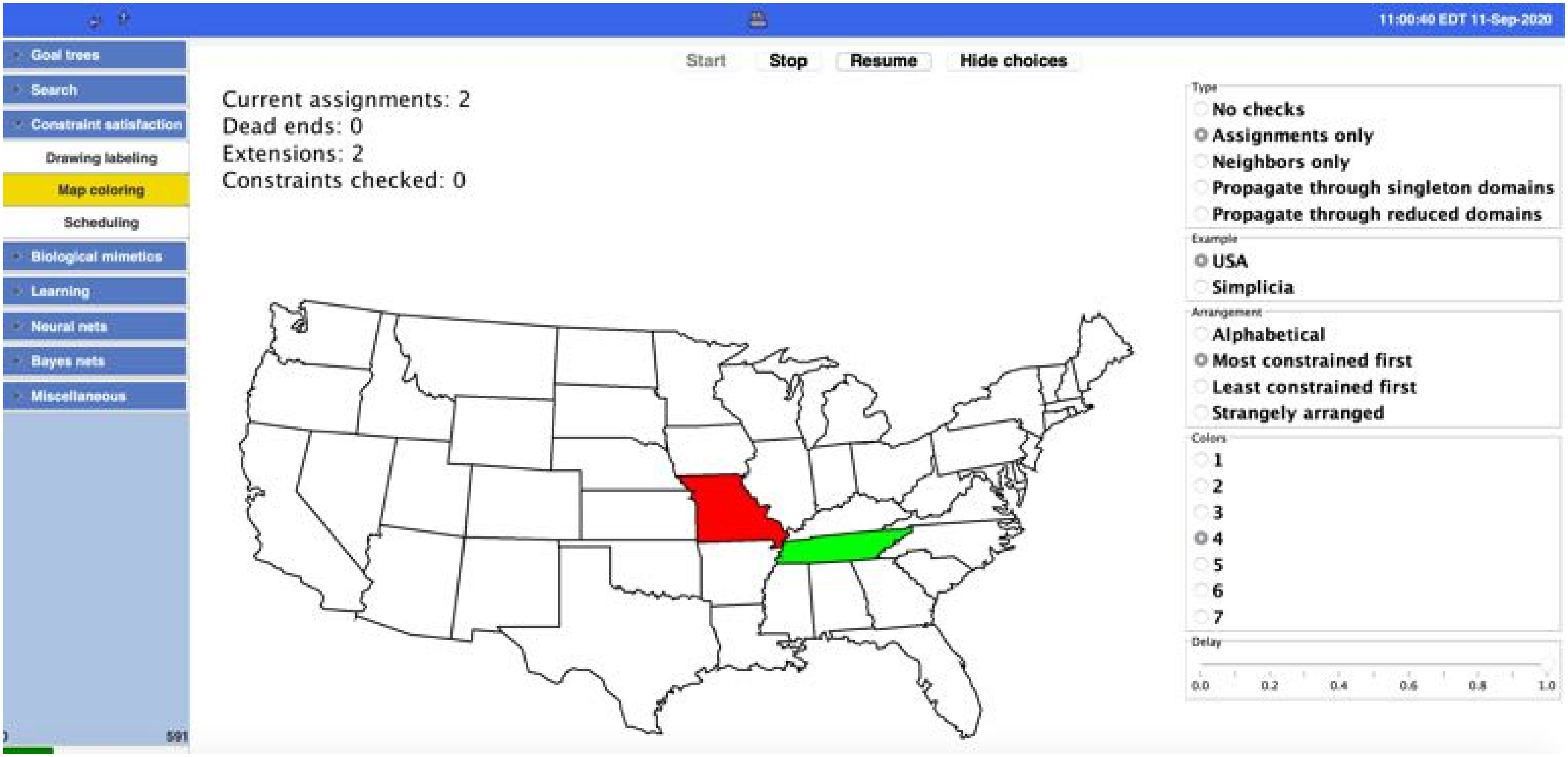
Flourish 2: color most constrained state first



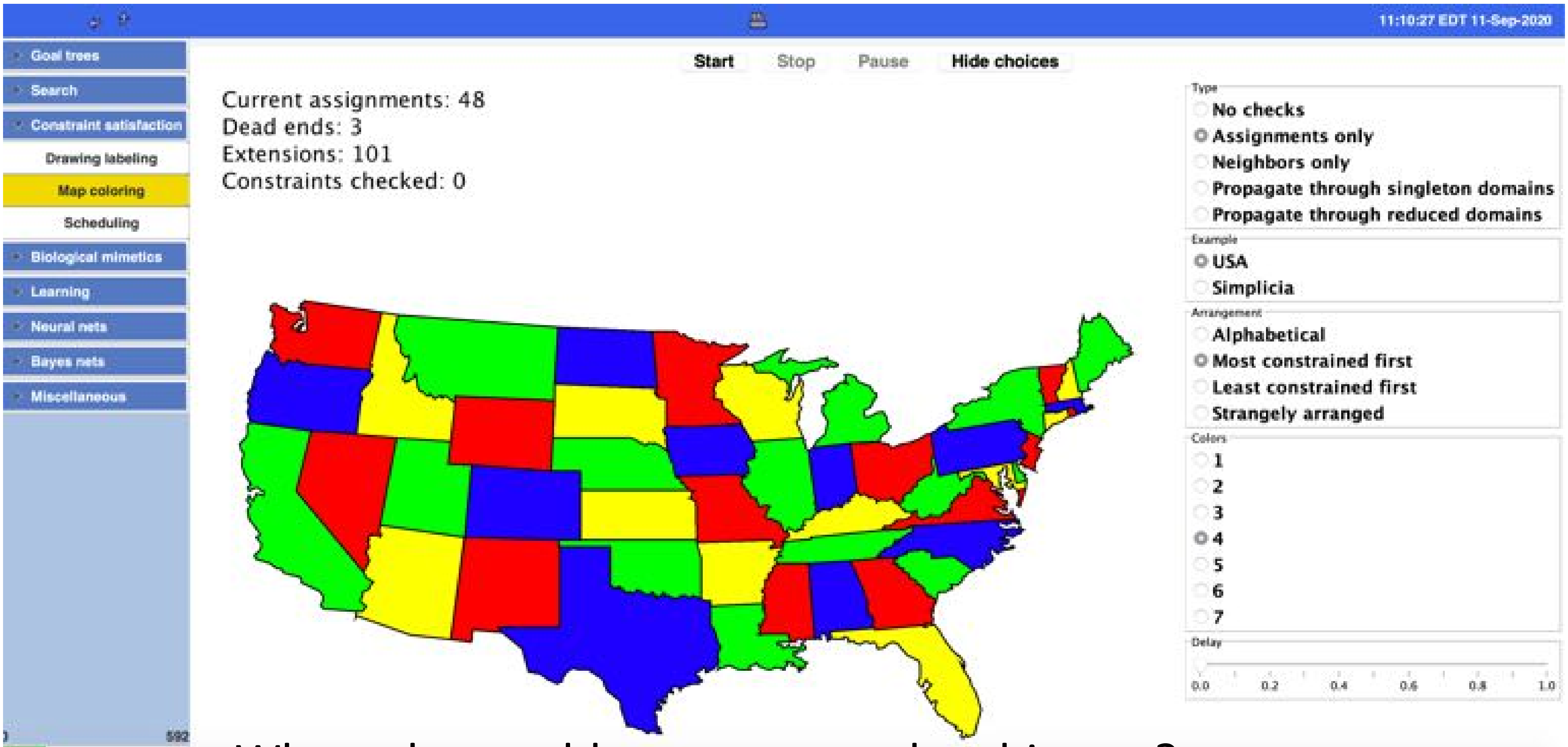
Flourish 2: color most constrained state first – end state



Check assignments only, no constraint propagation, most constrained state first



The dirty little secret: check assignments only, no constraint propagation, most constrained state first



What other problems can we solve this way?

What Do We “consider”?

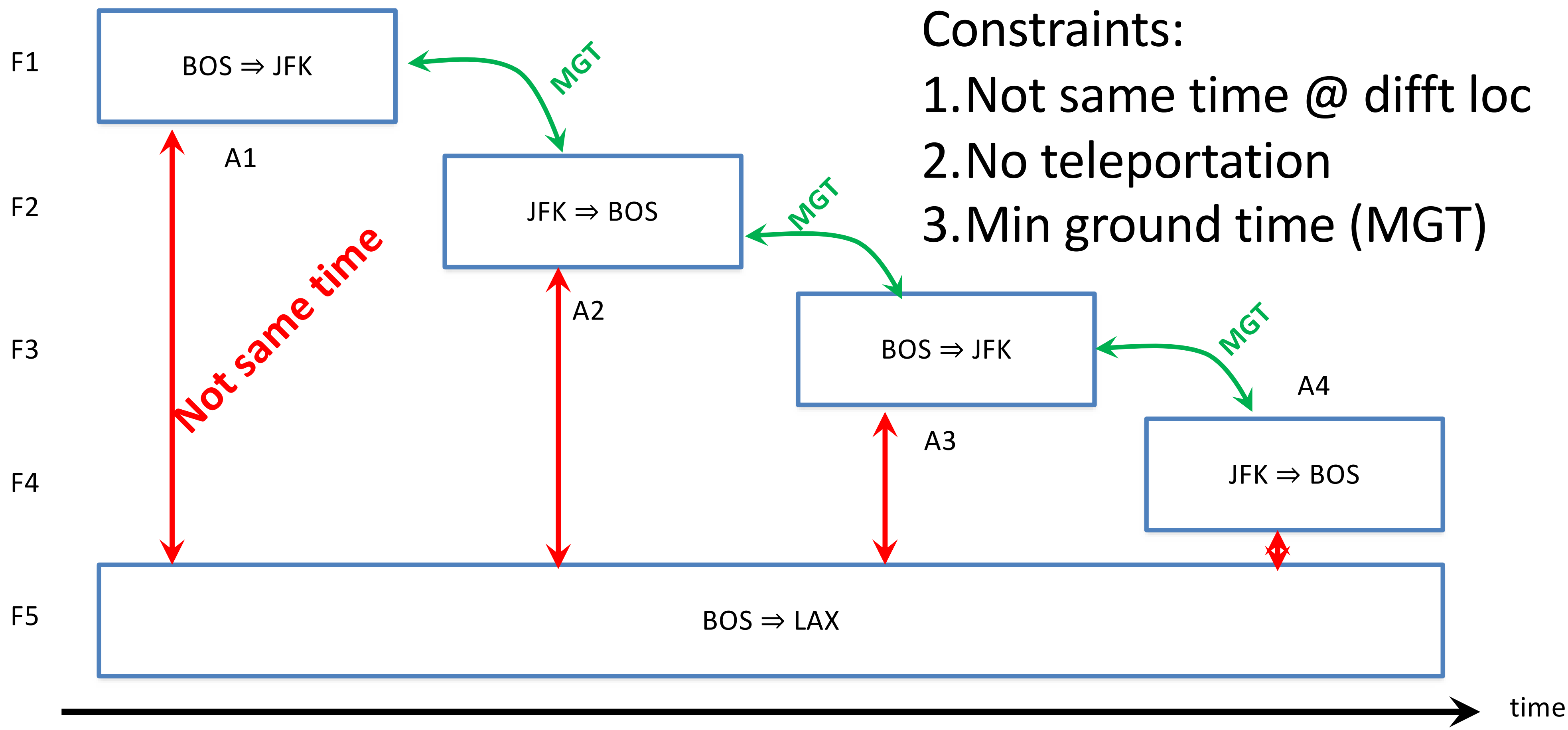
ordering of states: <u>s</u>trange, <u>al</u>phabetic, <u>m</u>ost, <u>l</u>east constrained

Consider	dead ends	extensions	constraints checked	
Assignments only	≈∞	≈∞	0	s
	1827	9217	0	a
	3	101	0	m
	≈∞	≈∞	0	l
Neighbors only	406	2113	4667	s
	0	82	244	a
	0	86	224	m
	1371	6945	10302	l
Propagate through singleton domains	0	75	585	s
	0	82	492	a
	0	86	299	m
	0	82	492	l
Propagate through reduced domains	0	75	2095	s
	0	82	2074	a
	0	86	1725	m
	0	82	2074	l

What other problems can we solve by constraint satisfaction?

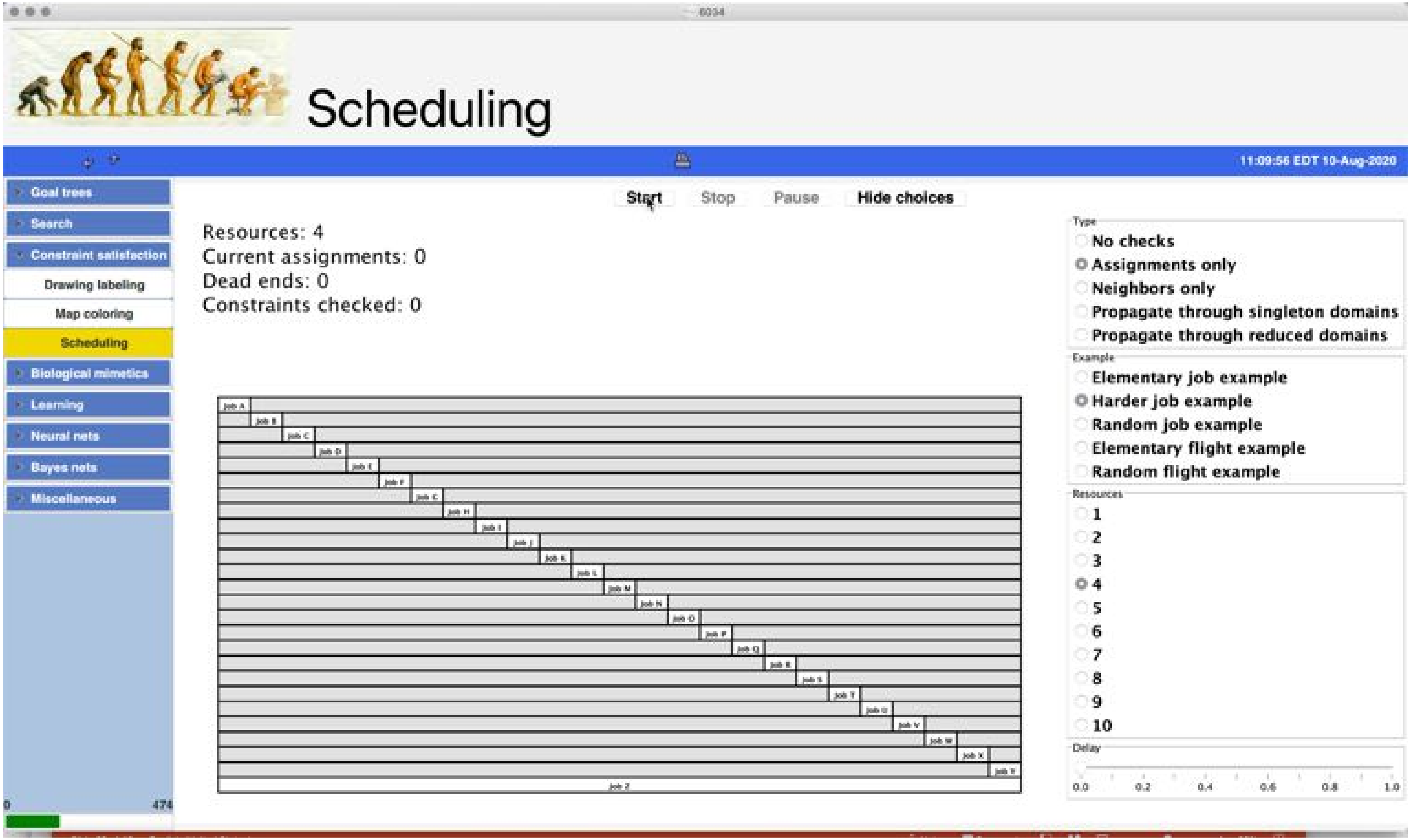
Resource Allocation via Constraint Propagation

- Consider JetGreen Airlines with the following proposed schedule, using 4 aircraft:

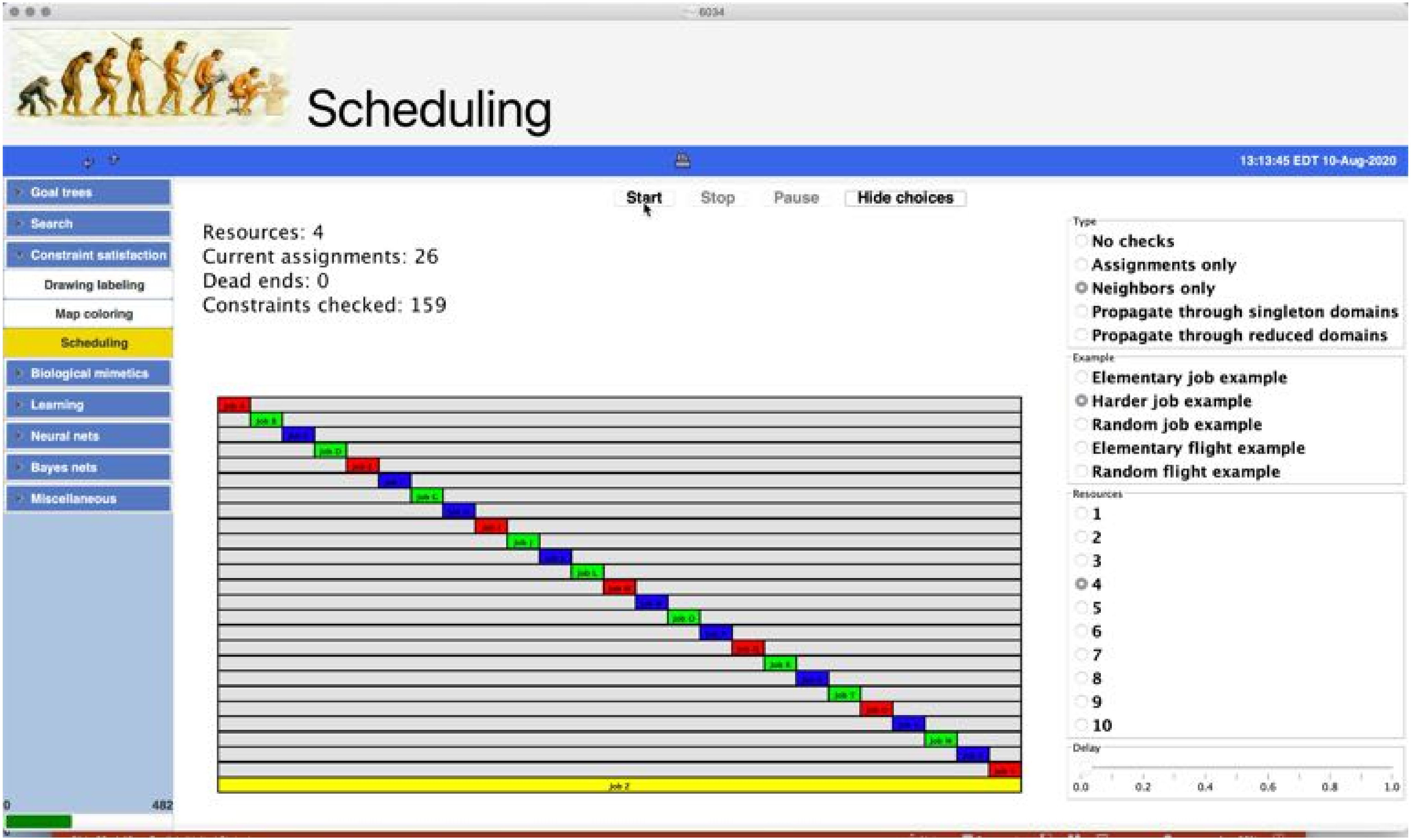


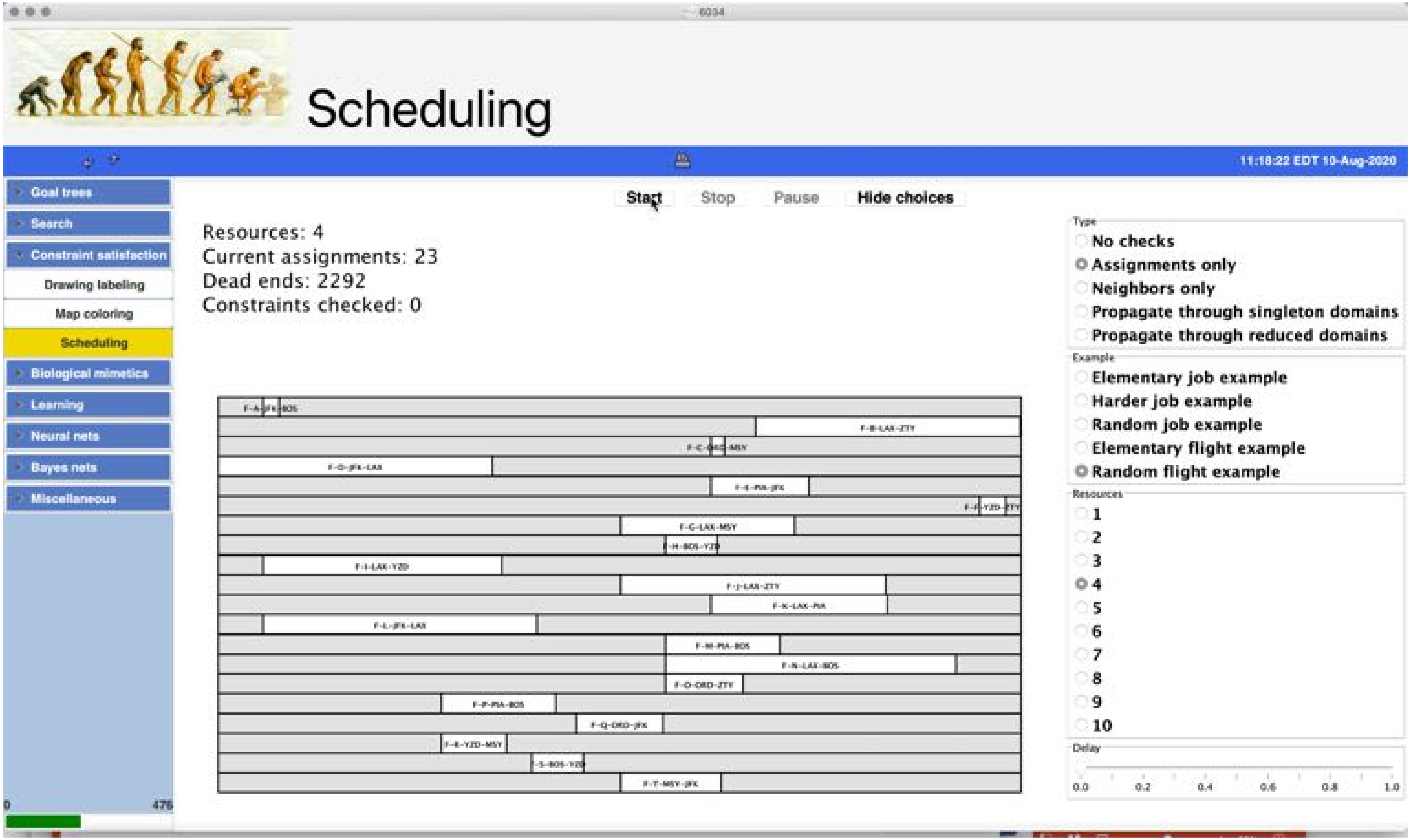
Constraints:

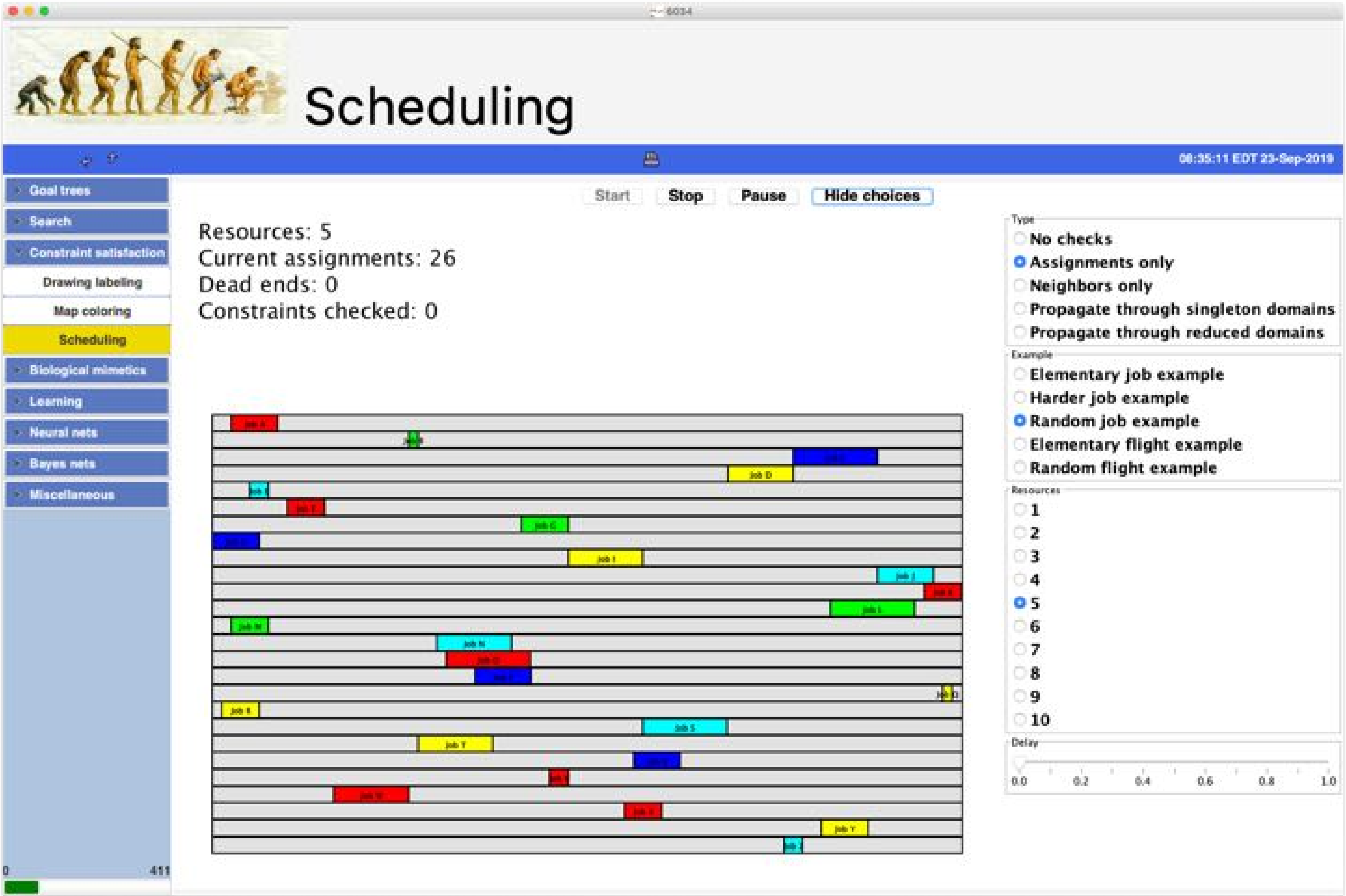
1. Not same time @ difft loc
2. No teleportation
3. Min ground time (MGT)



Scheduling
11:09:56 EDT 10-Aug-2020
Goal trees
Search
Constraint satisfaction
Drawing labeling
Map coloring
Scheduling
Biological mimetics
Learning
Neural nets
Bayes nets
Miscellaneous
0
474
Start
Stop
Pause
Hide choices
Resources: 4
Current assignments: 0
Dead ends: 0
Constraints checked: 0
Type
No checks
Assignments only
Neighbors only
Propagate through singleton domains
Propagate through reduced domains
Example
Elementary job example
Harder job example
Random job example
Elementary flight example
Random flight example
Resources
1
2
3
4
5
6
7
8
9
10
Delay
0.0
0.2
0.4
0.6
0.8
1.0


Scheduling
13:13:45 EDT 10-Aug-2020
Goal trees
Search
Constraint satisfaction
Drawing labeling
Map coloring
Scheduling
Biological mimetics
Learning
Neural nets
Bayes nets
Miscellaneous
Start
Stop
Pause
Hide choices
Resources: 4
Current assignments: 26
Dead ends: 0
Constraints checked: 159
Type
No checks
Assignments only
Neighbors only
Propagate through singleton domains
Propagate through reduced domains
Example
Elementary job example
Harder job example
Random job example
Elementary flight example
Random flight example
Resources
1
2
3
4
5
6
7
8
9
10
Delay
0.0
0.2
0.4
0.6
0.8
1.0
0
482
Job Z


Scheduling
11:18:22 EDT 10-Aug-2020
Goal trees
Search
Constraint satisfaction
Drawing labeling
Map coloring
Scheduling
Biological mimetics
Learning
Neural nets
Bayes nets
Miscellaneous
Start
Stop
Pause
Hide choices
Resources: 4
Current assignments: 23
Dead ends: 2292
Constraints checked: 0
Type
No checks
Assignments only
Neighbors only
Propagate through singleton domains
Propagate through reduced domains
Example
Elementary job example
Harder job example
Random job example
Elementary flight example
Random flight example
Resources
1
2
3
4
5
6
7
8
9
10
Delay
0.0
0.2
0.4
0.6
0.8
1.0
0
476


Scheduling
08:35:11 EDT 23-Sep-2019
Goal trees
Search
Constraint satisfaction
Drawing labeling
Map coloring
Scheduling
Biological mimetics
Learning
Neural nets
Bayes nets
Miscellaneous
0
411
Start
Stop
Pause
Hide choices
Resources: 5
Current assignments: 26
Dead ends: 0
Constraints checked: 0
Type
No checks
Assignments only
Neighbors only
Propagate through singleton domains
Propagate through reduced domains
Example
Elementary job example
Harder job example
Random job example
Elementary flight example
Random flight example
Resources
1
2
3
4
5
6
7
8
9
10
Delay
0.0
0.2
0.4
0.6
0.8
1.0

Summary: Constraint satisfaction

- Different architectures/algorithms that exploit
  - Constraints exposed by
    - Representations that support
      - Models of perception, thinking, and action

Many different, difficult problems can be solved this way!

Dutch human language analysis solved like line labeling

"We know that Cecilia tries to teach Hans to crack a nut"

We weten dat Cecilia Hans een noot probeert te leren kraken