

Job Search IV: Modeling Career Choice

Jesse Perla, Thomas J. Sargent and John Stachurski

December 4, 2020

1 Contents

- Overview [2](#)
- Model [3](#)
- Exercises [4](#)
- Solutions [5](#)

2 Overview

Next we study a computational problem concerning career and job choices.

The model is originally due to Derek Neal [\[2\]](#).

This exposition draws on the presentation in [\[1\]](#), section 6.5.

2.1 Model features

- Career and job within career both chosen to maximize expected discounted wage flow.
- Infinite horizon dynamic programming with two state variables.

2.2 Setup

```
In [1]: using InstantiateFromURL
        # optionally add arguments to force installation: instantiate = true,
        ↪precompile = true
        github_project("QuantEcon/quantecon-notebooks-julia", version = "0.8.0")
```

```
In [2]: using LinearAlgebra, Statistics
```

3 Model

In what follows we distinguish between a career and a job, where

- a *career* is understood to be a general field encompassing many possible jobs, and

- a *job* is understood to be a position with a particular firm

For workers, wages can be decomposed into the contribution of job and career

- $w_t = \theta_t + \epsilon_t$, where
 - θ_t is contribution of career at time t
 - ϵ_t is contribution of job at time t

At the start of time t , a worker has the following options

- retain a current (career, job) pair (θ_t, ϵ_t) — referred to hereafter as “stay put”
- retain a current career θ_t but redraw a job ϵ_t — referred to hereafter as “new job”
- redraw both a career θ_t and a job ϵ_t — referred to hereafter as “new life”

Draws of θ and ϵ are independent of each other and past values, with

- $\theta_t \sim F$
- $\epsilon_t \sim G$

Notice that the worker does not have the option to retain a job but redraw a career — starting a new career always requires starting a new job.

A young worker aims to maximize the expected sum of discounted wages.

$$\mathbb{E} \sum_{t=0}^{\infty} \beta^t w_t \tag{1}$$

subject to the choice restrictions specified above.

Let $V(\theta, \epsilon)$ denote the value function, which is the maximum of (1) over all feasible (career, job) policies, given the initial state (θ, ϵ) .

The value function obeys

$$V(\theta, \epsilon) = \max\{I, II, III\},$$

where

$$\begin{aligned} I &= \theta + \epsilon + \beta V(\theta, \epsilon) \\ II &= \theta + \int \epsilon' G(d\epsilon') + \beta \int V(\theta, \epsilon') G(d\epsilon') \\ III &= \int \theta' F(d\theta') + \int \epsilon' G(d\epsilon') + \beta \int \int V(\theta', \epsilon') G(d\epsilon') F(d\theta') \end{aligned} \tag{2}$$

Evidently I , II and III correspond to “stay put”, “new job” and “new life”, respectively.

3.1 Parameterization

As in [1], section 6.5, we will focus on a discrete version of the model, parameterized as follows:

- both θ and ϵ take values in the set $\text{linspace}(\mathbf{0}, \mathbf{B}, \mathbf{N})$ — an even grid of N points between 0 and B inclusive
- $N = 50$
- $B = 5$

- $\beta = 0.95$

The distributions F and G are discrete distributions generating draws from the grid points `linspace(0, B, N)`.

A very useful family of discrete distributions is the Beta-binomial family, with probability mass function

$$p(k | n, a, b) = \binom{n}{k} \frac{B(k + a, n - k + b)}{B(a, b)}, \quad k = 0, \dots, n$$

Interpretation:

- draw q from a β distribution with shape parameters (a, b)
- run n independent binary trials, each with success probability q
- $p(k | n, a, b)$ is the probability of k successes in these n trials

Nice properties:

- very flexible class of distributions, including uniform, symmetric unimodal, etc.
- only three parameters

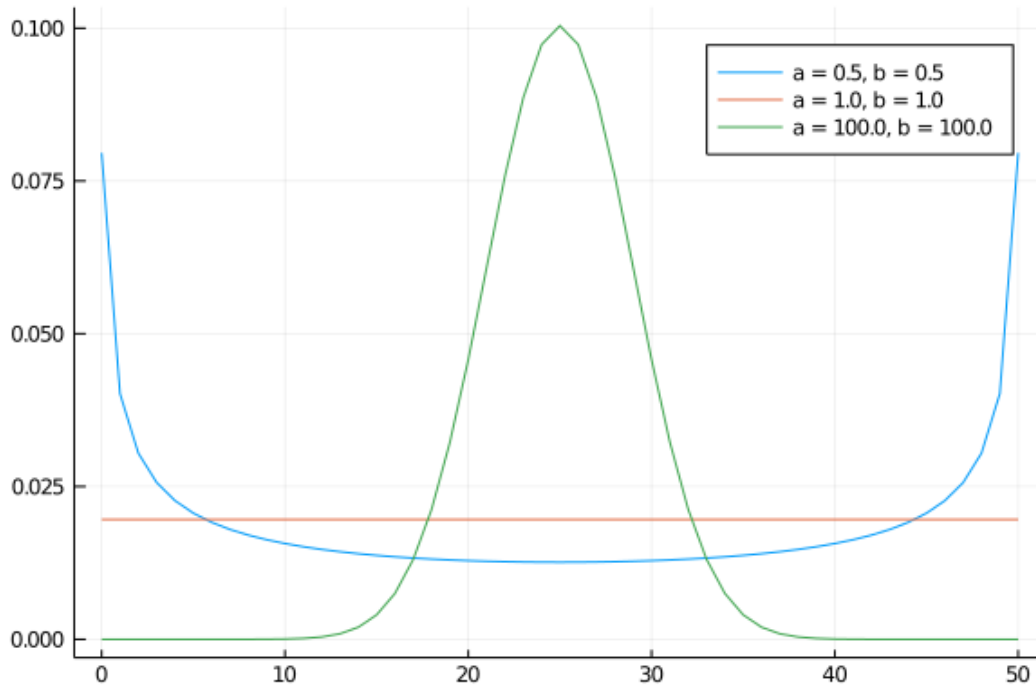
Here's a figure showing the effect of different shape parameters when $n = 50$.

```
In [3]: using Plots, QuantEcon, Distributions
gr(fmt=:png);

n = 50
a_vals = [0.5, 1, 100]
b_vals = [0.5, 1, 100]

plt = plot()
for (a, b) in zip(a_vals, b_vals)
    ab_label = "a = $a, b = $b"
    dist = BetaBinomial(n, a, b)
    plot!(plt, 0:n, pdf.(dist, support(dist)), label = ab_label)
end
plt
```

Out[3]:



Implementation:

The code for solving the DP problem described above is found below:

```
In [4]: function CareerWorkerProblem(; $\beta$  = 0.95,
    B = 5.0,
    N = 50,
    F_a = 1.0,
    F_b = 1.0,
    G_a = 1.0,
    G_b = 1.0)

     $\theta$  = range(0, B, length = N)
     $\epsilon$  = copy( $\theta$ )
    dist_F = BetaBinomial(N-1, F_a, F_b)
    dist_G = BetaBinomial(N-1, G_a, G_b)
    F_probs = pdf.(dist_F, support(dist_F))
    G_probs = pdf.(dist_G, support(dist_G))
    F_mean = sum( $\theta$  .* F_probs)
    G_mean = sum( $\epsilon$  .* G_probs)
    return ( $\beta$  =  $\beta$ , N = N, B = B,  $\theta$  =  $\theta$ ,  $\epsilon$  =  $\epsilon$ ,
        F_probs = F_probs, G_probs = G_probs,
        F_mean = F_mean, G_mean = G_mean)
end

function update_bellman!(cp, v, out; ret_policy = false)

    # new life. This is a function of the distribution parameters and is
    # always constant. No need to recompute it in the loop
    v3 = (cp.G_mean + cp.F_mean .* cp. $\beta$  .*
        cp.F_probs' * v * cp.G_probs)[1] # do not need 1 element array

    for j in 1:cp.N
        for i in 1:cp.N
            # stay put

```

```

v1 = cp.θ[i] + cp.ϵ[j] + cp.β * v[i, j]

# new job
v2 = (cp.θ[i] .+ cp.G_mean .+ cp.β .*
      v[i, :]' * cp.G_probs)[1] # do not need a single element

↪array

    if ret_policy
        if v1 > max(v2, v3)
            action = 1
        elseif v2 > max(v1, v3)
            action = 2
        else
            action = 3
        end
        out[i, j] = action
    else
        out[i, j] = max(v1, v2, v3)
    end
end
end
end

function update_bellman(cp, v; ret_policy = false)
    out = similar(v)
    update_bellman!(cp, v, out, ret_policy = ret_policy)
    return out
end

function get_greedy!(cp, v, out)
    update_bellman!(cp, v, out, ret_policy = true)
end

function get_greedy(cp, v)
    update_bellman(cp, v, ret_policy = true)
end

```

Out[4]: get_greedy (generic function with 1 method)

The code defines

- a named tuple **CareerWorkerProblem** that
 - encapsulates all the details of a particular parameterization
 - implements the Bellman operator T

In this model, T is defined by $Tv(\theta, \epsilon) = \max\{I, II, III\}$, where I , II and III are as given in (2), replacing V with v .

The default probability distributions in **CareerWorkerProblem** correspond to discrete uniform distributions (see [the Beta-binomial figure](#)).

In fact all our default settings correspond to the version studied in [1], section 6.5.

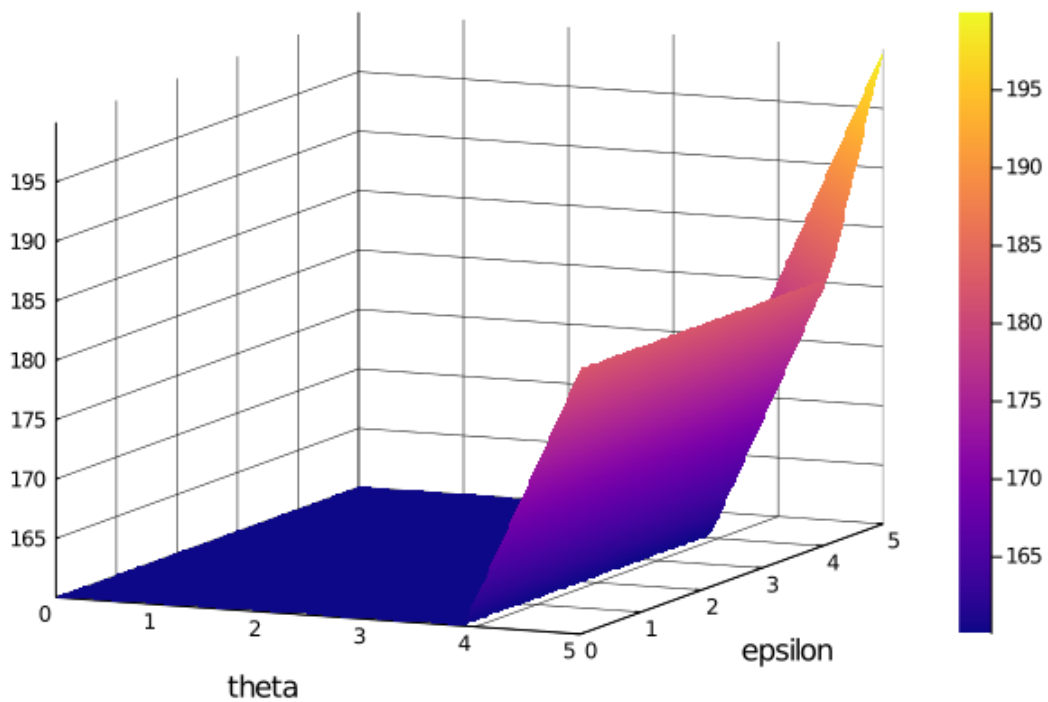
Hence we can reproduce figures 6.5.1 and 6.5.2 shown there, which exhibit the value function and optimal policy respectively.

Here's the value function

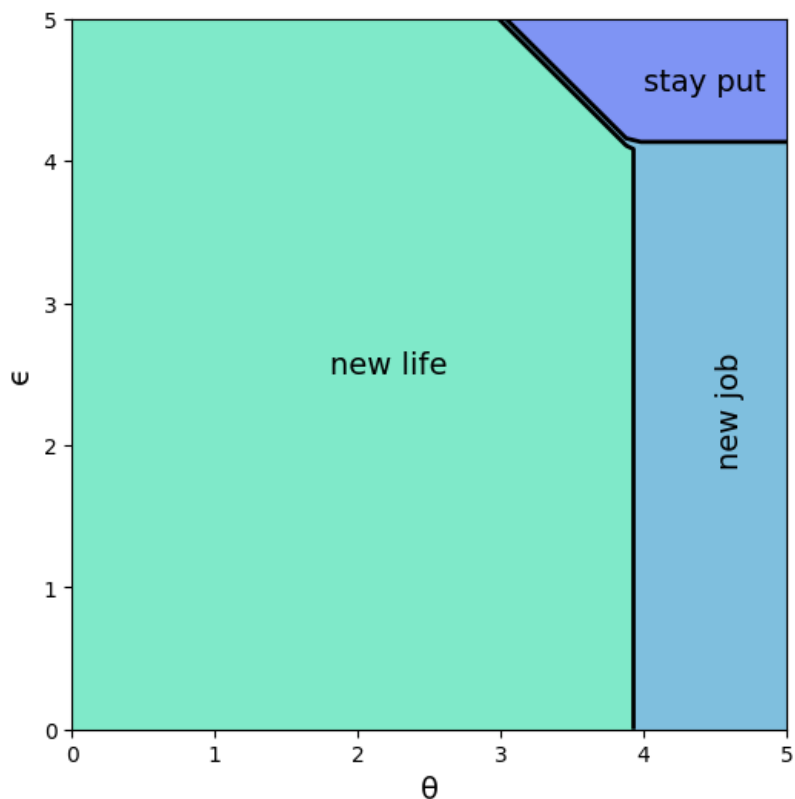
```
In [5]: wp = CareerWorkerProblem()
v_init = fill(100.0, wp.N, wp.N)
func(x) = update_bellman(wp, x)
v = compute_fixed_point(func, v_init, max_iter = 500, verbose = false)

plot(linetype = :surface, wp.θ, wp.ϵ, transpose(v), xlabel="theta",
ylabel="epsilon",
seriescolor=:plasma, gridalpha = 1)
```

Out[5]:



The optimal policy can be represented as follows (see [Exercise 3](#) for code).



Interpretation:

- If both job and career are poor or mediocre, the worker will experiment with new job and new career.
- If career is sufficiently good, the worker will hold it and experiment with new jobs until a sufficiently good one is found.
- If both job and career are good, the worker will stay put.

Notice that the worker will always hold on to a sufficiently good career, but not necessarily hold on to even the best paying job.

The reason is that high lifetime wages require both variables to be large, and the worker cannot change careers without changing jobs.

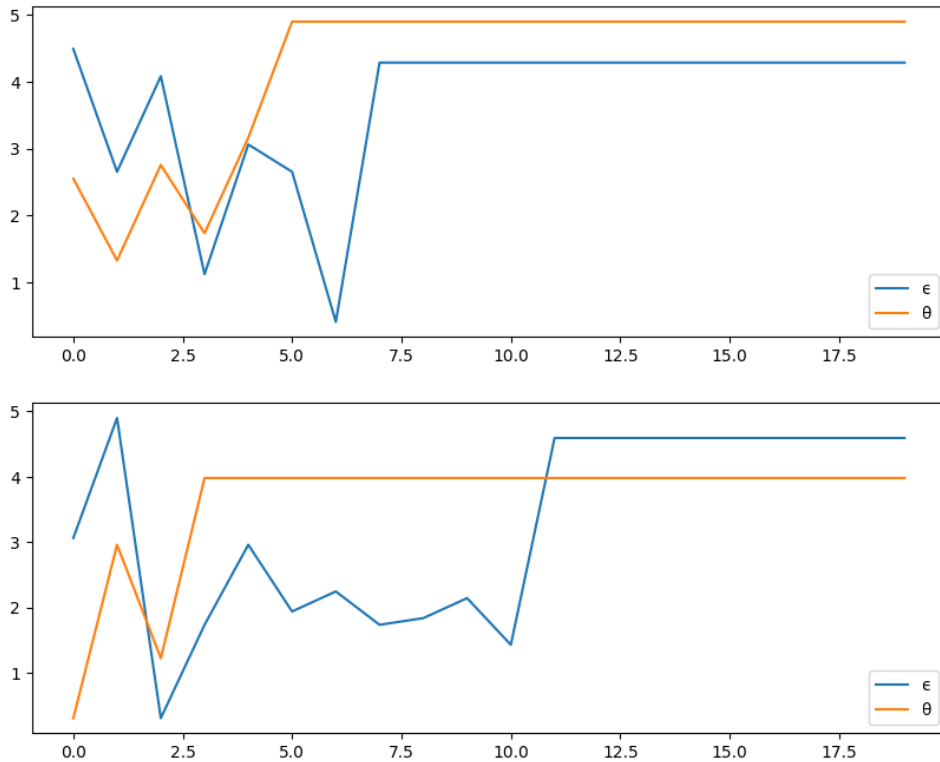
- Sometimes a good job must be sacrificed in order to change to a better career.

4 Exercises

4.1 Exercise 1

Using the default parameterization in the `CareerWorkerProblem`, generate and plot typical sample paths for θ and ϵ when the worker follows the optimal policy.

In particular, modulo randomness, reproduce the following figure (where the horizontal axis represents time)



Hint: To generate the draws from the distributions F and G , use the type [DiscreteRV](#).

4.2 Exercise 2

Let's now consider how long it takes for the worker to settle down to a permanent job, given a starting point of $(\theta, \epsilon) = (0, 0)$.

In other words, we want to study the distribution of the random variable

$$T^* := \text{the first point in time from which the worker's job no longer changes}$$

Evidently, the worker's job becomes permanent if and only if (θ_t, ϵ_t) enters the "stay put" region of (θ, ϵ) space.

Letting S denote this region, T^* can be expressed as the first passage time to S under the optimal policy:

$$T^* := \inf\{t \geq 0 \mid (\theta_t, \epsilon_t) \in S\}$$

Collect 25,000 draws of this random variable and compute the median (which should be about 7).

Repeat the exercise with $\beta = 0.99$ and interpret the change.

4.3 Exercise 3

As best you can, reproduce [the figure showing the optimal policy](#).

Hint: The `get_greedy()` method returns a representation of the optimal policy where values 1, 2 and 3 correspond to “stay put”, “new job” and “new life” respectively. Use this and the plots functions (e.g., `contour`, `contour!`) to produce the different shadings.

Now set `G_a = G_b = 100` and generate a new figure with these parameters. Interpret.

5 Solutions

5.1 Exercise 1

```
In [6]: wp = CareerWorkerProblem()

function solve_wp(wp)
    v_init = fill(100.0, wp.N, wp.N)
    func(x) = update_bellman(wp, x)
    v = compute_fixed_point(func, v_init, max_iter = 500, verbose = false)
    optimal_policy = get_greedy(wp, v)
    return v, optimal_policy
end

v, optimal_policy = solve_wp(wp)

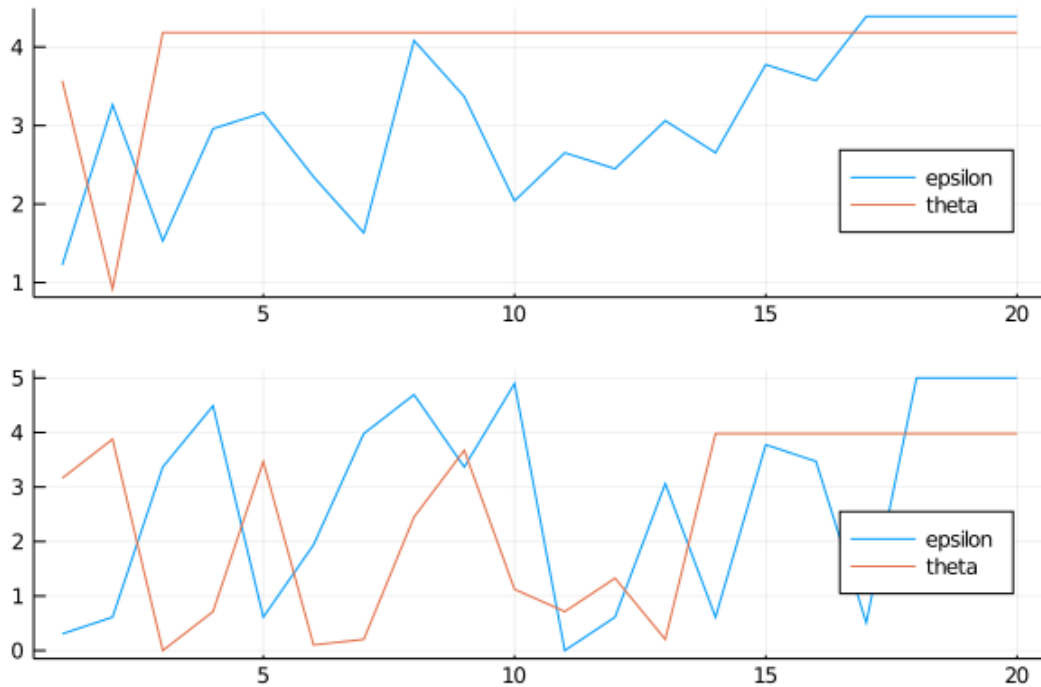
F = DiscreteRV(wp.F_probs)
G = DiscreteRV(wp.G_probs)

function gen_path(T = 20)
    i = j = 1
    θ_ind = Int[]
    ε_ind = Int[]

    for t=1:T
        # do nothing if stay put
        if optimal_policy[i, j] == 2 # new job
            j = rand(G)[1]
        elseif optimal_policy[i, j] == 3 # new life
            i, j = rand(F)[1], rand(G)[1]
        end
        push!(θ_ind, i)
        push!(ε_ind, j)
    end
    return wp.θ[θ_ind], wp.ε[ε_ind]
end

plot_array = Any[]
for i in 1:2
    θ_path, ε_path = gen_path()
    plt = plot(ε_path, label="epsilon")
    plot!(plt, θ_path, label="theta")
    plot!(plt, legend=:bottomright)
    push!(plot_array, plt)
end
plot(plot_array..., layout = (2,1))
```

Out[6]:



5.2 Exercise 2

The median for the original parameterization can be computed as follows

```
In [7]: function gen_first_passage_time(optimal_policy)
    t = 0
    i = j = 1
    while true
        if optimal_policy[i, j] == 1 # Stay put
            return t
        elseif optimal_policy[i, j] == 2 # New job
            j = rand(G)[1]
        else # New life
            i, j = rand(F)[1], rand(G)[1]
        end
        t += 1
    end
end

M = 25000
samples = zeros(M)
for i in 1:M
    samples[i] = gen_first_passage_time(optimal_policy)
end
print(median(samples))
```

7.0

To compute the median with $\beta = 0.99$ instead of the default value $\beta = 0.95$, replace `wp=CareerWorkerProblem()` with `wp=CareerWorkerProblem($\beta=0.99$)`.

The medians are subject to randomness, but should be about 7 and 14 respectively. Not surprisingly, more patient workers will wait longer to settle down to their final job.

```
In [8]: wp2 = CareerWorkerProblem( $\beta=0.99$ )

v2, optimal_policy2 = solve_wp(wp2)

samples2 = zeros(M)
for i in 1:M
    samples2[i] = gen_first_passage_time(optimal_policy2)
end
print(median(samples2))
```

14.0

5.3 Exercise 3

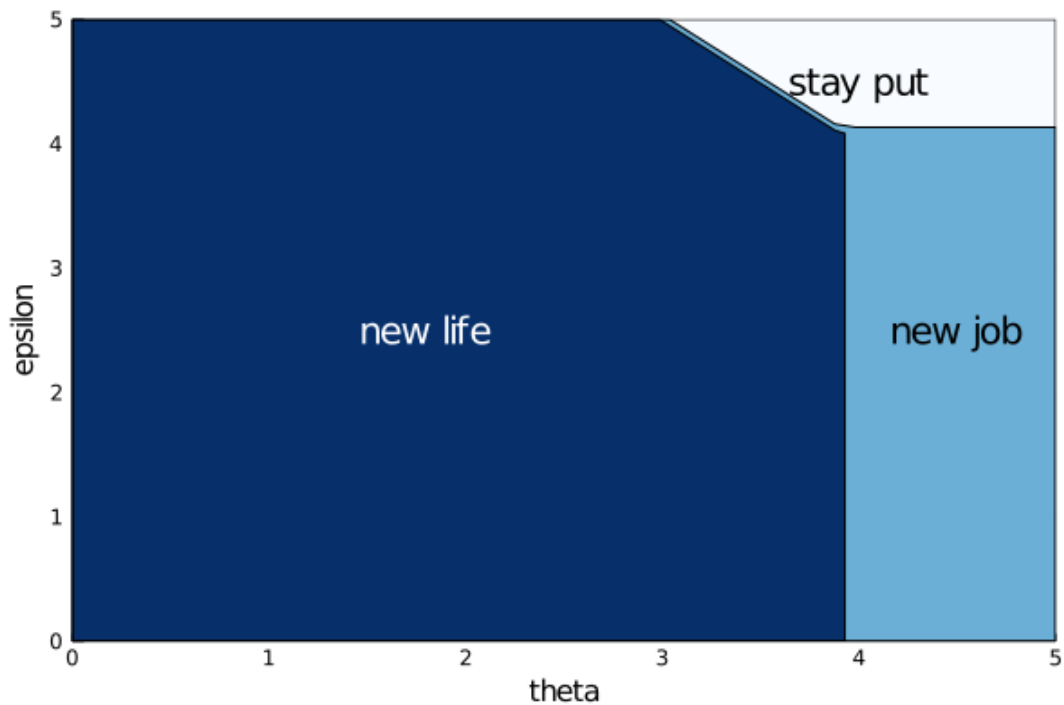
Here's the code to reproduce the original figure

```
In [9]: wp = CareerWorkerProblem();
v, optimal_policy = solve_wp(wp)

lvls = [0.5, 1.5, 2.5, 3.5]
x_grid = range(0, 5, length = 50)
y_grid = range(0, 5, length = 50)

contour(x_grid, y_grid, optimal_policy', fill=true, levels=lvls,color = :
↳Blues,
        fillalpha=1, cbar = false)
contour!(xlabel="theta", ylabel="epsilon")
annotate!([(1.8,2.5, text("new life", 14, :white, :center))])
annotate!([(4.5,2.5, text("new job", 14, :center))])
annotate!([(4.0,4.5, text("stay put", 14, :center))])
```

Out[9]:



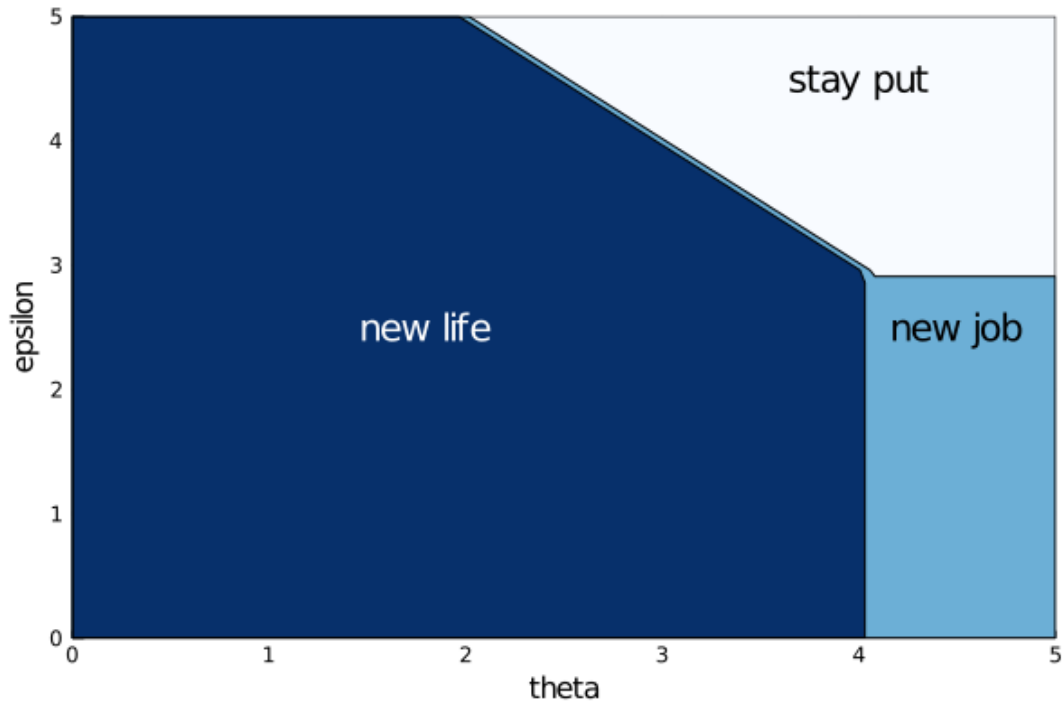
Now, we need only swap out for the new parameters

```
In [10]: wp = CareerWorkerProblem(G_a=100.0, G_b=100.0); # use new params
v, optimal_policy = solve_wp(wp)

lvls = [0.5, 1.5, 2.5, 3.5]
x_grid = range(0, 5, length = 50)
y_grid = range(0, 5, length = 50)

contour(x_grid, y_grid, optimal_policy', fill=true, levels=lvls, color = :
↳Blues,
        fillalpha=1, cbar = false)
contour!(xlabel="theta", ylabel="epsilon")
annotate!([(1.8,2.5, text("new life", 14, :white, :center))])
annotate!([(4.5,2.5, text("new job", 14, :center))])
annotate!([(4.0,4.5, text("stay put", 14, :center))])
```

Out[10]:



You will see that the region for which the worker will stay put has grown because the distribution for ϵ has become more concentrated around the mean, making high-paying jobs less realistic.

References

- [1] L Ljungqvist and T J Sargent. *Recursive Macroeconomic Theory*. MIT Press, 4 edition, 2018.
- [2] Derek Neal. The Complexity of Job Mobility among Young Men. *Journal of Labor Economics*, 17(2):237–261, 1999.