PLEASE IGNORE THIS THIS IS TEMP FOR GETTING LATEX MARKDOWN

[0] [0] Jonathan Woollett-light
[2020-12-01 07:51:15]
[python]
[https://math.stackexchange.com/questions/3929868/please-ignore-this-this-is-temp-for-getting-latex-markdown]

Production

Production is very simple, simply component-wise multiplying a bunch of vectors.

Only S_{count} (number of pops of a species) in the equation is not a vector, rather being an integer scalar.

All vectors are of floating point values and of the length 28 (the number of technical resources in an unmodded game, please see the resource file [1])

$$production = \sum_{empire} empire_{mod} \odot \sum_{planet} planet_{mod} \odot \sum_{species} species_{count} \cdot species_{mod}$$

Current system is fast and simple, but could be better, changing from iterating over vectors to using matricies.

Optimization

Optimisation is a lot more complicated than production. There are 2 types:

1. Intra-planetary/Planetary optimisation: Pops allocated to jobs within planets to maximise production values of planets.
2. Inter-planetary/Empire optimisation: Pops allocated to jobs across planets to maximise production value of empire.

What is our loss function?

I use the overall production of all resources component-wise multiplied by their market values then summed as the value we are maximising.

Allows evaluating the resource production of anything (empire/planet/job/worker) into a single scalar value.

An example:

A pop works a job producing 2 minerals, 1 energy, 0.5 exotic gases and 0.25 dark matter.

The production:

$[1, 2, \ldots, 0.5, \ldots, 0.25, \ldots]$

The resource values:

$[1, 1, \ldots, 10, \ldots, 20, \ldots]$

We component-wise multiply these together:

$[1, 2, \ldots, 0.5, \ldots, 0.25, \ldots] \odot [1, 1, \ldots, 10, \ldots, 20, \ldots] = [1, 2, \ldots, 5, \ldots, 5, \ldots]$

And then sum the result:

$sum\{[1, 2, \ldots, 5, \ldots, 5, \ldots]\} = 13$

13 is our production value.

How do we maximise this value?

Now this is the tricky part.

Our end goal is to calculate the value of having each species in each job (the production value of 1 member of a species working said job).

After that, assigning best species to best jobs is fairly trivial.

0. Definitions:

(Here r, j, s and p are the number of resources, jobs, species and planets respectively)

- J_{prod}: Job productions: $j \times r$ matrix of what quantities of each resource every job produces.
- J_{mod}: Job modifiers: $j \times r$ matrix of empire wide modifiers to specific jobs (e.g. +5% unity from nobles).
- S_{mod}: Species modifiers: $s \times r$ matrix of how species traits affect their production (e.g. +10% minerals).
- S_{emod}: Empire specific species modifiers: $s \times r$ matrix of how empire policies affect species production (e.g. residence type may give +5% resources)
- S_{employ}: Species employability: $s \times j$ matrix of 0s and 1s describing which jobs each species can work, 0=cannot work, 1=can work (e.g. nerve stapled prevents working advanced jobs).
- $S_{eemploy}$: Empire specific species employability: $s \times j$ matrix of 0s and 1s describing which jobs each species can work in a specific empire (e.g. slaves cannot work advanced jobs).
- V: Resource values: Length r vector of how much we value each resource.
- E_{mod}: Empire modifier: Length r vector of empire wide modifiers to resources (e.g. +10% minerals per month)
- P_{mod}: Planet resource modifiers: $p \times r$ matrix of planet wide modifiers to resources (e.g. +5 energy)

1. Calculating intermediary values:

Now we descend into the algorithm.

- $J_{adjust} = J_{prod} \odot J_{mod}$: Job adjusted: Standard production of every job in a specific empire.
- $S_{cmod} = S_{mod} \odot S_{emod}$: Compressed species modifiers: Modifiers for specific species in specific empire.
- $S_{cemploy} = S_{employ} \odot S_{eemploy}$: Compressed species employability: Employability for specific species in specific empire.
- $V_e = V \odot E_{mod}$: Empire market values: Relative market values accounting for specific empire global modifiers.

2. Calculating job priorities

Repl post I made to illustrate the more complex calculations [2] (from S_{jrv} onwards).

This is the most complex bit, I'll try my best to make it simple.

1. $J_{rv} = J_{adjust} \odot J_{j,1} V_e$: Job resource values: $j \times r$ matrix formed from the component-wise multiplication of each row of J_{adjust} by V_e. Here $J_{j,1}$ is a $j \times 1$ matrix of 1s. From how much of each resource each job produces ($J_{j,1}$), to the values of each resource each job produces (J_{rv}).
2. $(S_{jrv})_{j,s,r} = (J_{rv})_{j,r}(S_{cmod})_{s,r}$: Species job resoure values: $j \times s \times r$ tensor of the values of resources produced by jobs by specific species. For every job and species combination gives the values of resources produced (`species_jobs=np.einsum("jr,sr->jsr",jobs,species)`)
3. $(S_{sejrv})_{s,j,r} = (S_{jrv})_{j,s,r}(S_{cemploy})_{s,j}$: Species employable job resource values: $j \times s \times r$ tensor of the values of resources produced by jobs by specific species, with resource for jobs a species cannot work, set to 0. For every job and species combination gives the values of resources produced, with the value being 0 when the species cannot work the work. In simply words, the values of resources produced by each species in each job. (`species_employable_jobs = np.einsum("sjr,sj->sjr",species_jobs,species_employability)`).
4. $(P_{prior})_{p,s,j} = (S_{jrv})_{j,s,r}(P_{mod})_{p,r}$: Planet species job values: $p \times j \times s$ tensor representing the value of each species in each job on each planet, ultimately, the values which we can order to get the

priorities of where pops should be (`planet_species_jobs = np.einsum("sjr,pr->psj",species_employable_jobs,planets)`).

(I don't know how to properly write einstein summations, so please give feedback if I'm wrong here)

Whooh, that was quite a bit of shit going on there. I'm not 100% on it myself.

3. Assigning jobs

Here we are past most of the maths, now we just got some more technical stuff.

Now this part will differ depending if we are going empire-wide or planetary optimization, although either is fairly straightforward at this point.

When doing planetary optimisation, for each planet:

1. Flatten the matrix $(P_{prior})_p$ into a list of tuples with the original indexes and the values `(i,j,val)`.
2. Sort the list by `val`.
3. For each item in the list: `if (val<0) {break;} else {Assign max available pops of species j to job i}`.

We `break` when `val<0` since it is imaginable some pops are so bad in some jobs that the amount of resources the job consumes is more significant than the amount that the pop is able to produce.

[1] https://github.com/JonathanWoollett-Light/stellaris-performance-test/blob/master/resources.md
[2] https://repl.it/talk/share/Stellaris-job-priorities/83340