Schelling Model

1- انیمیشن شبیه سازی schelling_model_animation.py

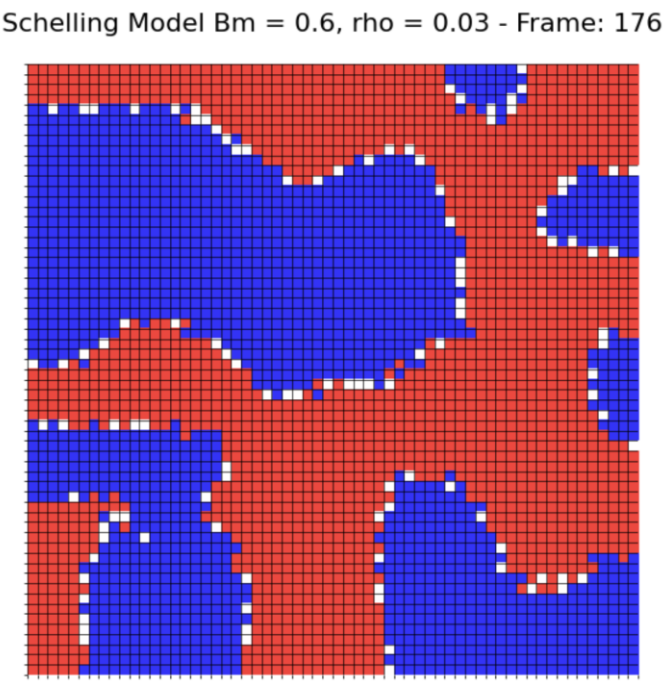
الگوریتم: ابتدا به صورت رندوم لتیس $N \times N$ را با اعداد 1 ,0 ,1- پر می کنیم. (1 با قرمز 0 با سفید و 1- با آبی نمایش داده می شوند.)
0 نمایانگر خانه های خالی است که با چگالی ρ و 1 و 1- مامور های مختلف هستند که هر کدام با چگالی $\frac{1-\rho}{2}$ پخش می شوند.
برای انتقال مامورها به خانه های خالی دو انتخاب داریم:

1) انتقال به نزدیک ترین خانه خالی
2) انتقال به خانه خالی رندوم

به همین دلیل در تابع *move()* یک متغیر *p_rand_move* تعریف می کنیم که احتمال انتخاب روش دوم نسبت به روش اول است. و به نحوی هر دو روش در الگوریتم به کار رفته است و می توانیم با یک احتمال بین آن ها انتخاب کنیم.

با مقایسه دو روش (*p_rand_move = 0 or 1*) به این نتیجه می رسیم که برخی مواقع انتقال به نزدیک ترین خانه خالی باعث می شود سیستم به تعادل نرسد و یک مامور به دام بیافتد. همچنین هزینه محاسباتی انتقال رندوم بسیار کمتر است و در اکثر اوقات در ایتریشن های کمتری به تعادل می رسد. پس انتقال به خانه رندوم نسبت به انتقال به نزدیک ترین خانه برتری دارد و از این پس با ***P_rand_move = 1*** ادامه می دهیم.

نتایج: گیف های شبیه سازی در فایل زیپ قرار دارد.



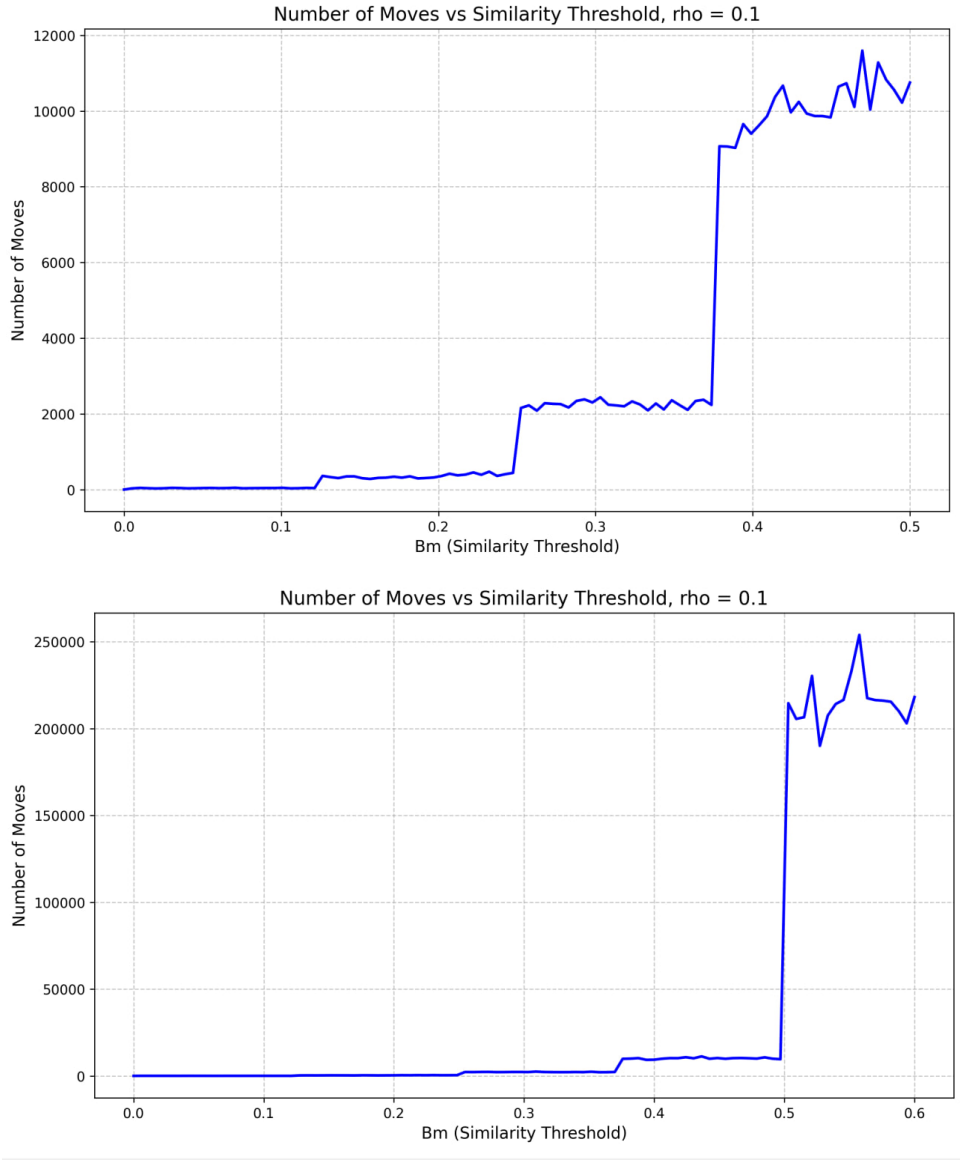
جداشدگی کاملا مشهود است. در برخی از سیستم ها با ρ کم و B_m بالا، سیستم به تعادل نمی رسد.

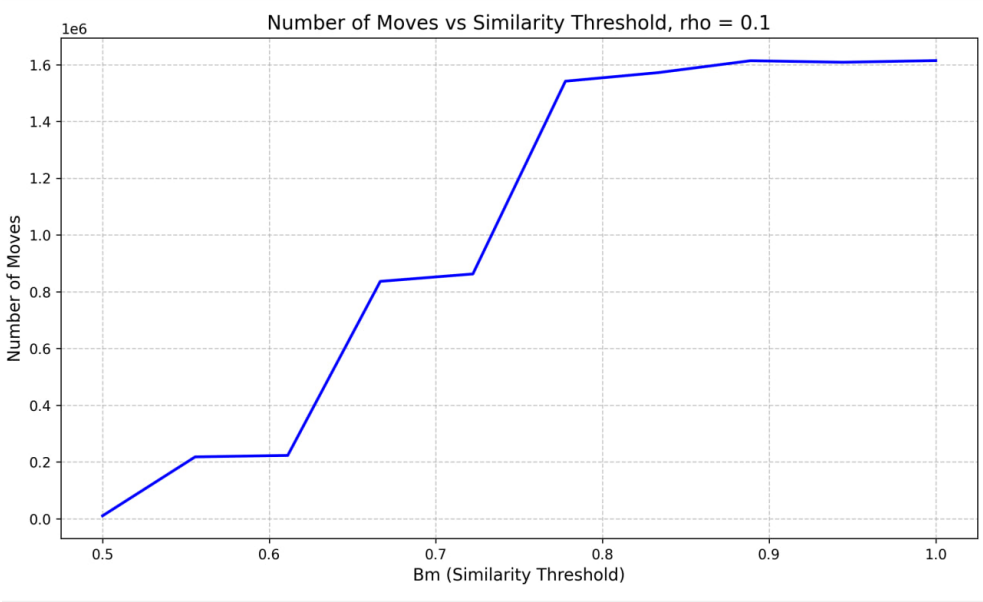
2- میانگین دفعات جابجایی برحسب نسبت کمینه تحمل num_moves.py

الگوریتم: برای رسم نمودار تعداد دفعات جابجایی برحسب ρ , B_m باید توجه داشته باشیم که برخی از سیستم ها به تعادل نمی رسند. پس باید عوامل دیگری برای متوقف کردن شبیه سازی هر سیستم در نظر بگیریم.

1) اگر تعداد جابجایی ها در این قدم صفر بود، شبیه سازی متوقف شود.
2) اگر تعداد قدم ها به max_iter رسید، شبیه سازی متوقف شود. (I)

نتایج:





مشاهده می کنیم که تعداد جابجایی ها به صورت پله پله در حال افزایش است. با بررسی بیشتر می بینیم، این پله ها در نقاط زیر قرار دارند.

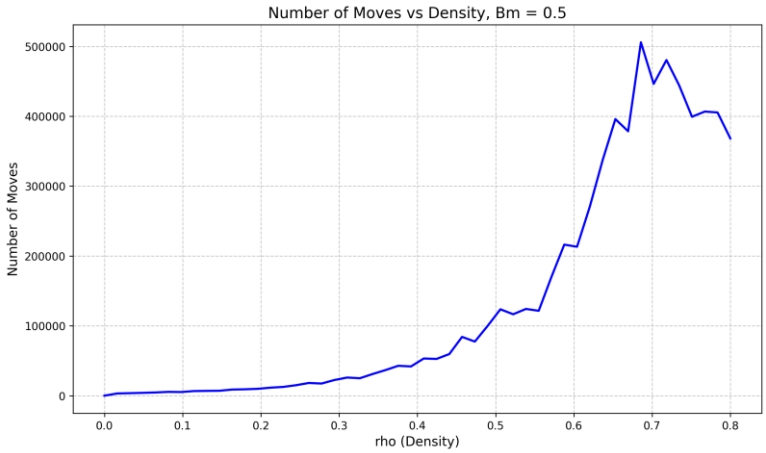
$$B_m: \quad \frac{n}{8} \quad , \quad n = 1, 2, 3, 4, 5, 6, 7, 8$$

دلیل این اتفاق این است که اکثر مامورها 8 همسایه دارند (به جز مامور های مرزی) . همچنین B_m نسبت تعداد همسایگان قابل تحمل به کل همسایگان است که قاعدتا برای اکثر مامورها به صورت n/8 می باشد.

به طور کلی مشاهده می شود هر پله بلند تر از پله قبلی است به جز پله های بعد از $B_m = \frac{5}{8} = 0.625$. دلیل این اتفاق این است که بعد از این مقدار اکثر سیستم ها به تعادل نمی رسند و با عوامل (I) متوقف می شوند.

3- میانگین دفعات جابجایی برحسب چگالی خانه های خالی num_moves.py

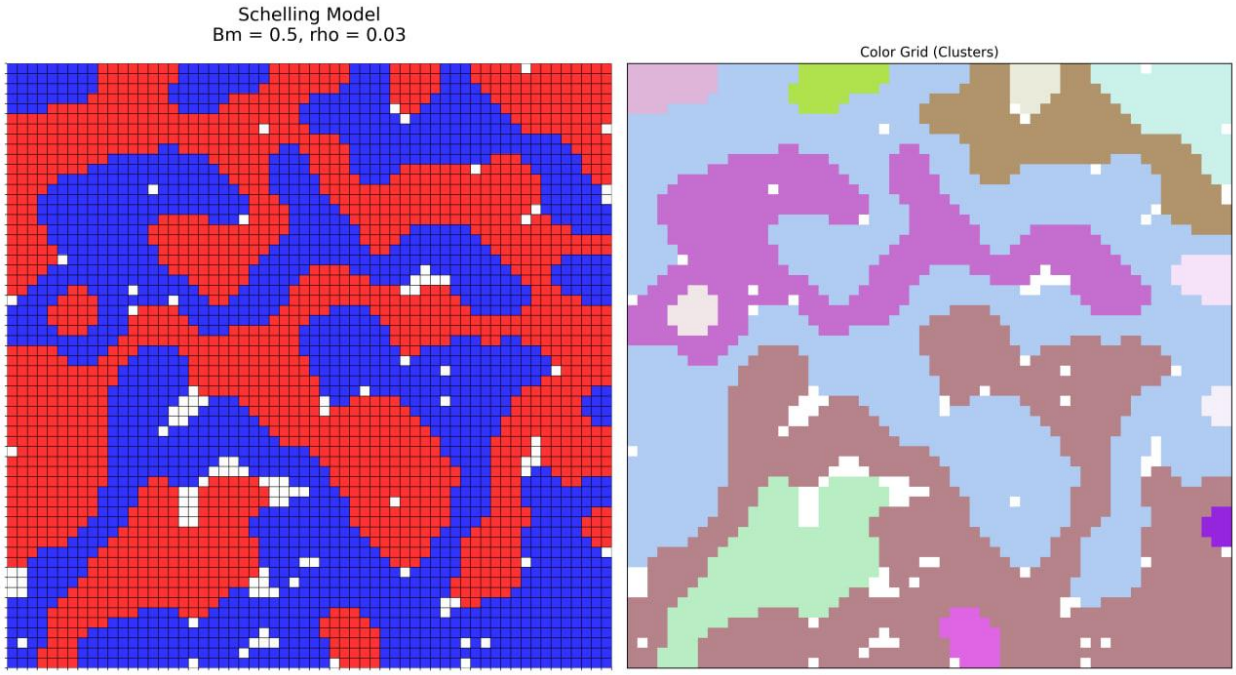
همانند قسمت قبل داریم:



مشاهده می کنیم که ابتدار در $\rho = 0$ به دلیل عدم وجود خانه خالی، هیچ جابجایی نداریم ولی به مرور جابجایی به صورت نمایی افزایش پیدا می کند.

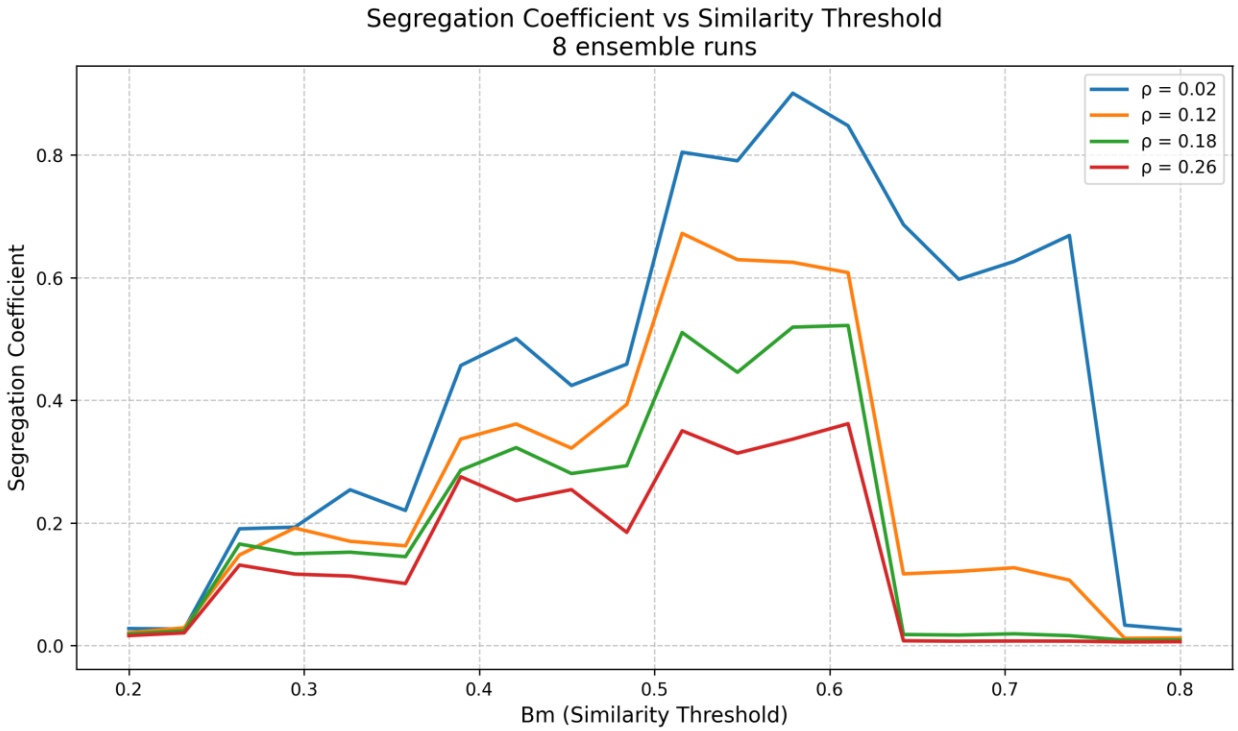
4- ضریب جداسازی segregation_coeficient.py

الگوریتم: برای تشخیص خوشه ها از الگوریتم رنگ آمیزی به کار رفته در تراوش، استفاده می کنیم. با این تفاوت که به جای 4 همسایه 8 همسایه را در نظر می گیریم. برای استفاده از این الگوریتم یک لتیس دیگر برای ترک کردن رنگ ها تعریف می کنیم. (color_grid)
باید توجه داشت که پردازشی روی خانه های خالی انجام ندهیم. و برای تشخیص خوشه از تعریف همسایگی Moore (هشت همسایه) استفاده نکنیم و از تعریف چهار همسایه بالا، پایین، چپ و راست استفاده کنیم.
هنگامی که سیستم به تعادل رسید یا شبیه سازی متوقف شد، این الگوریتم را صدا می کنیم.

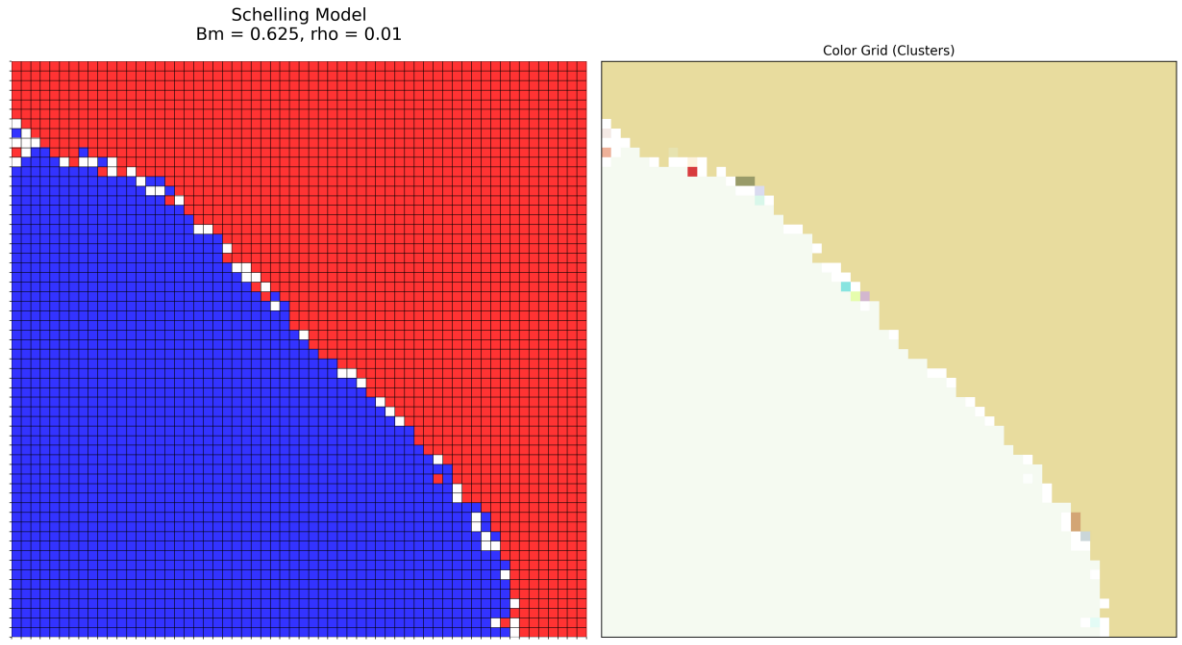


پس از تشخیص خوشه ها، برای پیدا کردن اندازه خوشه ها کافی است دوباره لتیس را اسکن کنیم تا تعداد مامورها با رنگ یکسان را به عنوان اندازه یک خوشه ذخیره کنیم. سپس می توانیم ضریب جداسازی را محاسبه کنیم.

نتایج:



طبق مقاله، هرچه ضریب جداسازی به 1 نزدیک تر باشد به این معنی است که سیستم فقط دو خوشه جدا دارد.
باز هم رفتار پله پله در n/8 مشاهده می شود. طبق نمودار در دو ناحیه $0.5 \leq B_m \leq 0.625$ جدا شدگی ماکسیمم مشاهده می شود و در دو سر نمودار جداشدگی به صفر میل می کند. همچنین به این نتیجه می رسیم هر چه ρ کمتر باشد، احتمال جداشدگی کامل بیشتر است.
برای بررسی بیشتر صحت این موضوع برای $B_m = 0.625$ شبیه سازی را تکرار می کنیم.



همانطور که انتظار داشتیم جداسازی کامل اتفاق افتاده است. گیف جداسازی کامل با نام full_seperation.gif در فایل زیپ موجود است.