Sequential and Parallel Algorithms for Mixed Packing and Covering

(from FOCS 2001)

Neal E. Young

UC Riverside

Marek is being punished.

He can eat only bacon, beans, and beets!

Can Marek get enough of what he needs
without getting too much of what he doesn’t?

Bacon

Nutrition Facts

Serving Size 4 pc

Servings per container 10

Amount Per Serving

Calories 110 Calories from Fat 80

	% Daily Value
Total Fat 10g	30 %
Saturated Fat 9g	50 %
Cholesterol 3mg	10 %
Sodium 500mg	40 %
Total Carbohydrates 0g	0 %
Dietary Fiber 0g	0 %
Sugars 0g	
Protein 5g	30 %

Vitamin A 0%	Vitamin B 7%
Vitamin C 2%	Iron 16%

Beans

Nutrition Facts

Serving Size 1 cup

Servings per container 2

Amount Per Serving

Calories 200 Calories from Fat 16

	% Daily Value
Total Fat 2g	5 %
Saturated Fat 0g	0 %
Cholesterol 0mg	0 %
Sodium 0 mg	0 %
Total Carbohydrates 6g	20 %
Dietary Fiber 2g	15 %
Sugars 0g	
Protein 6g	35 %

Vitamin A 40%	Vitamin B 0%
Vitamin C 22%	Iron 2%

Beets

Nutrition Facts

Serving Size 3 oz

Servings per container 5

Amount Per Serving

Calories 180 Calories from Fat 0

	% Daily Value
Total Fat 0g	0 %
Saturated Fat 0g	0 %
Cholesterol 3mg	15 %
Sodium 30mg	2 %
Total Carbohydrates 8g	30 %
Dietary Fiber 2g	15 %
Sugars 6g	
Protein 0g	0 %

Vitamin A 40%	Vitamin B 52%
Vitamin C 26%	Iron 3%

unknowns

bacon, bean, beet

1 serving beans has 35% of the RDA of protein

constraints

```
   protein: 30 bacon + 35 bean               > 100
 vitamin A:            40 bean + 43 beet > 100
 vitamin B:  7 bacon           + 52 beet > 100
 vitamin C:  2 bacon + 22 bean + 26 beet > 100
       fat: 30 bacon +  5 bean               < 100
     sugar:            15 bean + 37 beet < 100
      salt: 40 bacon           +  2 beet < 100
cholesterol: 10 bacon + 10 bean + 15 beet < 100
```

ε-approximate solutions:

```
    protein: 30 bacon + 35 bean           > (1-ϵ)100
  vitamin A:           40 bean + 43 beet > (1-ϵ)100
  vitamin B:  7 bacon          + 52 beet > (1-ϵ)100
  vitamin C:  2 bacon + 22 bean + 26 beet > (1-ϵ)100
        fat: 30 bacon +  5 bean           < (1+ϵ)100
      sugar:           15 bean + 37 beet < (1+ϵ)100
       salt: 40 bacon          +  2 beet < (1+ϵ)100
cholesterol: 10 bacon + 10 bean + 15 beet < (1+ϵ)100
```

Bibliography

[1950] von Neumann. Numerical method for determination of the value and the best strategies of a zero-sum two-person game with large numbers of strategies.

[1950] Brown and von Neumann. Solutions of games by differential equations.

[1952] Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. --- Chernoff bound, implicit use of Lmax-like functions

[1958] Ford and Fulkerson. A suggested computation for maximal multicommodity flow.

[1960] Dantzig and Wolfe. Decomposition principle for linear programs.

[1962] Benders. Partitioning procedures for solving mixed-variables programming problems.

[1971] Held and Karp. The traveling salesman problem and minimum spanning trees.

[1977] Khachiyan. Convergence rate of the game processes for solving matrix games.

...

[1979] Shapiro. A survey of Lagrangean techniques for discrete optimization.
Annals of Discrete Mathematics, 5:113--138, 1979.

Bibliography

Grigoriadis and Khachiyan.

A sublinear-time randomized approximation algorithm for matrix games. OR Research Letters, 1995.

An exponential-function reduction method for block-angular convex programs. Networks, 1995.

Young. Randomized rounding without solving the linear program. SODA, 1995.

Karger and Plotkin. Adding multiple cost constraints to combinatorial optimization problems, with applications to multicommodity flows. STOC, 1995.

Grigoriadis and Khachiyan.

Coordination complexity of parallel price-directive decomposition. MOR, 1996.

Approximate minimum-cost multicommodity flows in $o(knm/\varepsilon^2)$ time. Math. Programming, 1996.

Garg and Konemann. Faster and simpler algorithms for multicommodity flow and other fractional packing problems. FOCS, 1998. --- variable-size increments

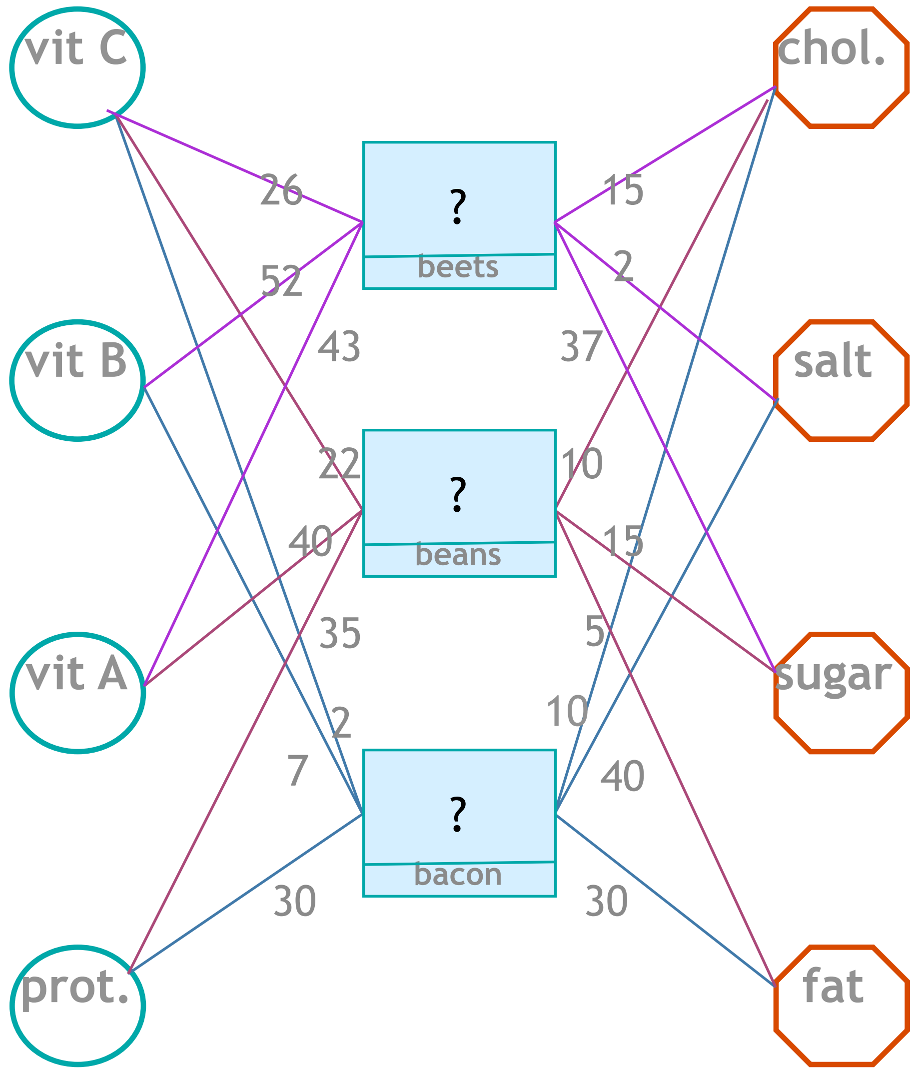
Konemann. Fast Combinatorial Algorithms for Packing and Covering Problems

PhD thesis, Max-Planck-Institute for Informatik, 2000. --- dropping met covering constraints

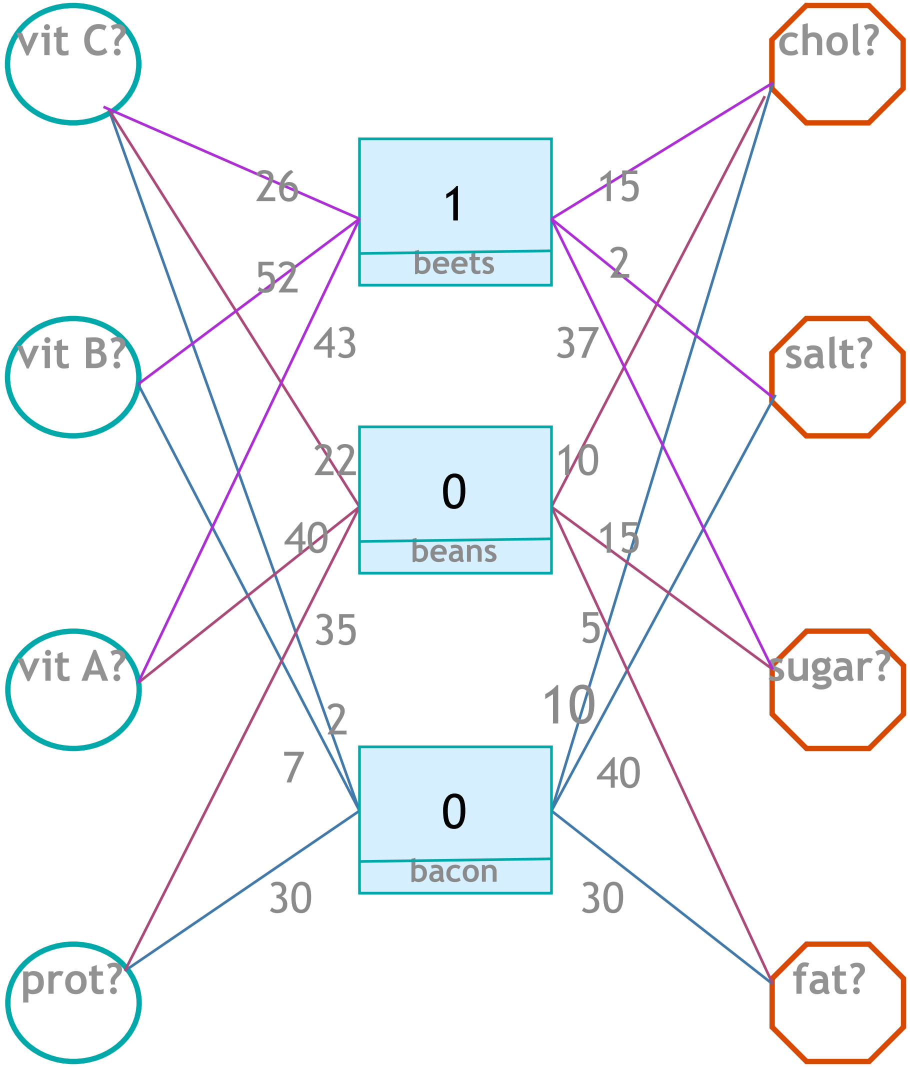
Fleischer. Approximating fractional multicommodity flow independent of the number of commodities. SIAM J. Discrete Math, 2000. --- partitioning increments into phases

linear program

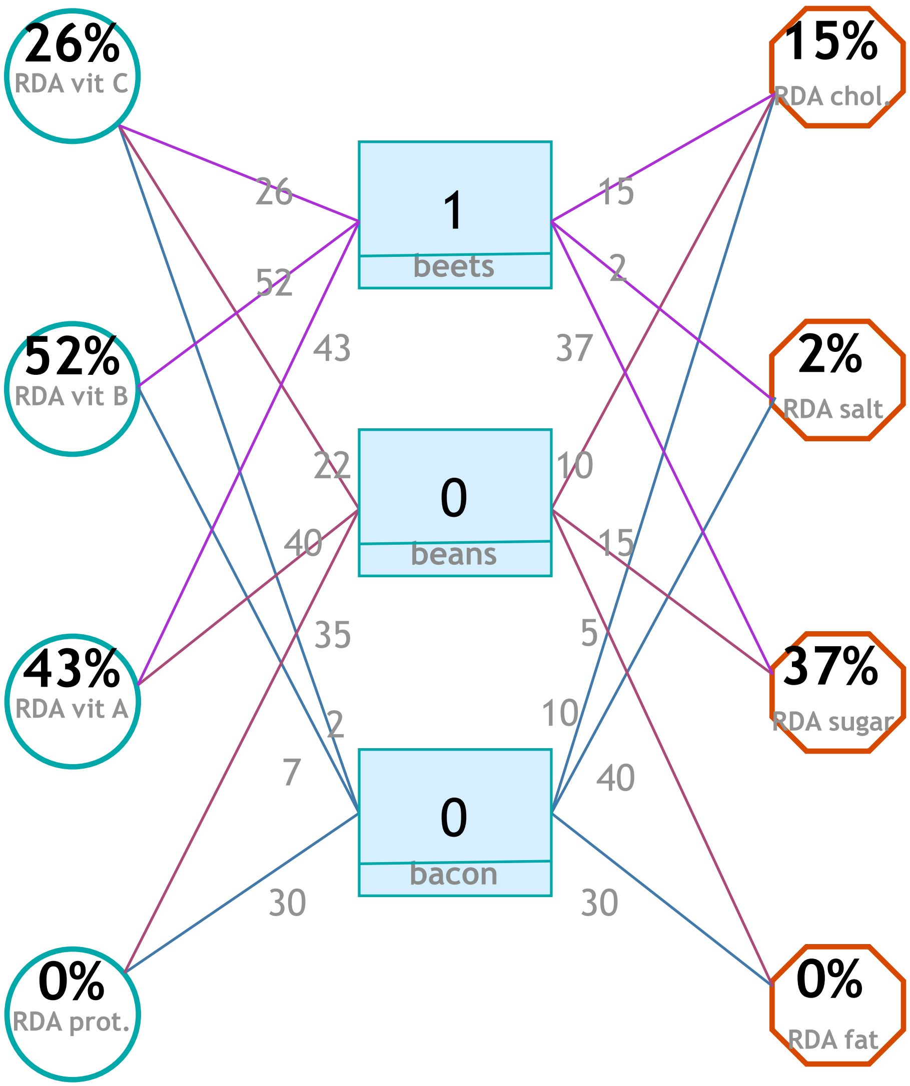
```
30 bacon + 35 bean              > 100
           40 bean + 43 beet > 100
 7 bacon            + 52 beet > 100
 2 bacon + 22 bean + 26 beet > 100

30 bacon +  5 bean              < 100
           15 bean + 37 beet < 100
40 bacon +             2 beet < 100
10 bacon + 10 bean + 15 beet < 100
```



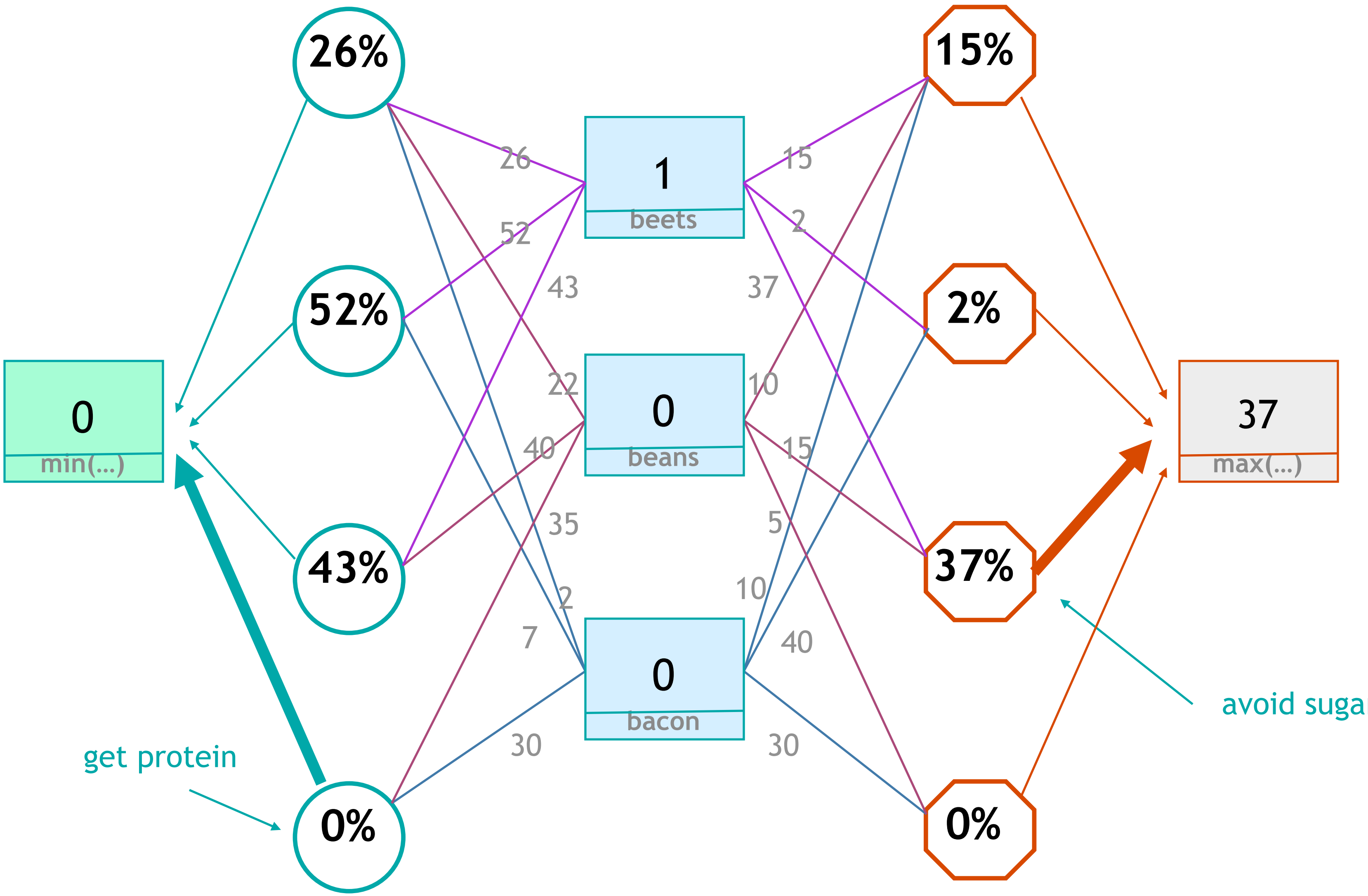
One serving of beets?



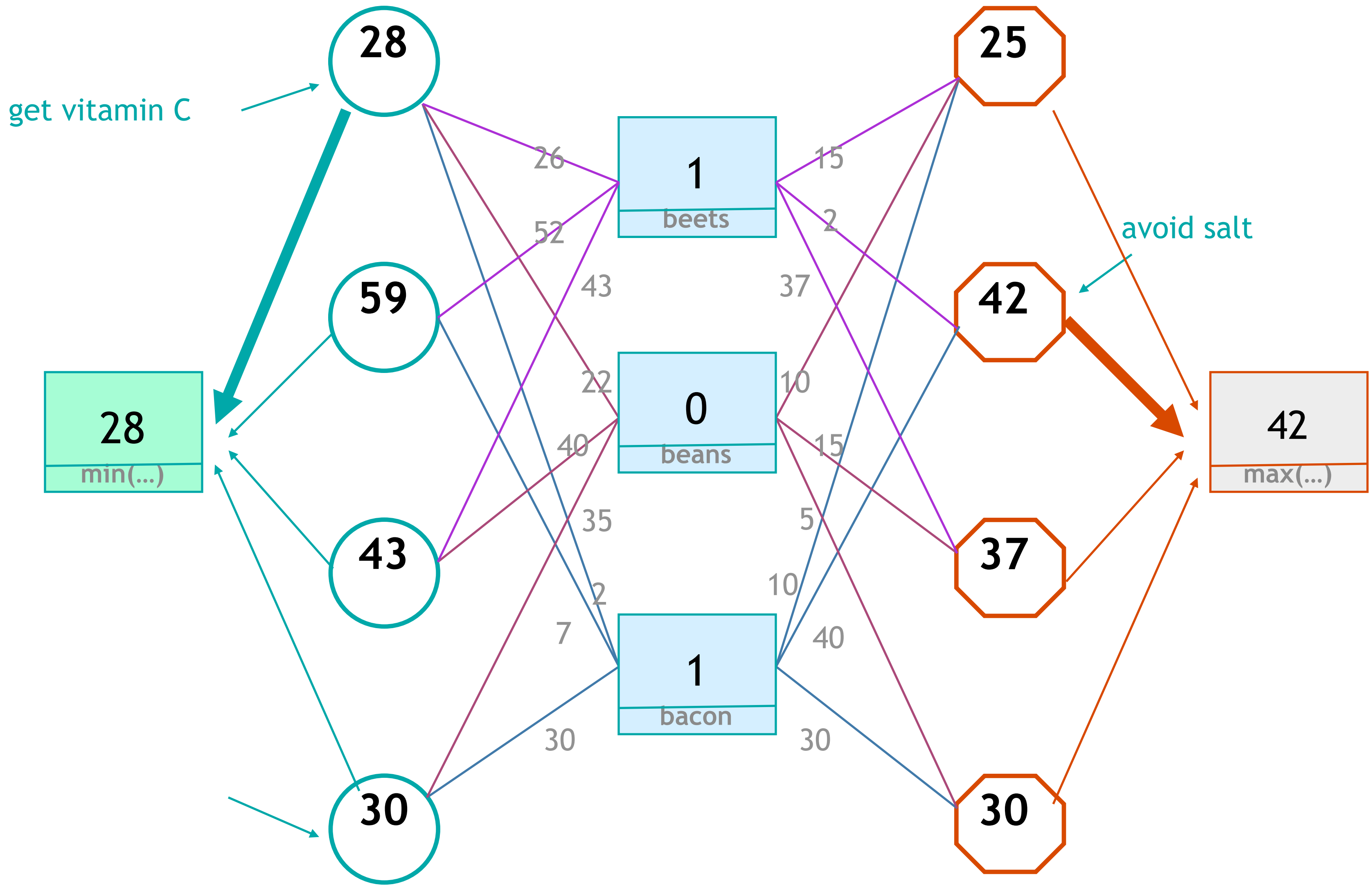
After one serving of beets

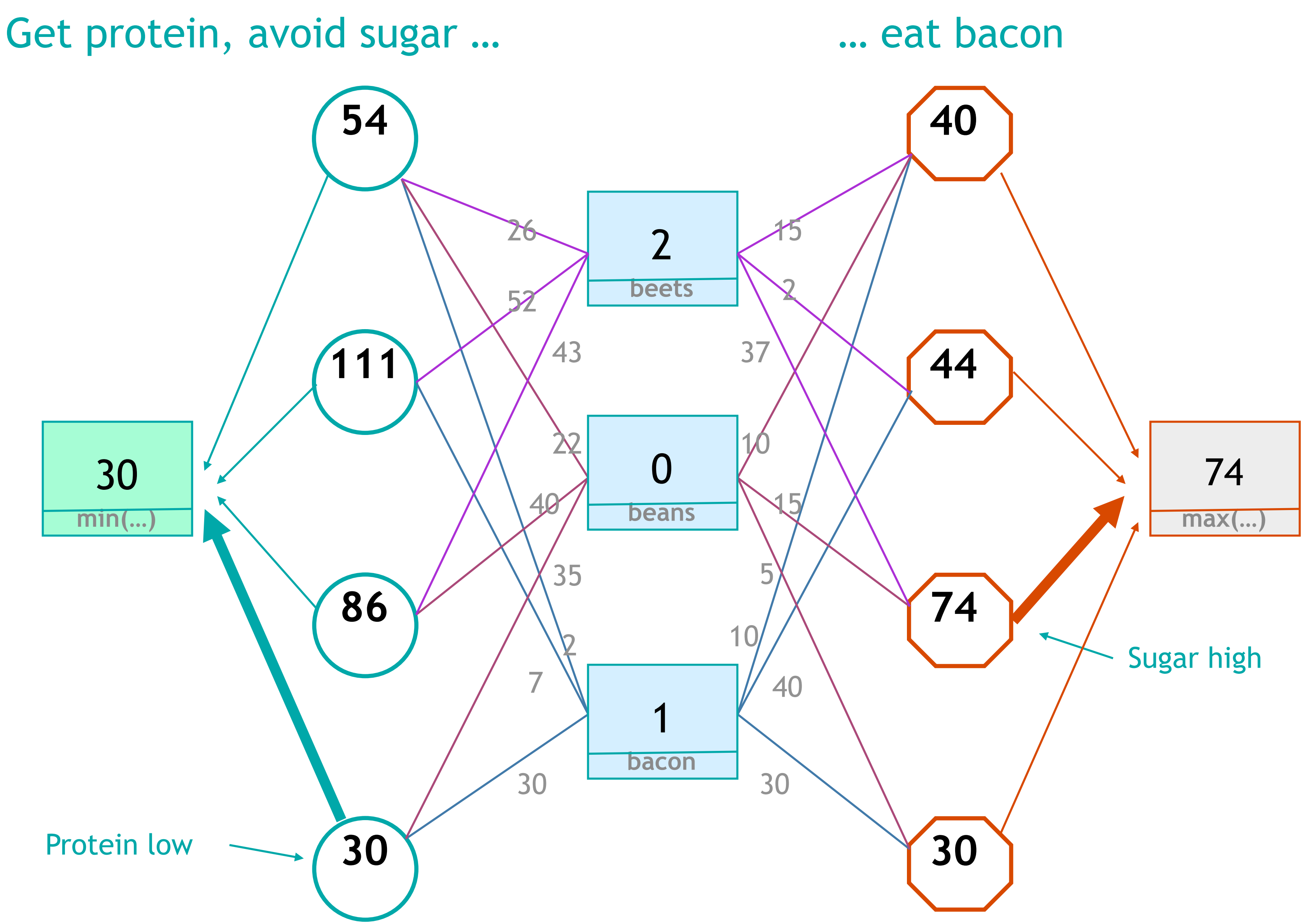


Greedy approach: Get protein, avoid sugar … … eat bacon

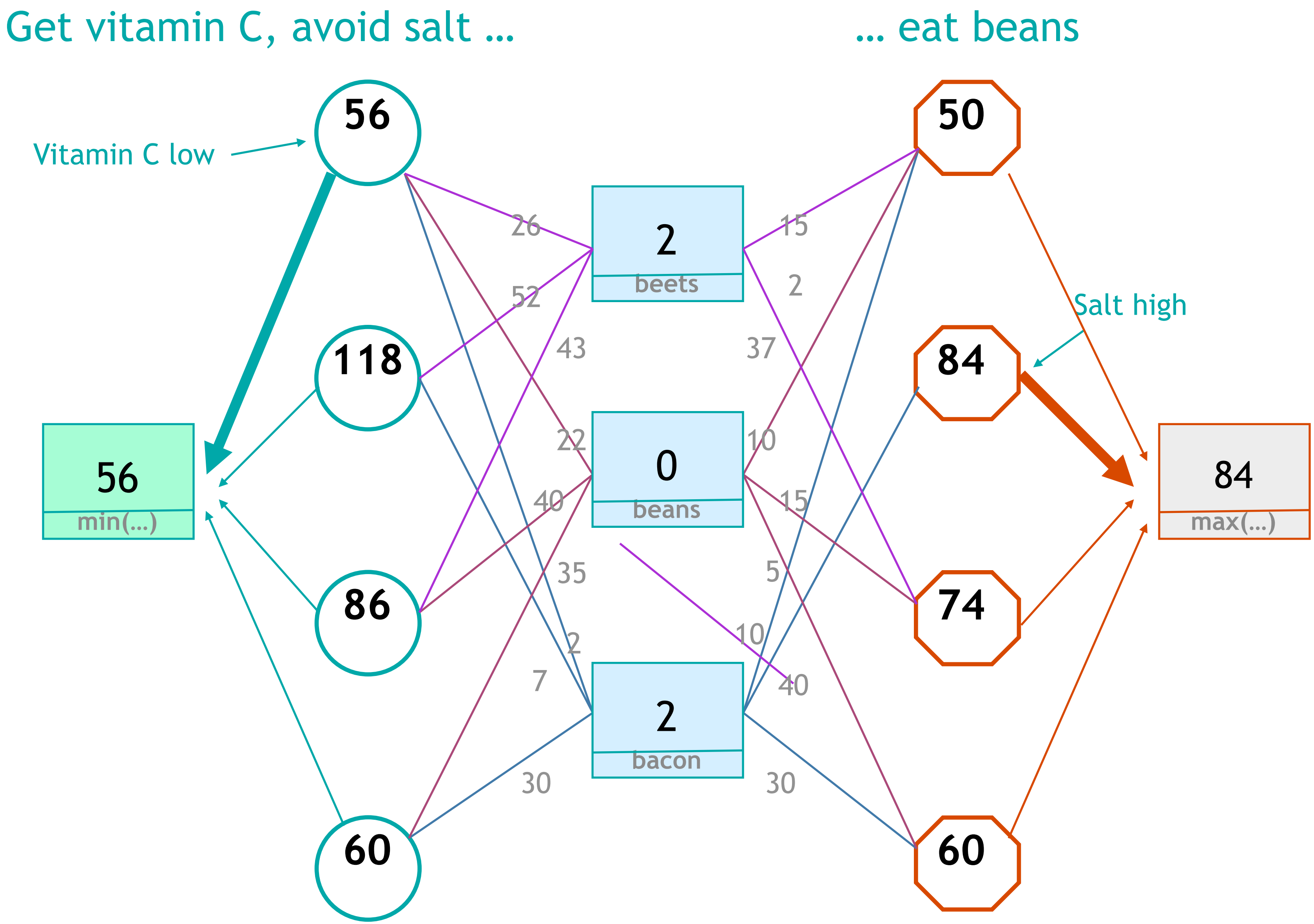


Get vitamin C, avoid salt … … eat beets



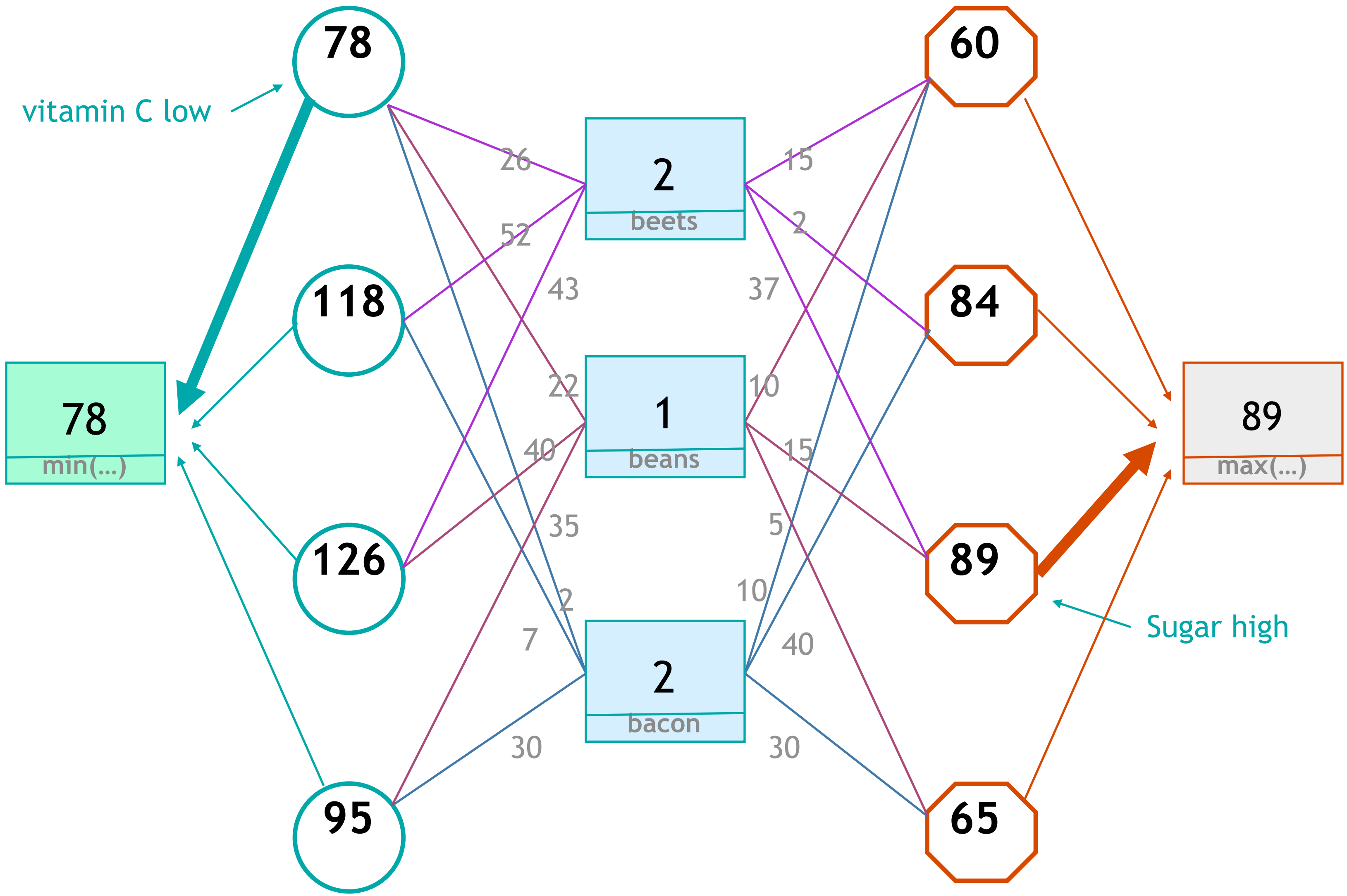


Get protein, avoid sugar …
… eat bacon
54
111
86
30
30
min(…)
2
beets
0
beans
1
bacon
26
52
43
22
40
35
2
7
30
15
2
37
10
15
5
10
40
30
40
44
74
30
74
max(…)
Sugar high
Protein low

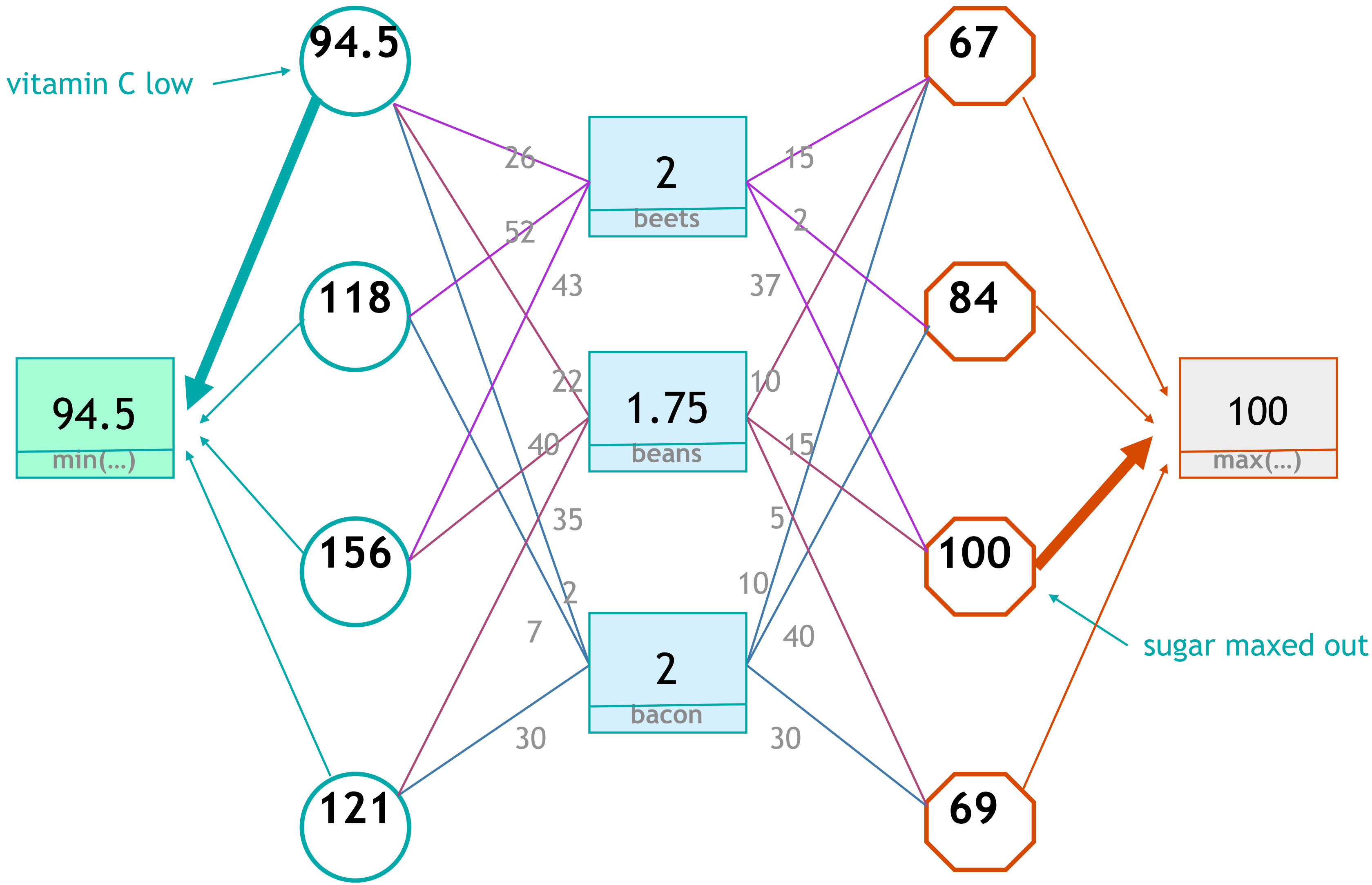


Get vitamin C, avoid salt ...
... eat beans
Vitamin C low
Salt high
56
118
86
60
56
min(...)
2
beets
0
beans
2
bacon
26
52
43
22
40
35
2
7
30
15
2
37
10
15
5
10
40
30
50
84
74
60
84
max(...)

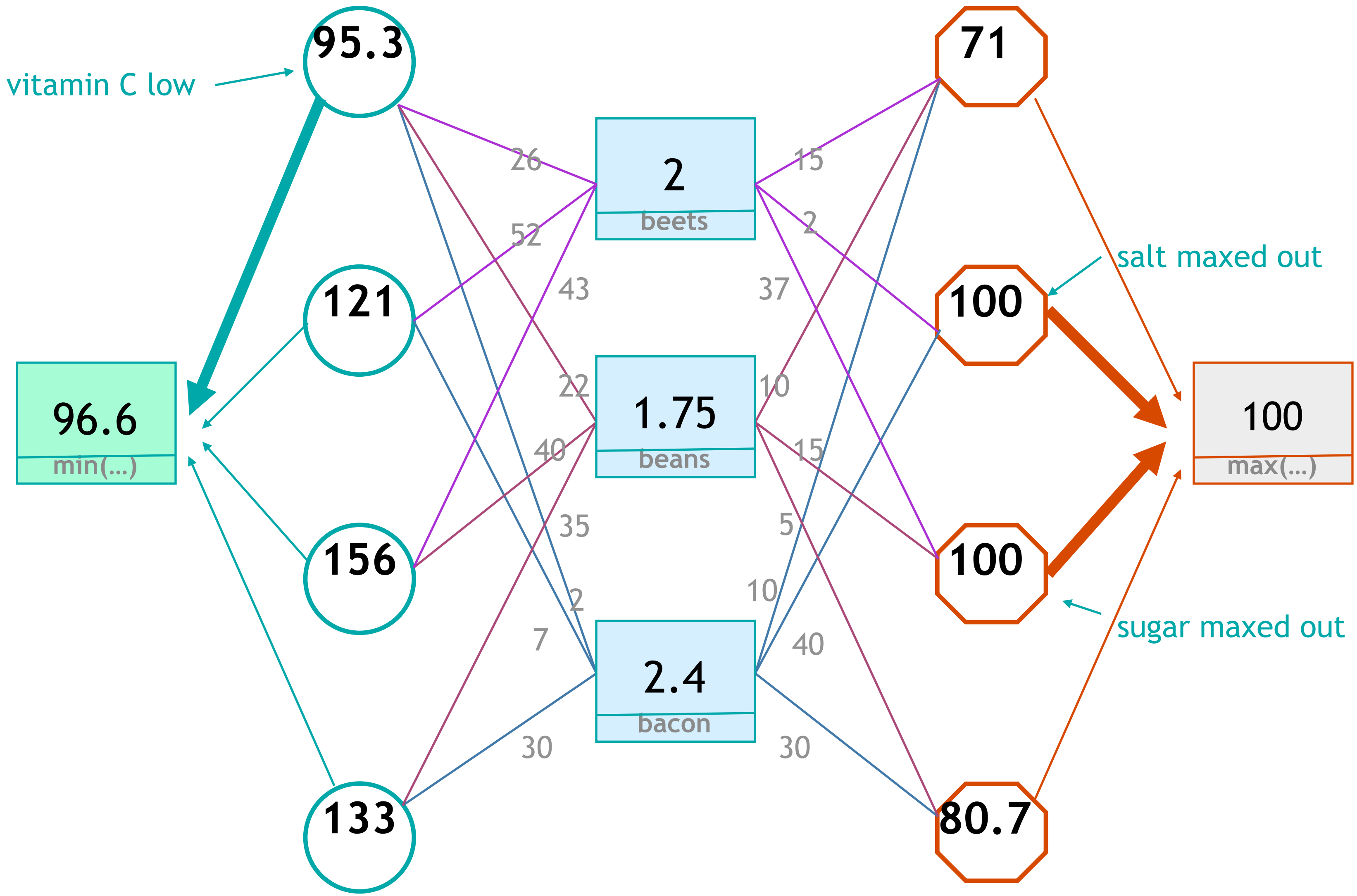
Get vitamin C, avoid sugar … … eat beans?



Get vitamin C, completely avoid sugar… … eat bacon?



Get vitamin C, completely avoid sugar and salt … … stuck!



Making a greedy approach work

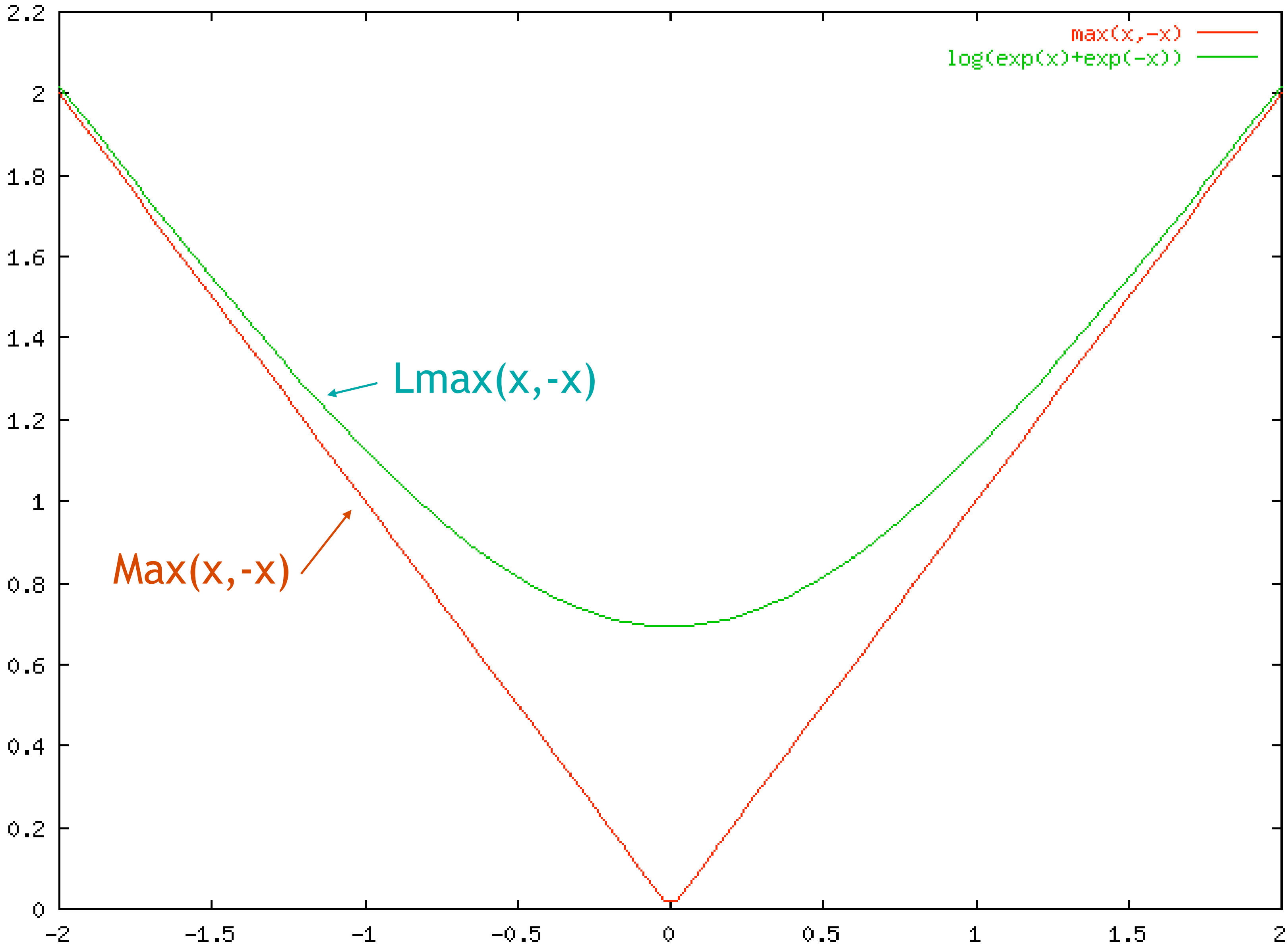
- Balance all needs in each step.
- Take small bites.

Balancing all needs…

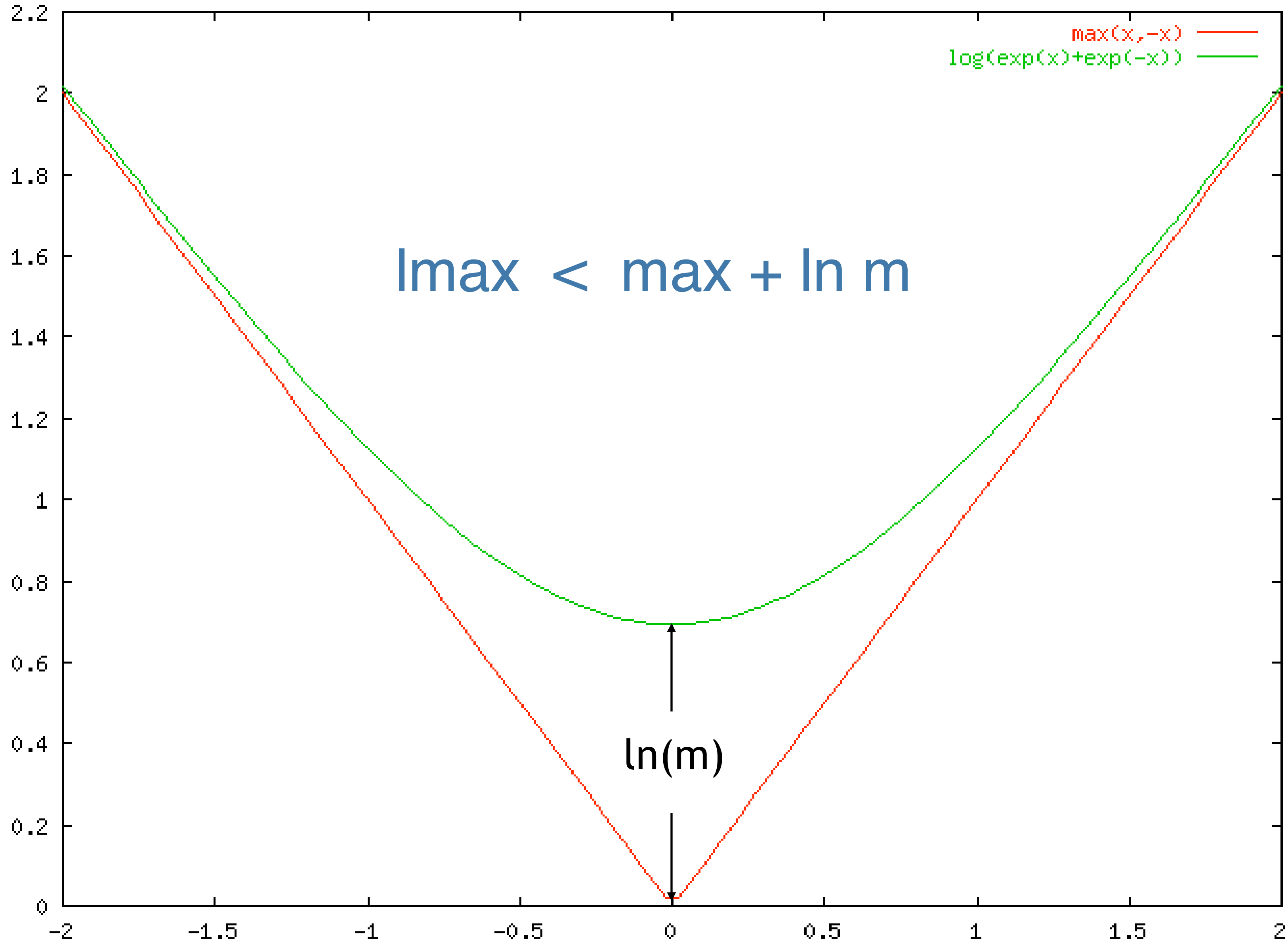
Smooth approximations of Max() and Min()

$$\mathrm{Lmax}(y_1, y_2, \ldots, y_m) = \ln \sum \mathrm{e}^{y_i}$$

$$\mathrm{Lmin}(y_1, y_2, \ldots, y_m) = -\ln \sum \mathrm{e}^{-y_i}$$

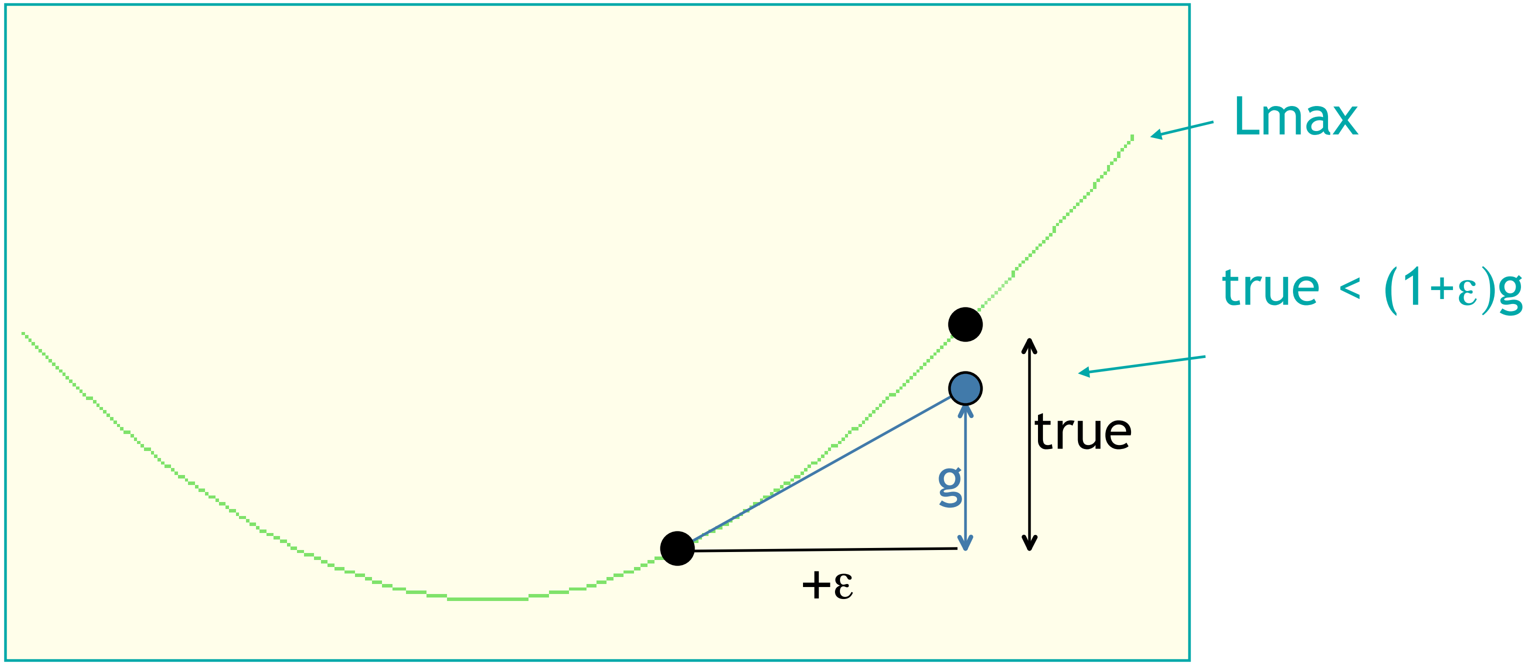


max(x,-x)
log(exp(x)+exp(-x))
Lmax(x,-x)
Max(x,-x)
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2


max(x,-x)
log(exp(x)+exp(-x))
lmax < max + ln m
ln(m)
2.2
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

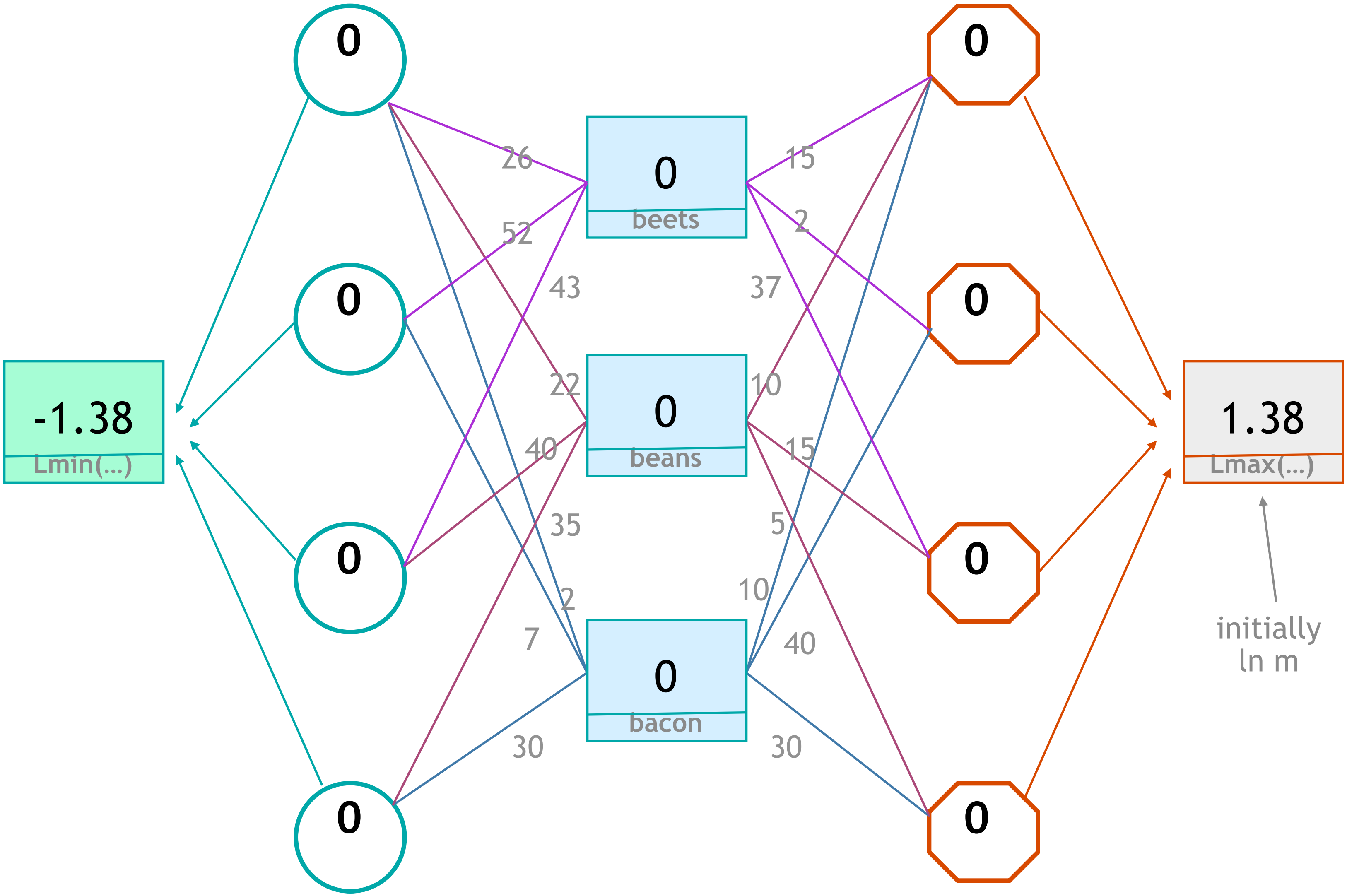
Change in Lmax() when inputs change:

gradient estimate is ε-approximate within ε-neighborhood



g = change estimated by gradient

Algorithm: use Lmin and Lmax instead of min and max.
Choose increments so Lmin increases by > (1-ε) times as much as Lmax.



Choose increments so Lmin increases by > (1-ε) times as much as Lmax.

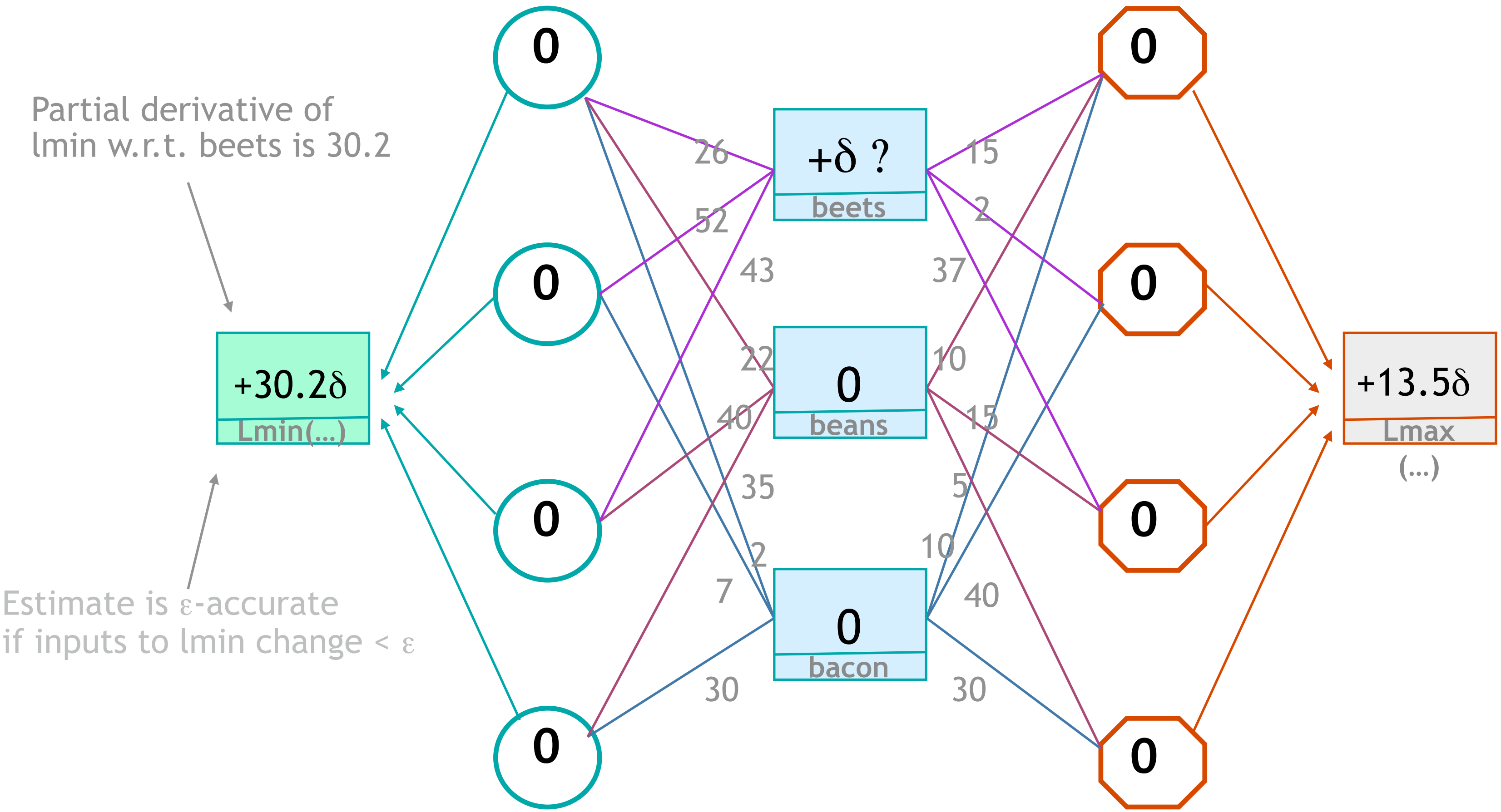
Stop when Lmin > ln(m)/ε (= 13.8)

to get ε-approximate solution:

target

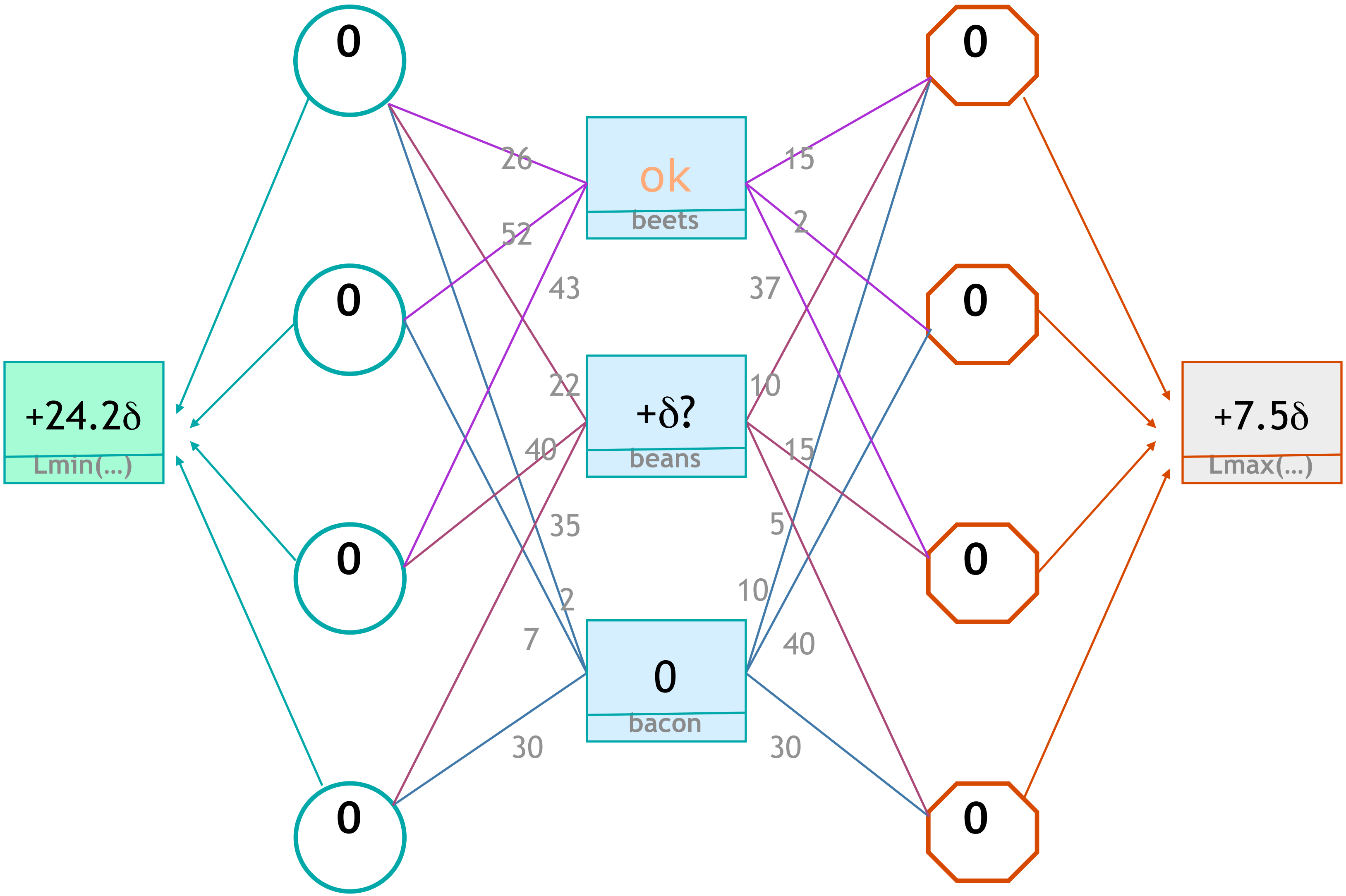
```
30 bacon + 35 bean           > ln(m)/ϵ
           40 bean + 43 beet > 13.8
 7 bacon           + 52 beet > 13.8
 2 bacon + 22 bean + 26 beet > 13.8

30 bacon +  5 bean           <  ln(m) + [ln(m)/ϵ + ln(m)]/(1-ϵ)  = (1+O(ϵ))ln(m)/ϵ
           15 bean + 37 beet <  (1+O(ϵ))13.8
40 bacon +            2 beet <  (1+O(ϵ))13.8
10 bacon + 10 bean + 15 beet <  (1+O(ϵ))13.8
```

Use gradients to estimate increase in Lmin and Lmax.

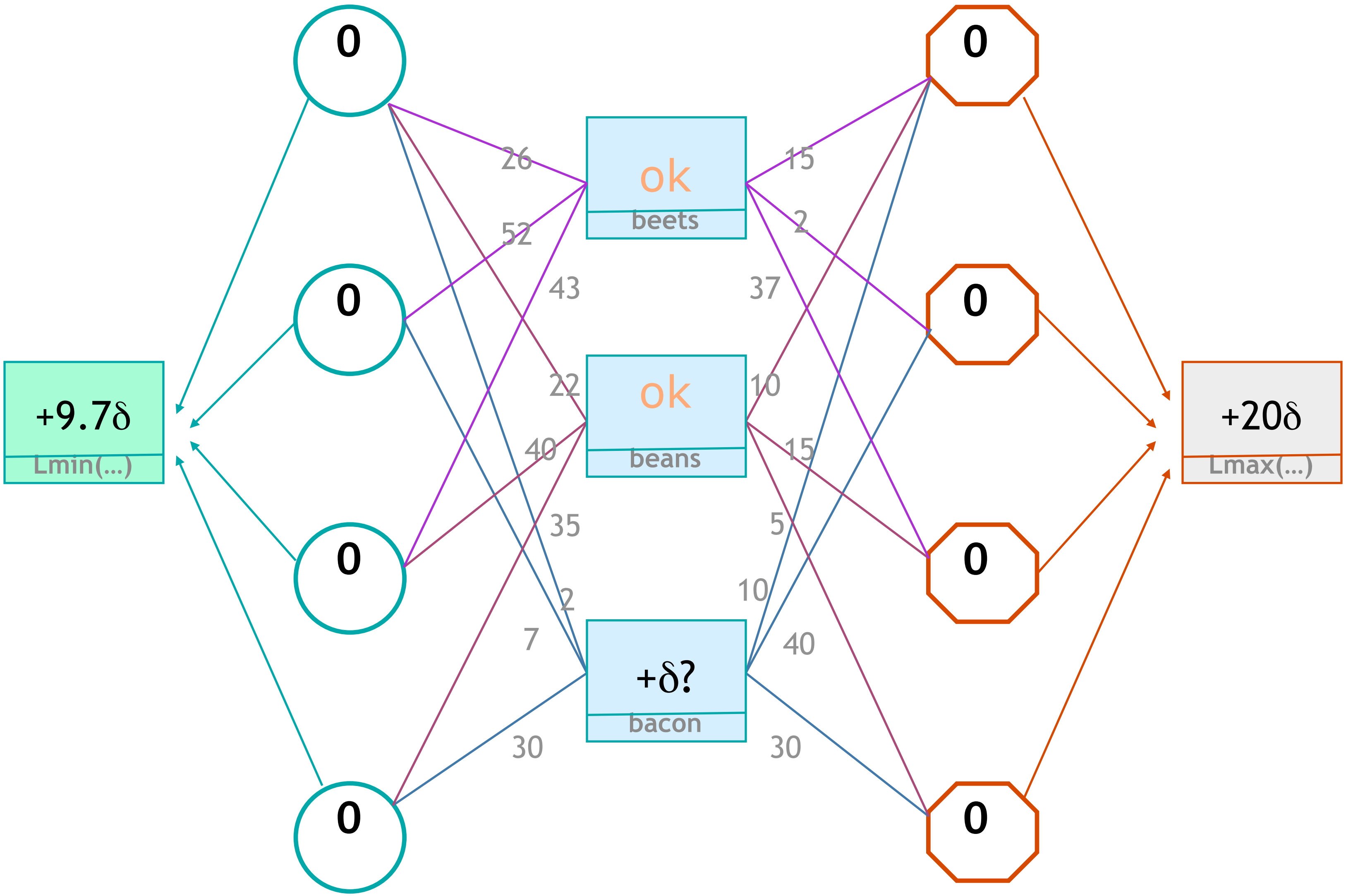


Variable ok to raise if est. increase in lmin > .9 est. increase in lmax

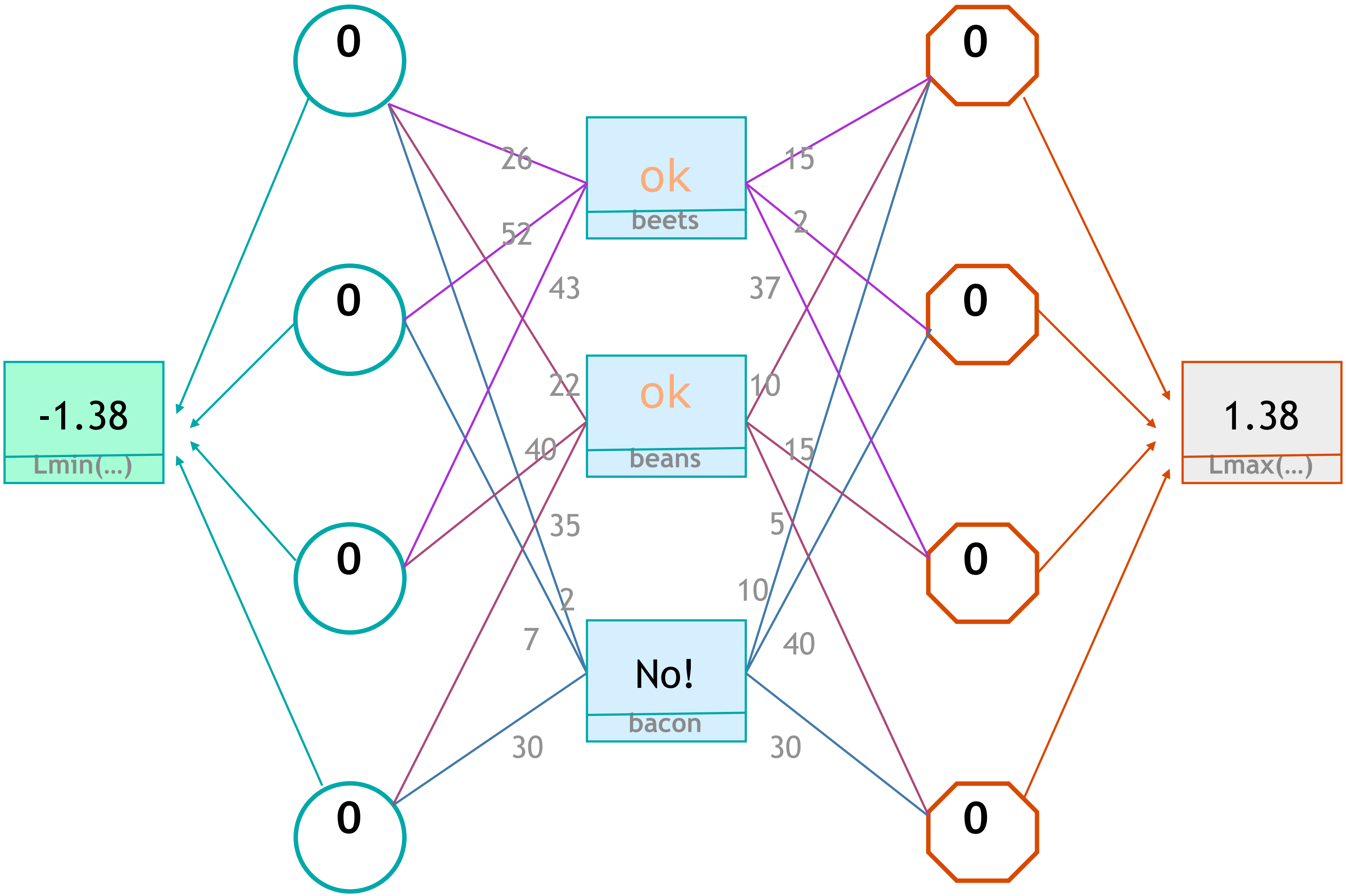
Beets are ok, what about beans?



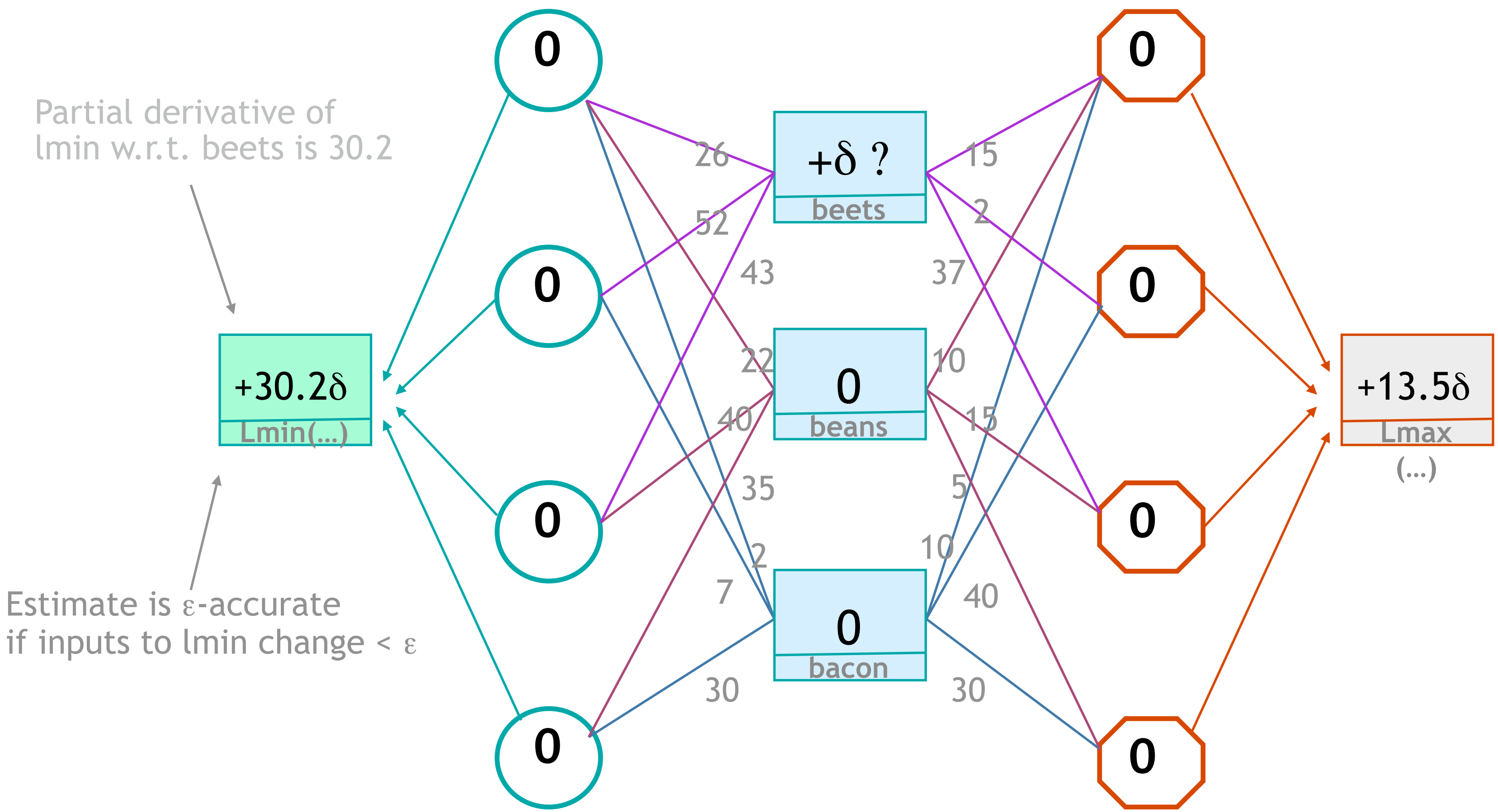
What about bacon?



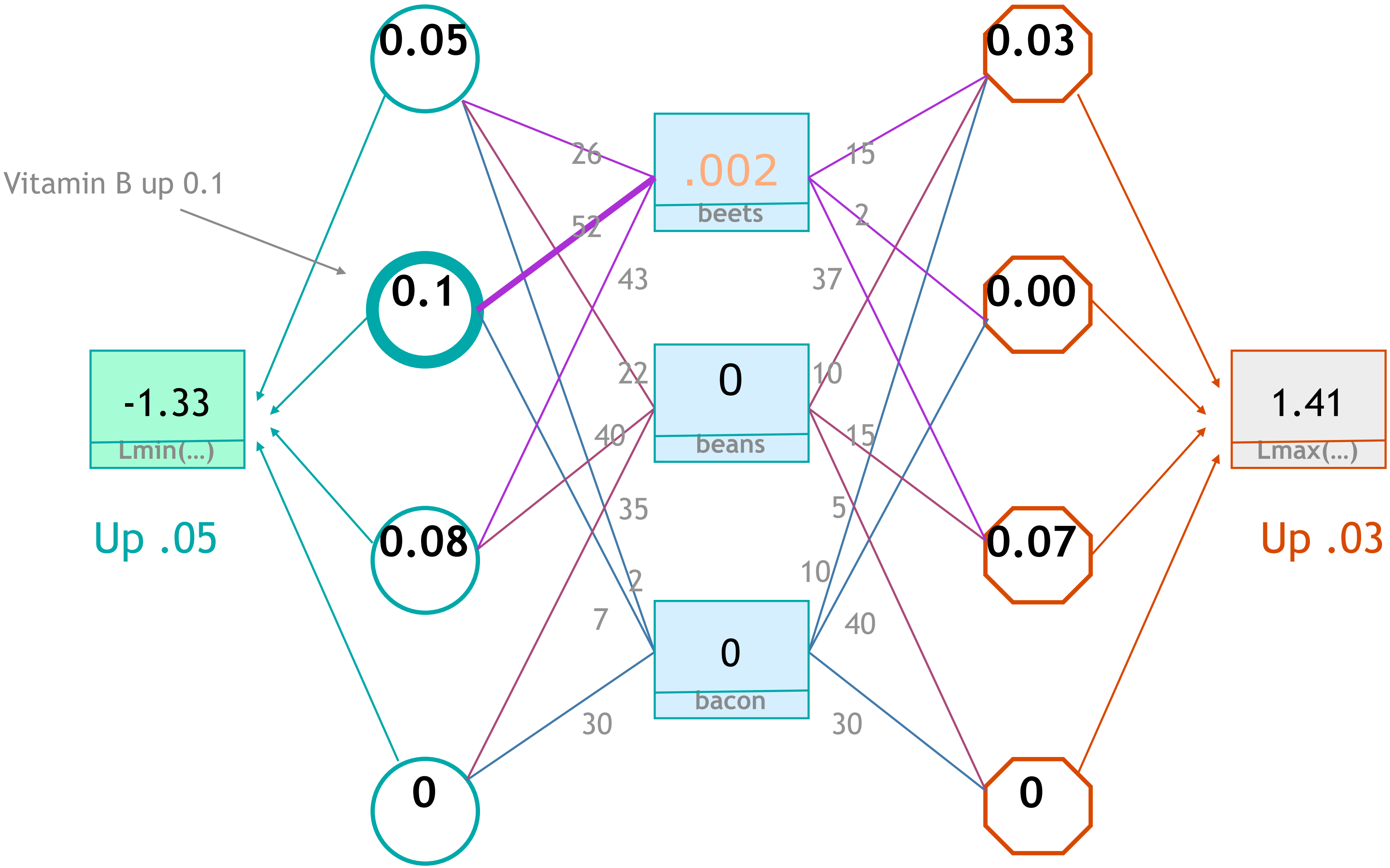
Beets and beans are okay to raise, but not bacon.



Raise beets. Now: by how much?
For gradient estimates to be ε-accurate,
need inputs to lmin and lmax to change by < ε

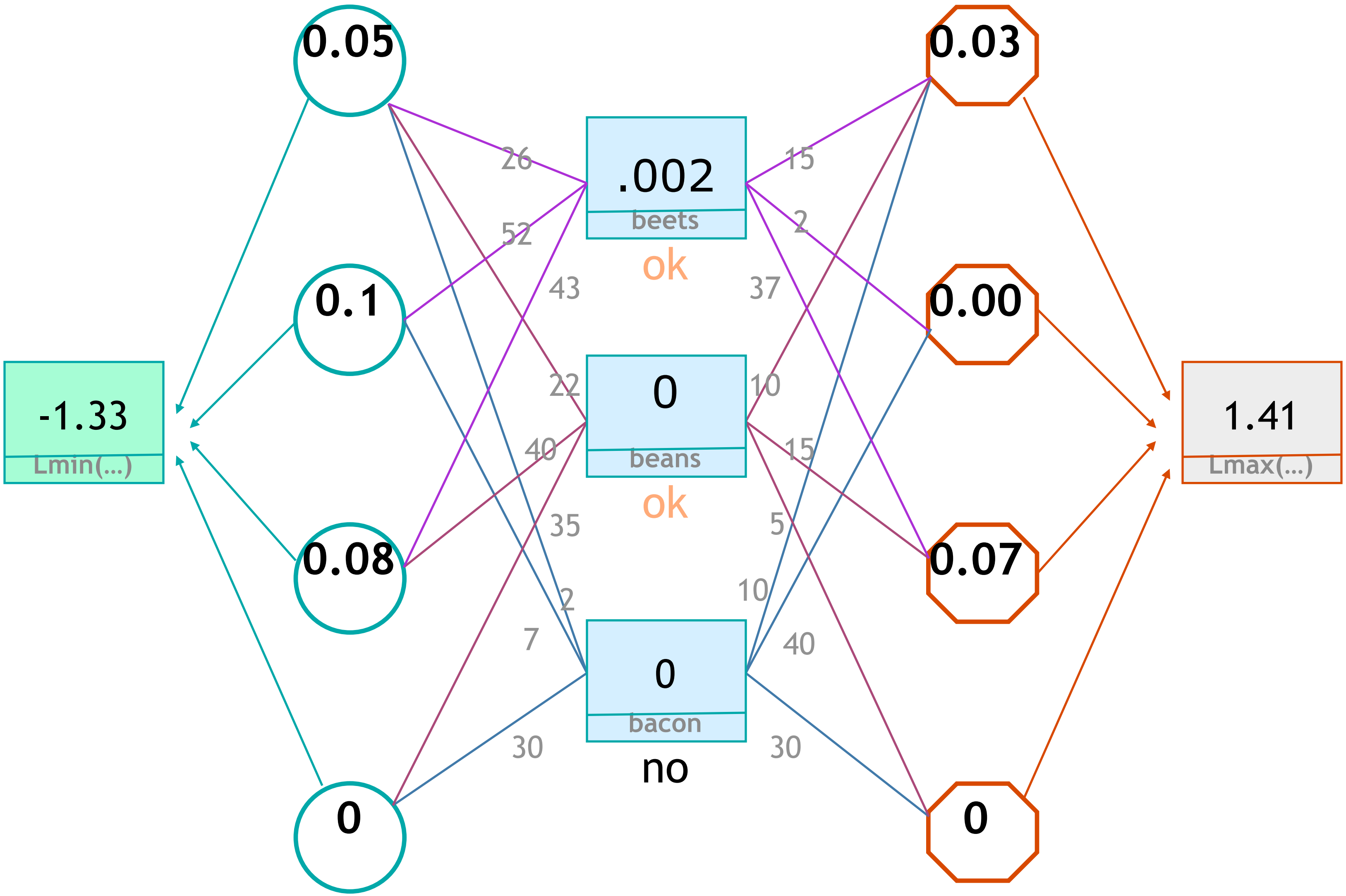


Raise beets just enough so some constraint increases by ε. ($\varepsilon = 0.1$)

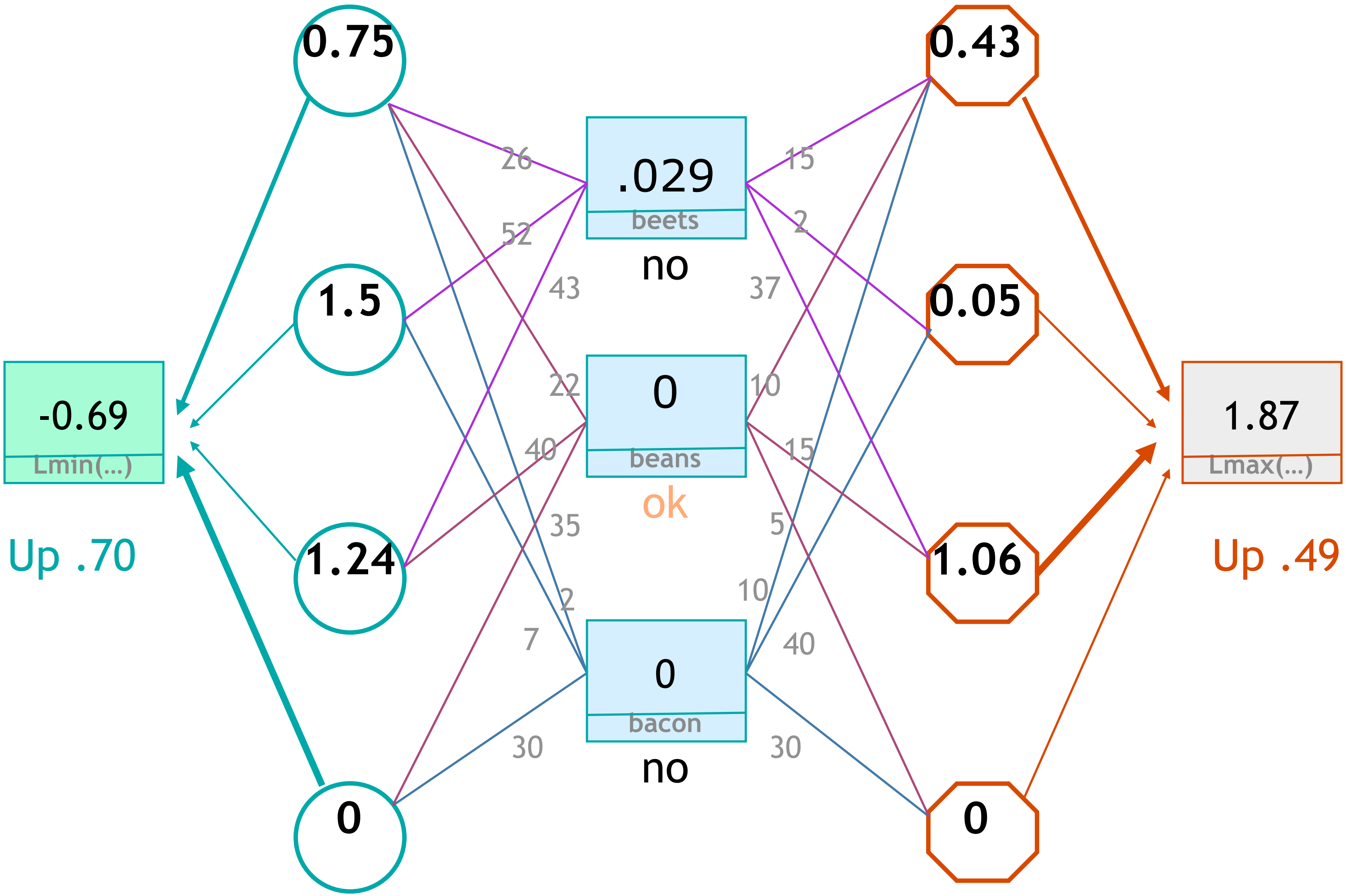


Repeat.

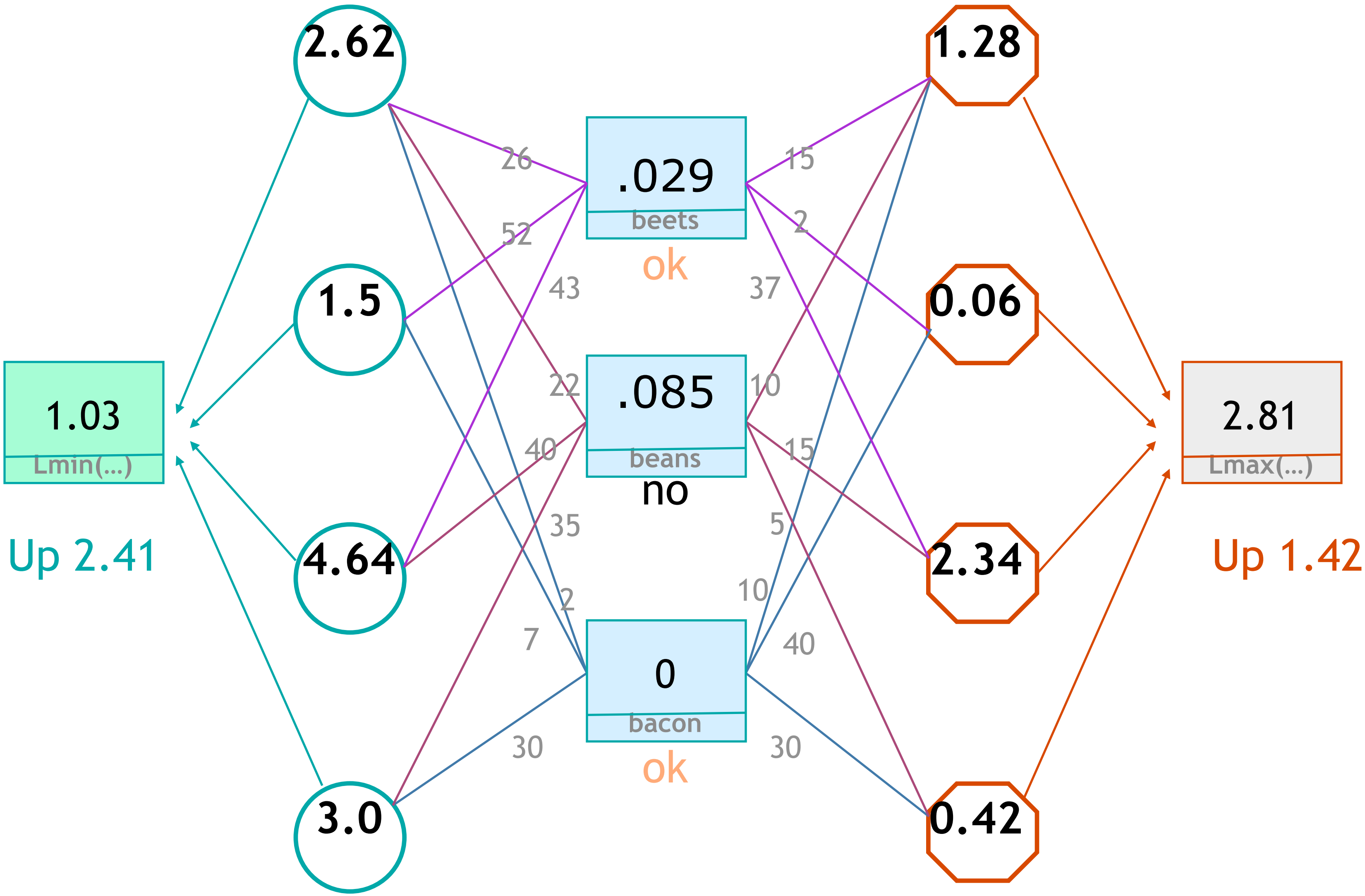
Check variables… and raise ok one so some constraint increases by ε



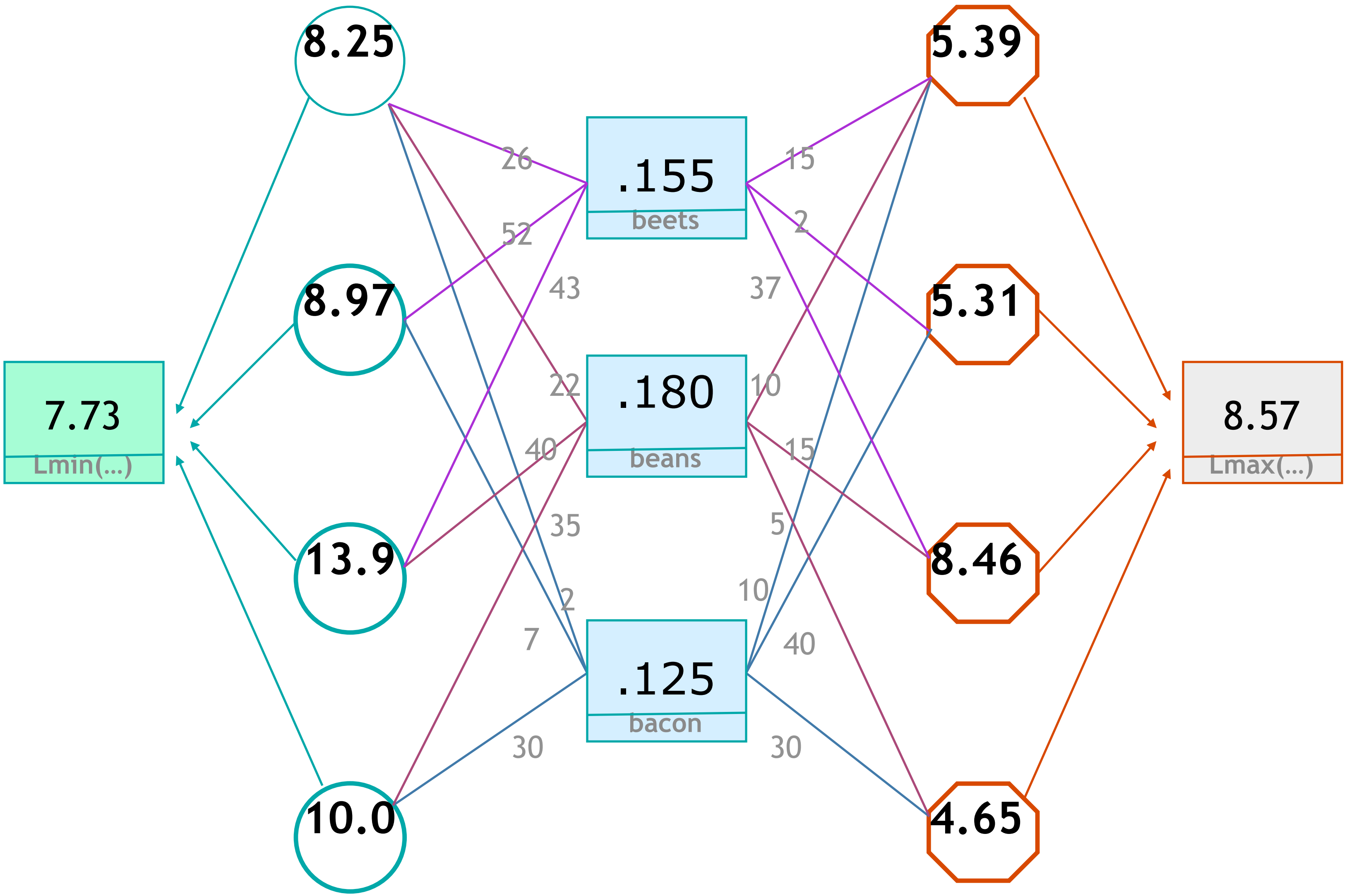
Fifteen (1/520)-servings of beets later…



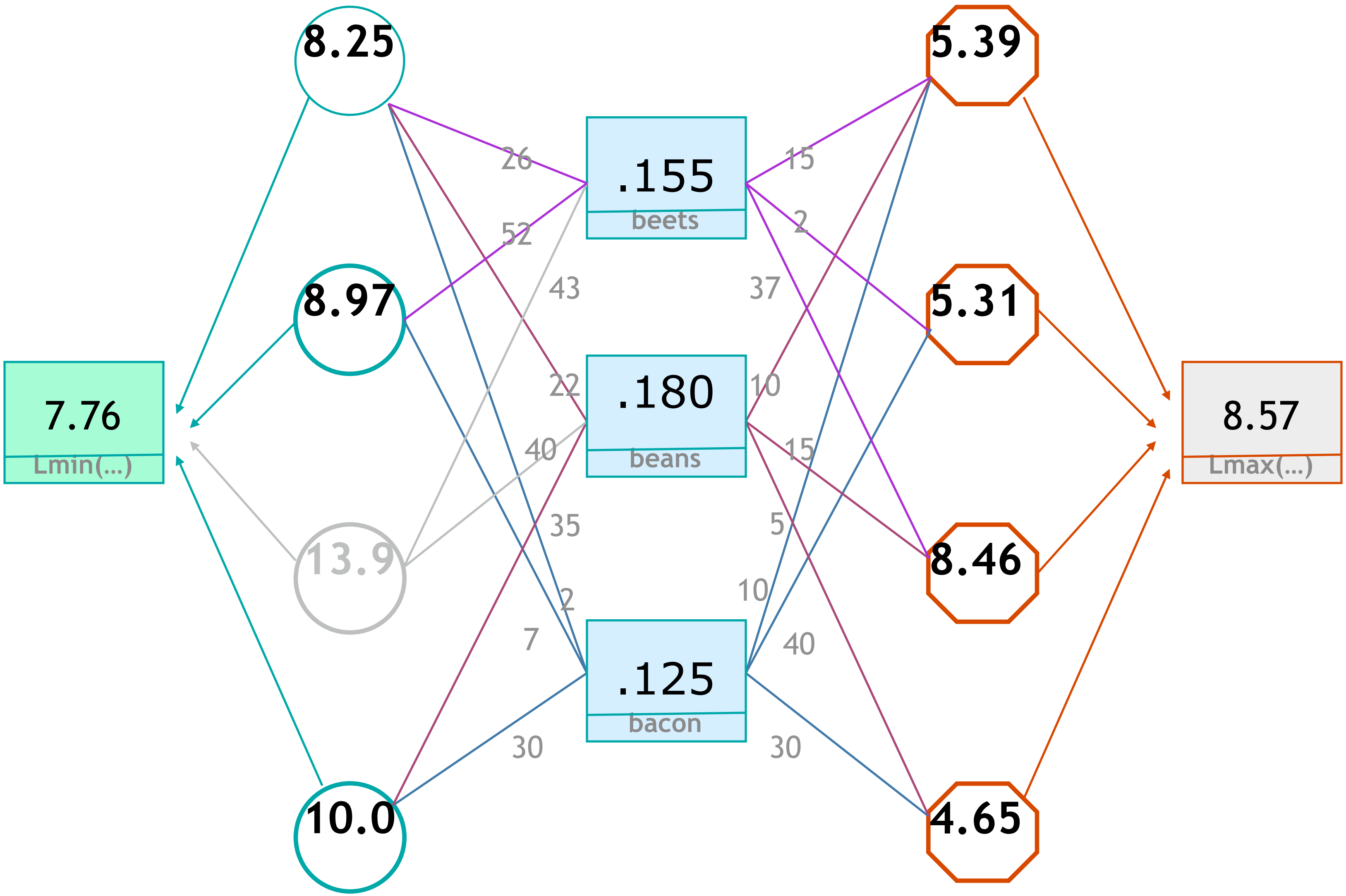
and thirty-four (1/400)-servings of beans…



After 204 rounds: (recall target = 13.8)



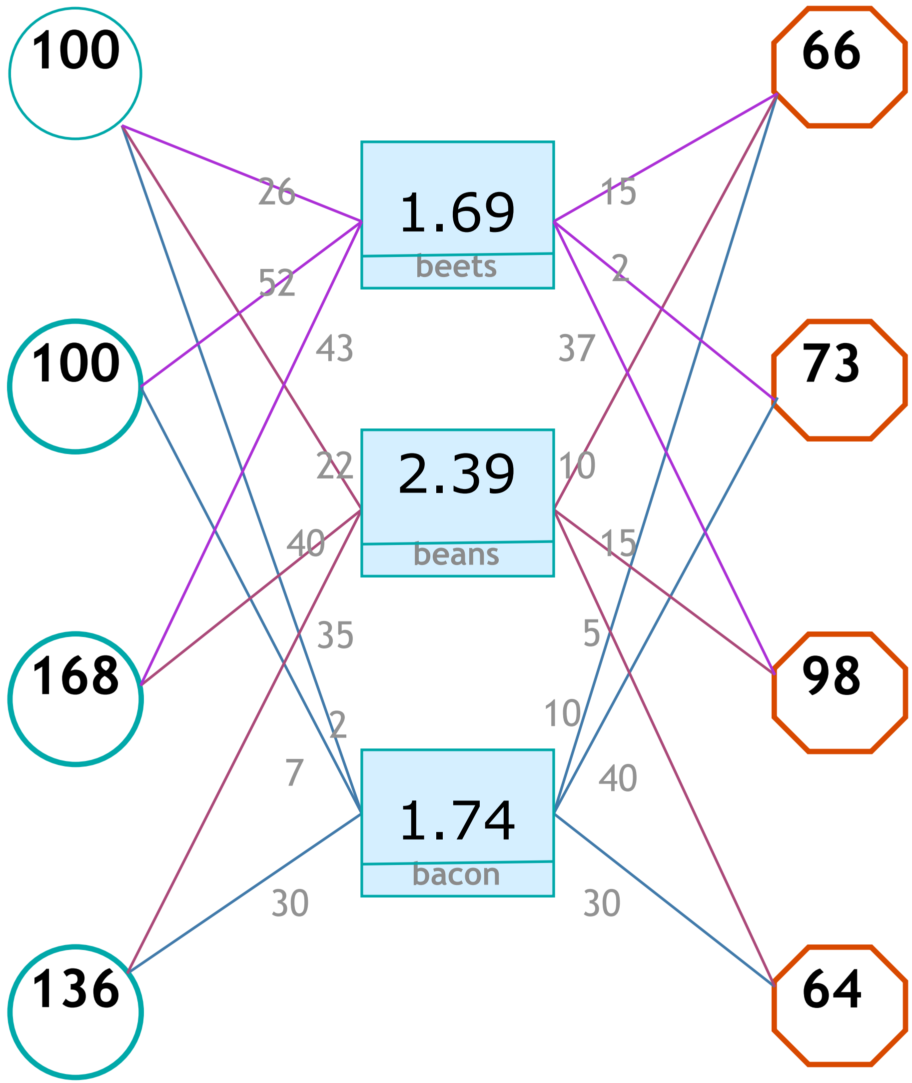
Delete covering constraints as they are satisfied



335 total rounds until lmin > target:

15 beet
34 bean
5 bacon
17 beet
12 bean
15 bacon
17 beet
12 bean
15 bacon
16 beet
12 bean
15 bacon
16 beet
12 bean
14 bacon
15 beet
13 bean
15 bacon
18 beet
9 bean
18 bacon
8 beet
10 bean

At end, scale up by 100/target:



Algorithm

$x = (0,0,\ldots,0)$

target = $\ln(m)/\varepsilon$

Until lmin > target do:

    Let $x = x + \delta$, where vector δ satisfies:

        $\delta[j] > 0$ only if ok to raise variable $x[j]$,

        and some constraint increases by ε.

    Delete any satisfied covering constraints.

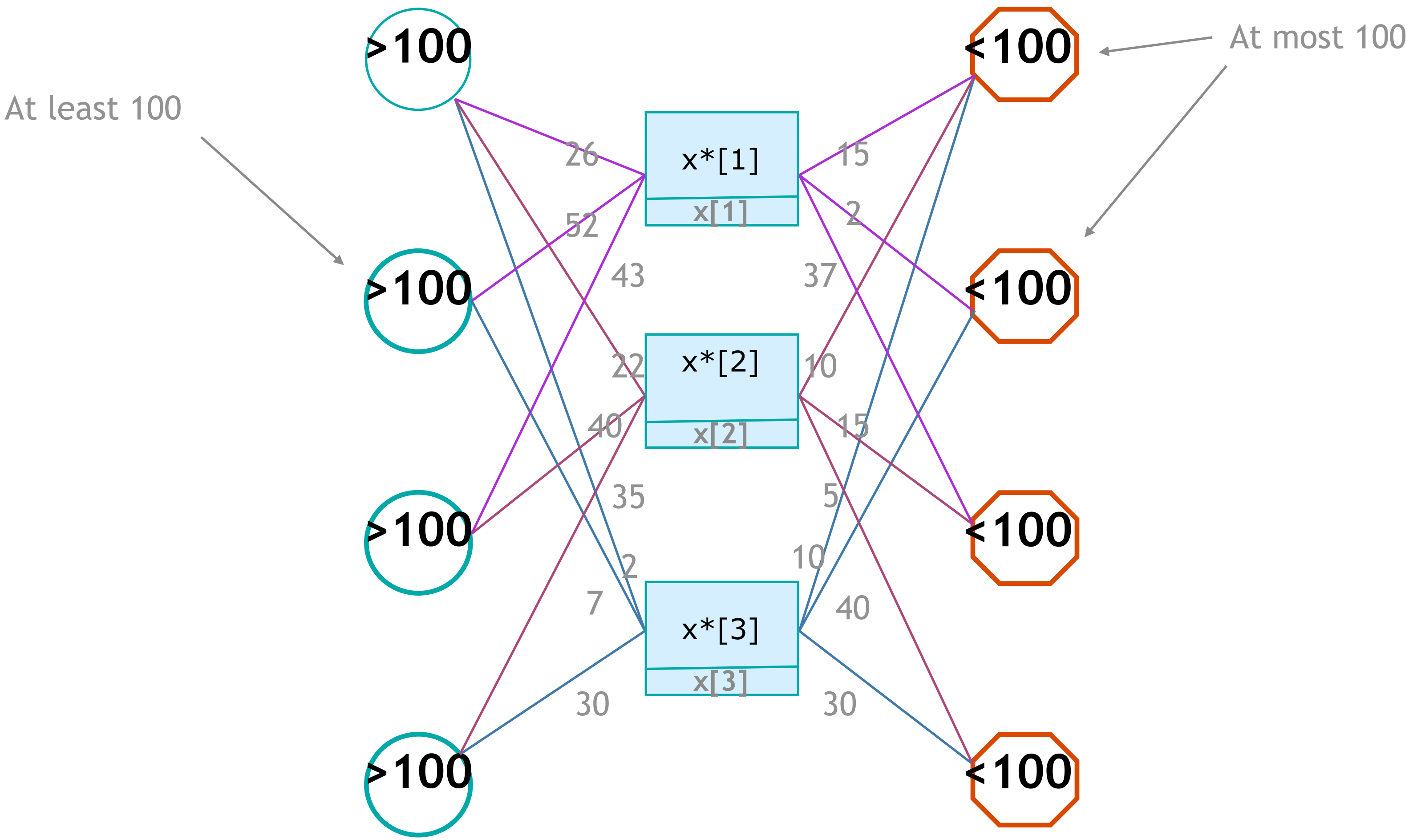
Return x, appropriately scaled.

Correctness

- Is there always a variable to raise?
- Does the algorithm terminate?

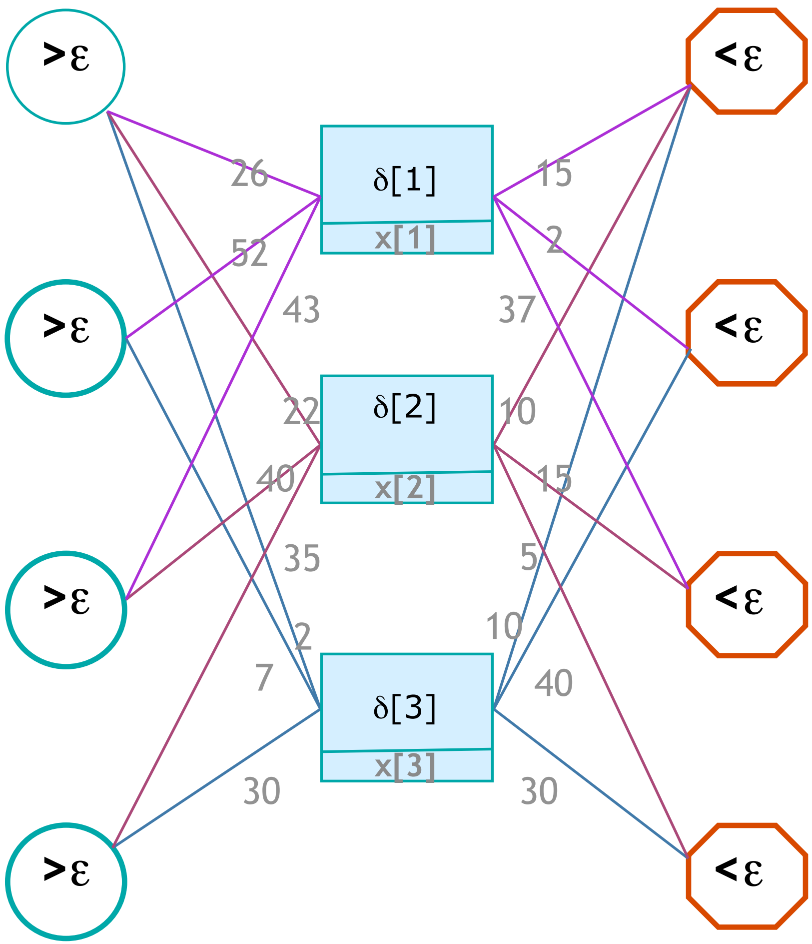
Lemma: If the constraints can be satisfied, there is always a variable to raise.

Suppose there is a feasible solution x*:



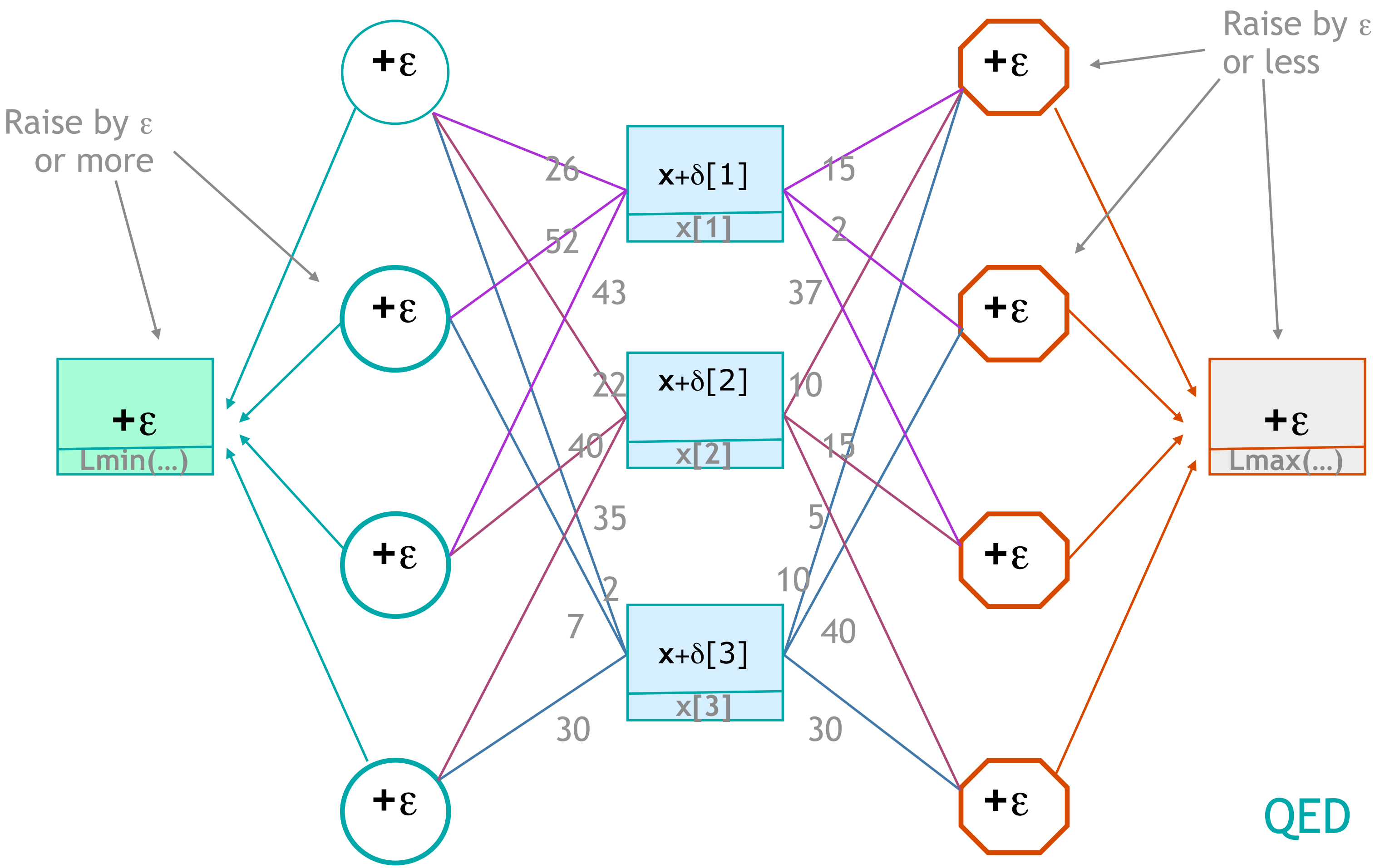
Lemma: If the constraints can be satisfied, there is always a variable to raise.

Take vector δ = ε x*/100.



Lemma: If the constraints can be satisfied, there is always a variable to raise.

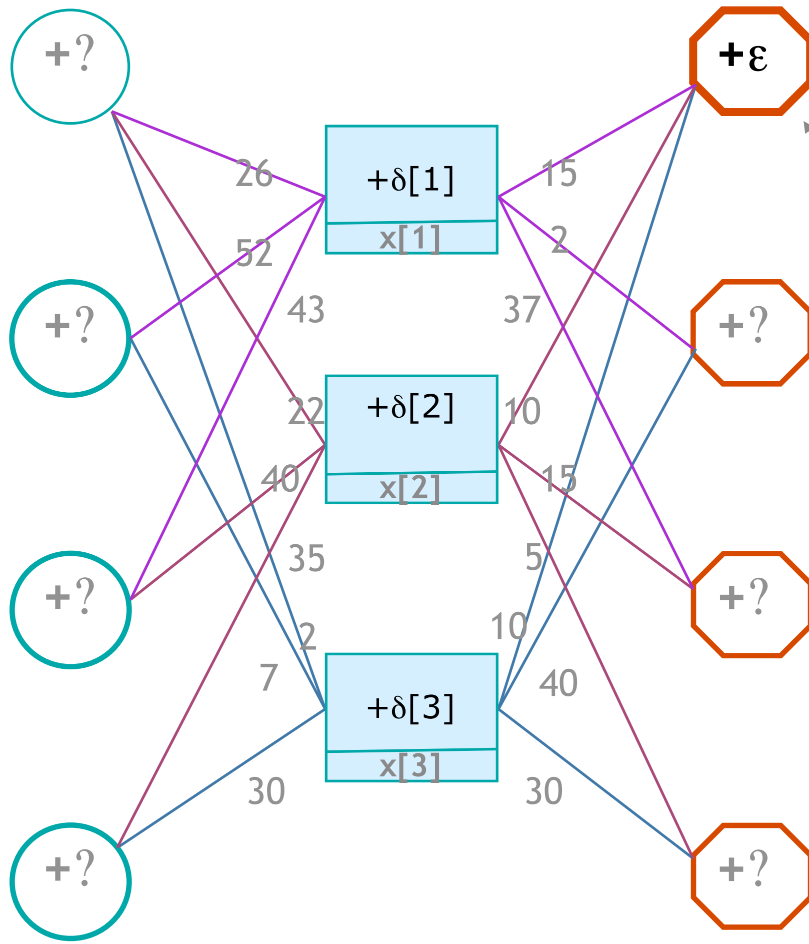
Add δ to current x. Lmin increases by as much as lmax.



QED

Lemma: There are $O(m \log(m)/\varepsilon^2)$ increments

Each increment increases at least one constraint by ε.



Constraint < O(target) at end,

so at most O(target/ε) increments per constraint.

target = O(log(m)/ε)

QED

Summary

Solutions meet constraints within $1+O(\varepsilon)$ factor.

$O(m \log(m)/\varepsilon^2)$ linear-time increments.

Faster implementations possible.