**Changes to SEIR OPT**

**Notes on Global\_seir.py** (main program)

- It can “optimize” or just “simulate”. I believe a run parameter controls this.

Simulate: For a given vaccination schedule V\_a(t), time t\_n and area where variant appears, solve the difference equations. V is reduced if needed to keep states nonnegative.

- Some of the outputs are used in the LP in Optimize:

    the number infectious I\_a(t), called “current I” or \hat{I}. Similarly, I^V\_a (t) for vaccinated.

    the time t\_n and area where variant appears

- Infection rate \alpha\_a(t) is computed from t\_n using the mutation model in the paper. The only difference between areas is that the areas where the variant does NOT occur are delayed by L.

- Currently, t\_n is an output of Simulate that is an input to Simulate in the next iteration. This isn’t necessary-see changes.

Optimize:

See Algorithm 1 and “Generating an initial solution” just before it.

1. Construct an initial “naïve” V\_a(t).

2. Simulate to find the “current I” called \hat{I}\_a(t) and I^V for vaccinated. Use them to find the time t\_n and area where variant appears.

3. Solve the LP for this \hat{I} and t\_n, finding the optimal V for this LP. The LP has “regularization constraints” keeping I close to \hat{i} so that the linear approximation is close.

If stopping criteria is not met, update the regulation tolerance \epsilon and repeat (2) and (3).

Note: The algorithm describes the Simulate using the initial V\_a(t) separately from the Simulate using V\_a(t) from Optimize. The eqns are the same. Not sure if it uses the same code.

- Infection rate \alpha\_a(t). This is computed from t\_n using the mutation model in the paper. The only difference between areas is that areas where the variant does NOT occur are delayed by T\_D.

**Planned changes**

- Change the name to **seir\_opt.py**.

- Change input files as listed below

- Keep areas and state variables as lists of strings (so that it is easy to pass them to functions) but when computing, for brevity define each state as a separate variable:

S(“area1”, t], *not* state\_variables\_simu[ar, "S", t]

- Allow the number of areas to vary. As in old code, input a list of areas as strings. Can have more areas in a file and select some.

Simulate

- Add non-negativity checks, so it is impossible to get negative variables. ~~[Done: SM]~~

- Add vaccine willingness to limit vaccinations. Add W(t) and V\*(t) computation. Section 3.3 (7)-(9). Note that V\* replaces V in difference eqns after (9).

- Add proportional reallocation option: reallocate excess vaccines to other areas proportionally to S. Without this option, vaccines that cannot be used in an area are discarded. Section 3.3 (13) and (14). Add input realloc\_flag

- Compute t\_n while running Simulate and use it to compute \alpha\_a(t) for the remaining t. This avoids using the old t\_n as an input to Simulate. Section 3.3 (10)-(12) and the steps listed: 1, 2ab.

Simulate and Optimize

- Change first day from 1 to 0, so that t = 0,…, T – 1 in all equations and Simulate uses T time steps.

- Remove travel from Simulate [Done: SM] and Optimize: new *V* is Section 3.1 (2).

- Modify testing (Section 3.4): remove input w, add input Δr = rate out of I, I^V due to testing: r^d\_a = r^d\_0 + Δr

- Change definition of p^e = proportion of transmission from vaccinated people. Affects “vector” *V* in Simulate Section 3.1 (2) [Done: SM] and Optimize *V* constraint.

- Input behavior γ\_a and use to compute α\_a using Section 3.2 (4)-(5). Use in dynamics.

- Compute initial states from vaccination rates and initial infection proportions using Section 3.1 (1) and Section 3.3 (6). Put in main program. Current method is in mid\_data.

- Add W(t+1) Section 3.3 (9) to dynamics: Simulate difference equations and Optimize constraint.

- Remove some outputs. Add Verbosity input to select amount of output.

Optimize

- Correct vacc willingness constraint Section 4 in optimization problem

- Eliminate “strict” policy restriction. Use … ≤ p^k B(t) constraints, not “=”

- Change initial vaccination schedule V\_a(t) from “proportional to population” to “donor first, then proportional to S” Section 4 (18). Note: (18) is correct even if there is no policy constraint, because then p^k = 1, so that V\_1(t) = B(t). This is used in the first Simulate.

- Change objective to Section 4 (16): Remove vaccine incentive. Add a Lagrange multiplier λ times (infection-days before t\_n).

- Change algorithm: Section 4 Algorithm 1. Some details are TBD. It uses new inputs

λ = Lagrange multiplier for infection-days

ϕ = Exploration multiplier for λ

δ = Termination tolerance

and redefines input ϵ′ = Termination tolerance for linear program. The basic idea is an outer loop that updates λ, I, and t\_n using a one-dimensional search method (quadratic approx. to last three points?) and an inner loop that solves the linear program to update V, then simulates.

- Change objective: add option of non-donor deaths, with small weight. Add input non\_donor\_deaths\_flag. This is to select better solutions from (nearly) tied solutions. TBD

- TBD: solve the quadratically constrained program (QCP) SEIR-OPT in Section 4 of new paper, not just iteratively solve the LP. With the Lagrangian change, the QCP will also have to be solved iteratively, updating λ and t\_n.

**New input files for global\_seir.py**

Currently, inputs are in

    1. mid\_seir\_data.py (for “real”, this is created by mid\_process\_df.py; for “four”, it is manually entered from output of Data Manipulation.Rmd

    2. mid\_params.py

    3. user\_input\_params.py

4. hard-coded in global\_seir.py

Note: vax\_budget.csv contains global vacc availability by day, but only for “real”, which we aren’t using.

There are several run-time prompts. I think they are:

    optimize or simulate (I can’t find this in the old code)

    number of areas

    p^k (max proportion of vaccines alloc to donor areas)

    strict policy restriction

    alpha

We will be creating input files manually for a small number of areas, similar to the “four”-area files. mid\_seir\_data is too messy, so define a new format with three input files: area data, other scenario data, and run parameters. Scenario and run parameters could be combined? Move hard-coded parameters to these files.

**Areas input file**

Description Name in paper Name Name in old code

set of areas *A* all\_areas (mid\_data)

donor area set *D* donor (string) (seir)

paper allows a set of donors, code allows one

variant area m most\_infected\_area (mid\_data)

infection-days till new variant n N (param)

population by area N\_a N N\_a (mid\_data)

was computed from initial states

initial proportion vacc by area ρ^V vax\_rates\_s (mid\_data)

initial cases per day by area ρ^I N replaces initial\_infected = I\_a(0)+I^V\_a(0) (mid data)

enter as a count ρ^I N,

divide by N to get proportion ρ^I

testing effect on rate out of I, Δr\_a delta\_r replaces tests\_per\_day

by area

behavior infection multiplier γ\_a gamma gamma was the convergence param

by area

proportion willing to be ρ\_a rho rho (params)

vaccinated, by area. Didn’t depend on area in old code

**Scenario input file** (see Tables 1 and 2)

Description Name in paper Name Name in old code

rate out of state E into I r^I r\_I (param)

rate out of state I w/o testing r\_0 r\_0 r\_d (param)

(called r^d in old paper)

rate out of state H r^R r\_R (param)

P(H | I) p^H p\_H (param)

P(H | I^V) p^H\_V p\_V\_H (param)

P(D | H) p^D p\_D (param)

initial infection rate a\_0 ALPHA\_VALUE (user)

change in infection rate Δa (was λ in old paper) delta\_a lmbda (param)

proportion of transmission p^e p\_e (param)

from a vaccinated person

proportion of transmission p^r p\_r (param)

to a vaccinated person

lag for variant to reach other areas L L (param)

time for variant to dominate T\_D T\_D (param)

Proportion of people in state I p p (param)

that have the new variant

when it is introduced

time horizon T T (user)

vaccine available day 0 B(0) B\_0 --

vaccine available as prop of B(t)/B(0), i.e., b[t] B\_t (user, by day)

that avail day 0, by day. B(t) = B\_0b(t) defined as count, not proportion

If not in file, use the default value of 1, so that B(t) = B\_0 for all t as in the old code.

**Run parameters input file** (see Table 2)

Description Name in paper Name Name in old code

verbosity of output verbosity

0: least verbose (objective value(s), variant area/day, iterations, run time, accuracy)

1: more verbose (0 plus objective by iteration, vaccinations by area/day)

2: most verbose (1 plus final state variables by area, scenario and parameters files)

optimize or simulate   -- opt\_flag --

proportional reallocation of -- realloc\_flag --

unused vacc to non-donors in simulate

non-donor deaths in objective non\_donor\_deaths\_flag --

max prop of vacc alloc to p^k p\_k (user)

donor areas

Lagrange multiplier for infection λ lambda\_0 --

(initial value)

exploration multiplier for λ ϕ phi --

exploration tolerance for I ϵ\_0 (was ϵ) exploration\_tolerance (seir)

termination tolerance for I δ\_I (was ϵ′) termination\_tolerance (seir)

termination tolerance for λ δ delta --

convergence parameter β beta gamma (seir) (was γ in old paper)

iteration limit for I -- iteration\_limit (seir)

iteration limit for λ -- iter\_lmt\_search --

Move from mid\_data to seir

set of state var’s state\_vars

initial states by area, state S\_a(0), etc. initial\_pop\_a

computed from population, vacc rate, and parameters initial\_pop\_states

Move from param to seir

decay rate of mutation model k k

computed from mutation parameters

**Command line arguments Default**

area, scenario, parameters input file names area, scenario, param

v verbosity of output (parameters file)

s optimize or simulate (parameters file)

p max prop of vacc alloc to donor areas (p\_k) (parameters file)

l initial λ (lambda\_0) (parameters file)

a initial infection rate (a\_0) (scenario file)

**Outputs**

If opt-flag = simulate:

Verbosity 0:

Write CSV file. Use Abraham’s format as much as possible.

name: sim\_[parameters file name]

state variables, actual vaccinations V\*, and willing to vaccinate W by area/day

day variant emerges (t\_n)

lag (L)

time horizon T

donor deaths D[donor, T]

total deaths sum over a of D[a, T]

total vaccinations sum over a, t of V\*[a, t]

donor vaccinations sum over t of V\*[donor, t]

variant area m

variant time t\_n (real, not integer)

Verbosity 1:

vaccinations by day/area:

V\*[area1, 0] V\*[area2, 0] …

V\*[area1, 1] V\*[area2, 1] …

Verbosity 2:

final state variables by area: S[a, T] etc.

scenario and parameters files

If opt-flag = optimize: TBD

**Search algorithm**

This search algorithm finds λ to minimize z = donor deaths. It is the outer loop in Optimize. The overall structure is in Algorithm 1. To use standard notation, let λ = x and z = f(x), so we minimize f(x).

### Search parameters

initial solution x0 = λ

exploration multiplier for λ ϕ > 1

termination tolerance for λ δ

iteration limit for λ iter\_lmt\_search

### Phase 1: Find an interval that contains optimal x by searching until f changes direction

# Initialization

fL = f(x0) #i.e., set x = x0 and call f(x). fL = f at left end pt

fR= f(ϕx0) #fR = f at right end pt

iter = 2

If (fL ≤ fR) #f is increasing, search to left

mult = 1/ϕ #x multiplier < 1

x1 = ϕx0 #1st of 3 current x values

x2 = x0 #2nd of 3 current x values

f1 = fR #f(x1)

f2 = fL #f(x2)

If (fL > fR) #f is decreasing, search to right

mult = ϕ #x multiplier > 1

x1 = x0

x2 = ϕx0

f1 = fL

f2 = fR

# Main loop

While (iter < iter\_lmt\_search)

x3 = mult\*x2 #3rd of 3 current x values

f3 = f(x3)

If (f3 > f2) go to Phase 2 #x3 is past the minimum

x1 ← x2, x2 ← x3, f1 ← f2, f2 ← f3, iter ← iter + 1 #shift x’s for next iteration

### Phase 2: Golden ratio search on interval [a, b] with check for unimin

# Initialization

If (x1 < x3) a = x1, b = x3, fa = f1, fb = f3

If (x1 > x3) a = x3, b = x1, fa = f3, fb = f1

x = a + 0.618 \* (b - a) #current larger x value

y = b - 0.618 \* (b - a) #current smaller x value

fx = f(x)

fy = f(y)

iter = iter + 2 #account for two f evaluations

# Main loop

While (|fx – fy| > δ and iter < iter\_lmt\_search)

if fx > fy # minimum is in [a,x]

b, x, fx = (x, y, fy)

y = b - 0.618 \* (b - a)

fy = f(y)

else # minimum is in [y,b]

a, y, fy = (y, x, fx)

x = a + 0.618 \* (b - a)

fx = f(x)

If (fy < fx and fx > fb) print “Warning: f is not unimin” [print all 4 x values and their f values]

If (fy > fx and fa < fy) print “Warning: f is not unimin” [print all 4 x values and their f values]

If (fy ≤ fx) return optimal value fy and optimal solution y

If (fy > fx) return optimal value fx and optimal solution x