«به نام خدا»

تکلیف 1 هوش – مرضیه علیدادی – 9631983

---

1)

الف- برازندگی افراد:

$f(x_1) = (6+5) – (4+1) + (3+5) – (3+2) = 9$
$f(x_2) = (8+7) – (1+2) + (6+6) – (0+1) = 23$
$f(x_3) = (2+3) – (9+2) + (1+2) – (8+5) = -16$
$f(x_4) = (4+1) – (8+5) + (2+0) – (9+4) = -19$

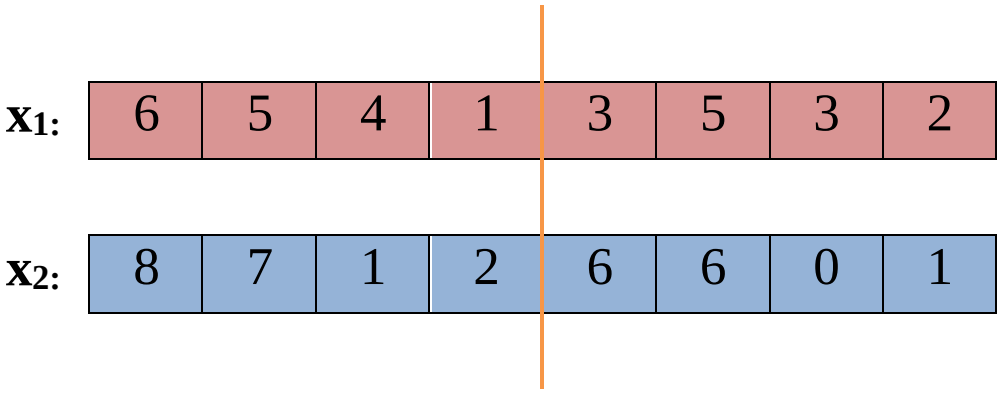
به ترتیب از بیشترین به کمترین برازندگی از چپ به راست:

$f(x_2) > f(x_1) > f(x_3) > f(x_4) \rightarrow x_2, x_1, x_3, x_4$

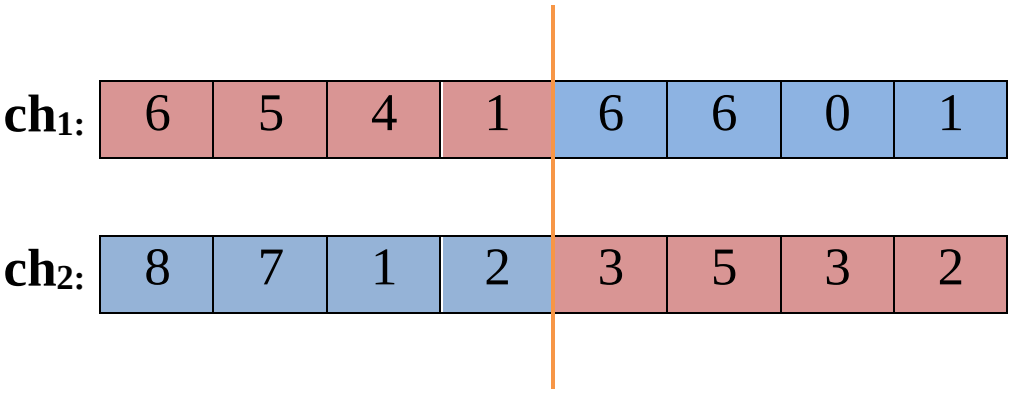
ب- عملیات های crossover :

- **single point crossover :**

Parents: x_1, x_2:

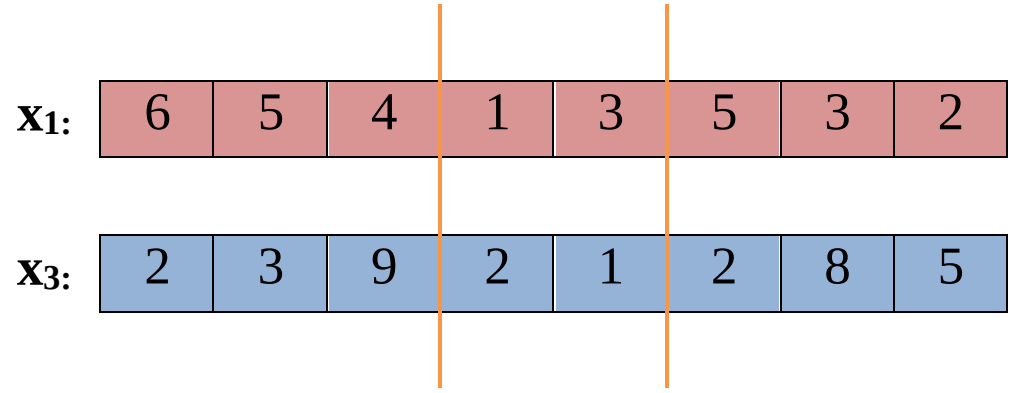


Children :

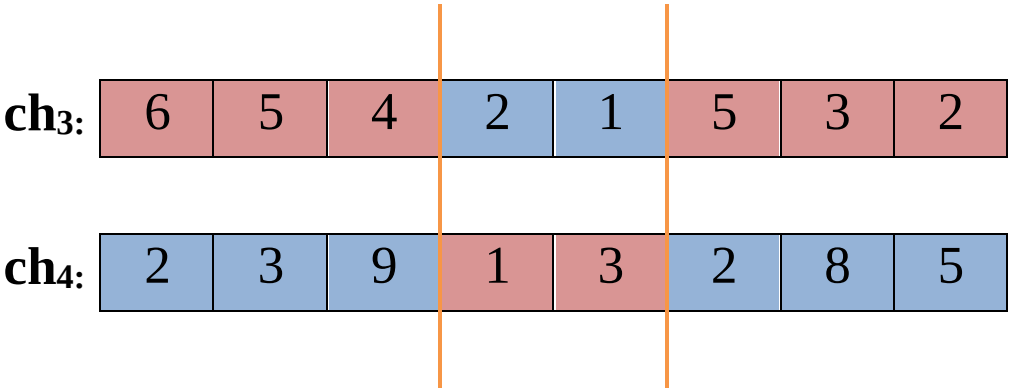


• **two point crossover :**

Parents: x_1, x_3:

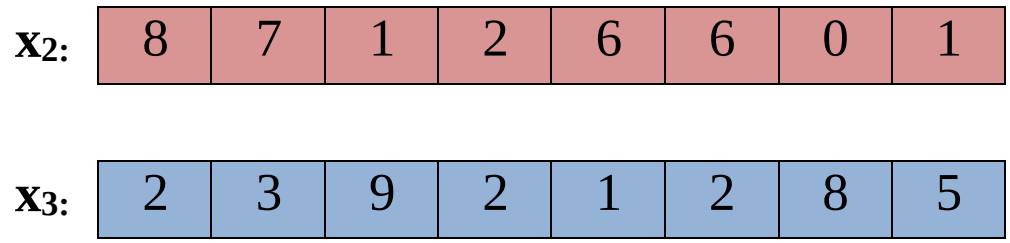


Children :

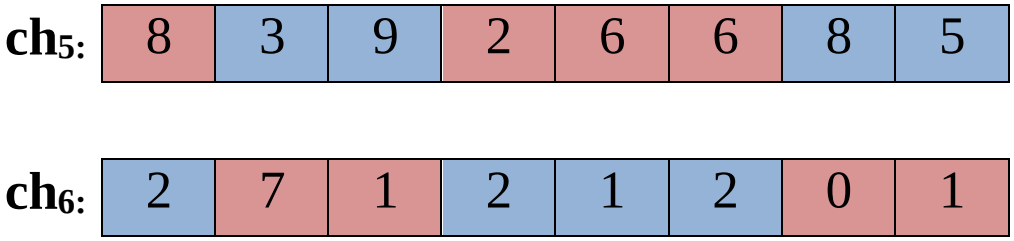


- **uniform crossover** :

Parents: x_2, x_3:



Children :



پ- برازندگی فرزندان حاصل:

$f(ch_1) = (6+5) – (4+1) + (6+6) – (0+1) = 17$
$f(ch_2) = (8+7) – (1+2) + (3+5) – (3+2) = 15$
$f(ch_3) = (6+5) – (4+2) + (1+5) – (3+2) = 6$
$f(ch_4) = (2+3) – (9+1) + (3+2) – (8+5) = -13$
$f(ch_5) = (8+3) – (9+2) + (6+6) – (8+5) = -1$
$f(ch_6) = (2+7) – (1+2) + (1+2) – (0+1) = 8$

دو فرزند ch_4 و ch_5 که برازندگی کمتری دارند را حذف می کنیم. باقی فرزندان یعنی ch_1 و ch_2 و ch_3 و ch_6 به عنوان نسل جدید در نظر گرفته می شوند. (با این فرض که نسل جدید فقط از فرزندان تشکیل می شود. و والدین را به کل کنار می گذاریم.)

برازندگی کل جمعیت اولیه:

$f_{total} = 9 + 23 + (-16) + (-19) = -3$

برازندگی کل جمعیت جدید:

$f_{total} = 17 + 15 + 6 + 8 = 46$

پس مقدار برازندگی کل **بهبود** یافته است.

ت- جواب بهینه:

با توجه به اینکه مقدار ژن های a و b و e و f تاثیر مثبت روی تابع برازندگی کروموزوم دارد، مقدار این ژن ها را در بیشترین حالت ممکن یعنی 9 می گیریم.

و با توجه به اینکه مقدار ژن های c و d و g و h تاثیر منفی روی تابع برازندگی کروموزوم دارد، مقدار این ژن ها را در کمترین حالت ممکن یعنی 0 می گیریم.

لذا کروموزوم بهینه عبارتست از:

$x_{optimum} = 99009900$

مقدار بیشترین برازندگی:

$f(x_{optimum}) = (9+9) - (0+0) + (9+9) - (0+0) = \mathbf{36}$

ث- عملگر جهش:

یک جهش مناسبی که می توان برای این مسئله در نظر گرفت، uniform mutation است. در این جهش به این صورت عمل می شود: مثلا از سمت چپ روی کروموزوم مورد نظر حرکت می کنیم. هر کدام از 8 تا ژن سازنده ی یک کروموزوم، با احتمال خیلی اندکی که برای همه یکسان است، ممکن است انتخاب شود، برای این که روی آن جهش رخ دهد. هر کدام از ژن ها که به صورت تصادفی انتخاب شد، مقدار موجود در آن موضع پاک می شود و یک عدد تصادفی در بازه ی مورد نظر یعنی [9 , 0] انتخاب می شود و در آن موضع قرار می گیرد؛ که احتمال انتخاب هر کدام از این 10 عدد صحیح موجود در بازه ی [9 , 0] با هم برابر است.

ج- خیر؛

با توجه به اینکه در عملیات single point crossover فقط از مقادیر موجود در والدین و ترکیب آن ها برای ساختن فرزندان استفاده می شود، و با توجه به جواب بهینه ی سراسری که در بخش (ت) بیان شد؛ امکان دست یابی به جواب بهینه با این نوع عملیات crossover و بدون هیچ عملیات جهشی وجود ندارد. چرا که برای دست یابی به جواب بهینه، باید در ژن های a و b و e و f مقدار 9 قرار بگیرد، و در ژن های c و d و g و h هم مقدار 0 قرار بگیرد؛ که این اتفاق هیچ گاه با ترکیب والدین در هیچ کدام از نسل ها رخ نخواهد داد.

(2

الف- steepest ascent/descent Hill Climbing ؛

در این الگوریتم، با احتمال $e^{\frac{\Delta E}{T}}$ به همسایه های بدتر خواهیم رفت. وقتی T = 0 باشد، توان این عبارت با توجه به اینکه ΔE همواره منفی است، نزدیک ∞- می شود، لذا این احتمال خیلی کم می شود. یعنی احتمال اینکه به همسایه های بدتر برویم، از بین می رود و رفتارِ greedy در الگوریتم افزایش پیدا می کند و randomness در الگوریتم از بین می رود. لذا در هر مرحله، علاقه مندیم که به بهترین همسایه برویم؛ پس مانند این است که الگوریتم SA به الگوریتم Hill Climbing عادی تبدیل شده است.

ب- Random Walk/Search ؛

در این الگوریتم، با احتمال $e^{\frac{\Delta E}{T}}$ به همسایه های بدتر خواهیم رفت. وقتی T = +∞ باشد، توان این عبارت به 0 میل می کند، لذا این احتمال خیلی زیاد و نزدیک به 1 می شود. یعنی احتمال اینکه به همسایه های بدتر برویم، خیلی زیاد می شود و randomness در الگوریتم افزایش پیدا می کند و رفتارِ greedy در الگوریتم از بین می رود. لذا در هر مرحله، از همسایه ها به صورت تصادفی و random انتخاب خواهیم کرد؛ پس مانند این است که الگوریتم SA به الگوریتم purely random search تبدیل شده است.

پ- steepest ascent/descent Hill Climbing ؛

در این الگوریتم، در ابتدا با k تا فرد به عنوان جمعیت اولیه مواجه هستیم. در هر مرحله، همسایه های این افراد را می سازیم؛ سپس از بین آن همسایه ها، k تا بهترین افراد را به عنوان جمعیت بعدی در نظر میگیریم. اگر k = 0 باشد، در ابتدا و در همه ی مراحل با 1 نفر به عنوان جواب مواجه هستیم، و در هر

مرحله، همسایه های همان 1 نفر را می سازیم؛ سپس از بین آن ها بهترین را انتخاب می کنیم. لذا مانند این است که الگوریتم ما به الگوریتم Hill Climbing عادی تبدیل شده است.

ت- Breadth-First ؛

در این الگوریتم مبتنی بر جمعیت، اگر جمعیت را برابر با ∞+ کنیم، در واقع کل جمعیت را بررسی کرده ایم و کل فضای حالت را پیمایش کرده ایم. و مانند این می شود که ما یک complete search داریم و دیگر الگوریتم ما local search نیست. در این حالت ما در هر مرحله همه جایگاه ها و همه ی جمعیت را در نظر گرفته‌ایم و به صورت یکجا کل جمعیت را هر مرحله خواهیم داشت؛ چرا که تعداد جمعیت به صورت بی نهایت در نظر گرفته شده است. که البته این کار خیلی منطقی نیست. ما اگر این را به این شکل در نظر بگیریم که از یک وضعیت شروع می‌کنیم در این جمعیت، و از آن شروع به بررسی می‌کنیم، می‌توانیم بگوییم که شبیه به breadth-first می شود؛ با این تفاوت که ما در الگوریتم breadth-first به صورت لایه لایه در node ها پیمایش انجام می دهیم، اما اینجا هر لایه شامل کل جمعیت به صورت یک جا می شود.

ث- با فرض در نظر گرفتن نسل حاضر در انتخاب نسل بعد: first-choice Hill Climbing و با فرض در نظر نگرفتن آن: Random Walk ؛

در GA، زمانی که جمعیت فقط شامل یک نفر باشد، برخی از مکانیزم ها و عملیاتی که در این الگوریتم انجام می دادیم، بی معنی می شود. از بین مولفه های این الگوریتم representation هنوز به همان شکلی که داشتیم تعریف می شود. تابع fitness هم به همان شکلی که هست باقی می ماند، که البته به طور کلی شبیه همان تابع هدفی است که در الگوریتم های جستجوی محلی مبتنی بر فرد واحد داشتیم. مولفه ی population و اندازه ی جمعیت، دیگر مطرح نیست؛ چرا که اندازه ی جمعیت دیگر برابر با یک است و فقط یک فرد وجود دارد. مکانیزم parent selection هم از بین می رود، چرا که فقط یک نفر را داریم و فرد دیگری وجود ندارد که بتوان آن ها را به عنوان دو والد انتخاب کرد و با هم ترکیب کرد. لذا عملیات recombination بین دو والد هم از بین می رود. اما عملیات mutation روی فرد انجام می شود و در نهایت هم فقط یک فرد دیگر به وجود می آید. و (با فرض اینکه والد ها در تشکیل نسل بعد در نظر گرفته می شوند) مکانیزم انتخاب نسل به این شکل صورت می گیرد که با توجه به تابع fitness، بین فرد نسل حاضر و فرد حاصل از mutation، یکی که fitness بهتری دارد، به عنوان نسل بعد در نظر گرفته می شود.

پس اینگونه شد که ما یک فرد داریم که روی آن عملیات mutation را با هدف max یا min شدن تابع هدف انجام می دهیم. به صورت تصادفی یک mutation در آن رخ می دهد، مثلا اگر نمایش آن

باینری باشد، یک بیت آن flip می شود. لذا فردی که با استفاده از mutation به وجود آمده است، مانند این است که همسایه ی فردی است که تشکیل دهنده ی نسل حاضر است. پس الگوریتم به این صورت شد که گویی ما در هر مرحله، یک همسایه به صورت تصادفی برای فرد حاضر به وجود می آوریم و اگر بهتر بود به آن می رویم(آن را به عنوان نسل بعدی انتخاب می کنیم)؛ و اگر بهتر نبود، دوباره همین فرد نسل حاضر را انتخاب میکنیم و دوباره عملیات ها را روی آن انجام می دهیم و mutation را دوباره روی آن اعمال می کنیم تا همسایه ی بهتر ایجاد شود و به آن برویم. پس مانند این است که GA به first-choice Hill Climbing تبدیل شده است.

حالا اگر الگوریتم را به این صورت در نظر بگیریم که برای تعیین افراد تشکیل دهنده ی نسل بعد، فرد تشکیل دهنده ی نسل حاضر را کلا کنار بگذاریم، و حتما فرد جدیدی که با mutation ایجاد کرده ایم، نسل بعدی را تشکیل بدهد؛ در این صورت الگوریتم GA به random walk تبدیل می شود.

---

3) (از بین node های با اسم G، آن که هزینه اش 1 بود را G_1 و دیگری را G_2 در نظر گرفتم.)

الف- Hill Climbing

مسیر با کمترین هزینه:

$A \rightarrow B \rightarrow G_2 \rightarrow G_1 \rightarrow F$

مسیر با بیشترین پاداش:

$A \rightarrow B \rightarrow C \rightarrow D \rightarrow G_1 \rightarrow E \rightarrow F$

یا

$A \rightarrow B \rightarrow C \rightarrow D \rightarrow G_2 \rightarrow G_1 \rightarrow E \rightarrow F$

(در D به صورت تصادفی بین G_1 و G_2 یکی انتخاب می شود و یکی از دو مسیر بالا در نظر گرفته می شود.)

ب- Tabu search

(چون در مسیرش هیچ تکراری وجود نداشت تا از tabu list استفاده کند، نتیجه دقیقا مانند زمانی شد که از hill climbing برای یافتن مسیر استفاده کردیم.)

مسیر با کمترین هزینه:

$A \rightarrow B \rightarrow G_2 \rightarrow G_1 \rightarrow F$

مسیر با بیشترین پاداش:

$A \rightarrow B \rightarrow C \rightarrow D \rightarrow G_1 \rightarrow E \rightarrow F$

یا

$A \rightarrow B \rightarrow C \rightarrow D \rightarrow G_2 \rightarrow G_1 \rightarrow E \rightarrow F$

(در D به صورت تصادفی بین G1 و G2 یکی انتخاب می شود و یکی از دو مسیر بالا در نظر گرفته می شود.)

پ- برای یافتن مسیر با کمترین هزینه، هر دو الگوریتم در مینیمم سراسری متوقف شده اند. ولی برای یافتن مسیر با بیشترین پاداش در ماکسیمم محلی متوقف شده اند.

ت- خیر؛ چرا که این دو مسئله حتی برای یافتن بیشترین پاداش هم به مشکل خوردند و در بهینه ی محلی متوقف شدند. لذا با در نظر گرفتن این دوجنبه کنارِ هم نیز، نمی توانند بهینه ی سراسری را حاصل شوند.

مثلا اگر به جای hill climbing عادی از stochastic hill climbing استفاده شود، با توجه به اینکه در هر مرحله صرفا بهترین همسایه انتخاب نمی شود، رویکرد greedy مسئله کاهش پیدا می کند. لذا احتمال همگرا شدن به بهینه ی سراسری با هر دو این ویژگی ها وجود دارد. که برای fitness هر حرکت، میتوان این تابع را در نظر گرفت:

$$f = \sum \text{پاداش} - \sum \text{هزینه}$$

---

4)

الف- اعضای فضای جستجو را نقاط با مختصات صحیح روی محور دو بعدی داده شده در نظر می گیریم، و آن ها را با زوج مرتب (x , y) نمایش می دهیم. که در فضای حالت، تعدای از افراد وجود دارد که نامعتبر هستند؛ که این افراد همان نقاطی هستند که چندضلعی ها درآن ها واقع شده اند. این افراد نامعتبر را در ابتدا مشخص می کنیم، و از در نظر گرفتن این افراد به عنوان فرد مورد نظر به عنوان جواب مسئله، خودداری می کنیم.

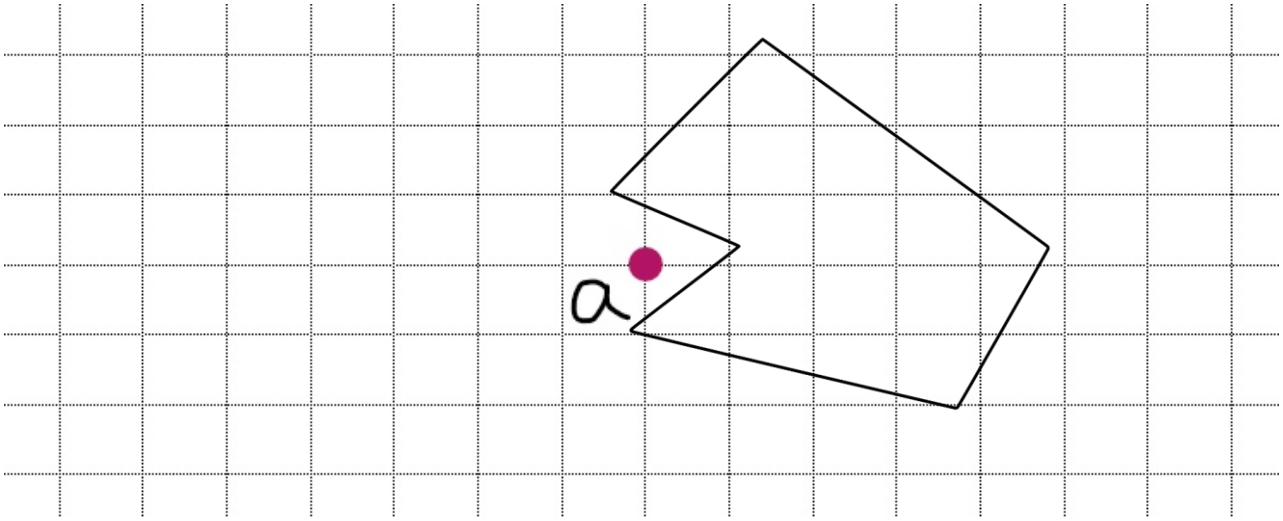
همسایگی به این ترتیب تعریف می شود: برای هر عضو از فضای جستجو، 3 همسایه وجود دارد. که به

ازای فردی با نمایش (x , y) ، همسایه های آن عبارتند از $(x+1 , y)$ ، $(x , y+1)$ و $(x+1 , y+1)$.
یعنی در بردار دو بعدی مختصات، موقعیت آن ها به این شکل است که، یک همسایه سمت راست فرد، دیگری بالای آن و سومی بالا سمت راست آن قرار دارد.
fitness افراد به این ترتیب محاسبه می شود: با توجه به اینکه نقطه ی هدف برای ما مشخص است و مختصات آن را داریم، فرضا مختصات نقطه ی هدف برابر است با $G = (x_G , y_G)$. fitness برای فردی با مختصات (x , y)، به این شکل در نظر گرفته می شود که چقدر به مختصات نقطه ی هدف نزدیک است. یعنی تابع زیر باید minimize شود:

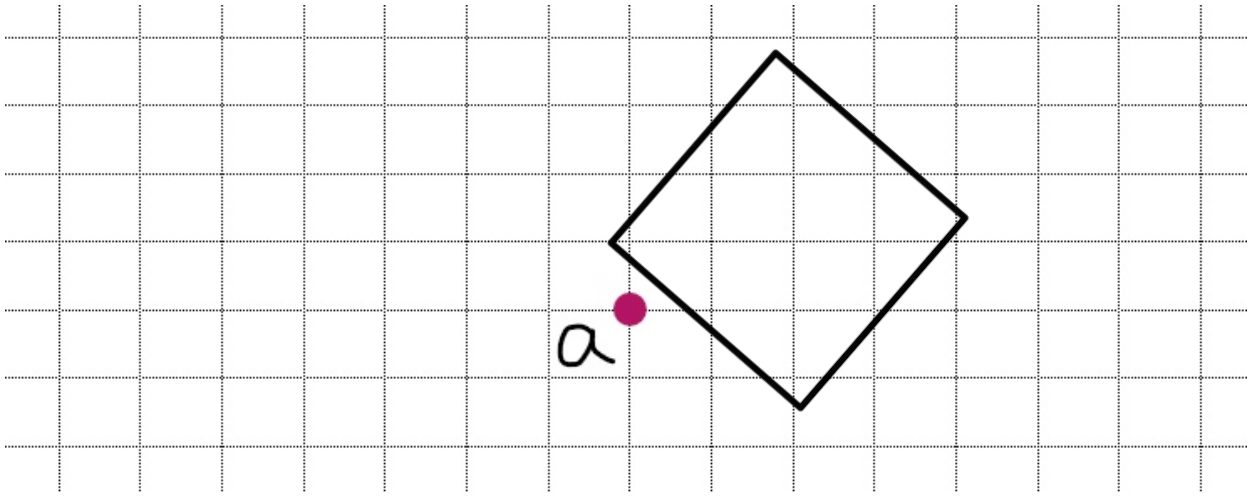
$$f = (x_G - x) + (y_G - y)$$

روند کار الگوریتم به این صورت است: در هر مرحله برای فرد مورد نظر، همسایه های نظیرش را می سازیم. سپس بررسی می کنیم که آیا این همسایه ها معتبر هستند (با مقایسه نمایش آن ها با نقاط نامعتبری که به عنوان محل قرار گرفتن چندضلعی ها داریم) یا خیر؟! سپس از بین همسایه هایی که معتبر تشخیص داده شده اند (که تعداد آن ها کمتر یا مساوی 3 است) ، همسایه ای که fitness بهتری دارد، انتخاب می شود و به عنوان فرد جدید در نظر گرفته می شود. با توجه با اینکه این مسئله minimization است، همسایه ای که مقدار f اش کمتر شد، برنده است؛ و اگر دو یا چند همسایه مقدار f یکسانی داشتند، از بین آن ها یکی به تصادف انتخاب می شود.

ب- برای مثال اگر برای شکل غیر محدب زیر، فرد در موقعیت a قرار گیرد، الگوریتم در همین نقطه که بهینه ی محلی است، متوقف می شود. چرا که با توجه به تعریف همسایگی، برای فرد همسایه ی معتبری وجود ندارد.



پ- بله؛ مثلا وقتی فرد در موقعیت زیر قرار گیرد، همسایه هایش نامعتبر می شوند و الگوریتم در بهینه ی محلی متوقف می شود:



ت- در صورت استفاده از SA ، با توجه به اینکه الگوریتم در ابتدای کار رویکرد greedy اش کم است و بیشتر رویکرد تصادفی دارد، همسایه های با fitness بد تر از خود فرد هم شانس انتخاب شدن دارند. لذا می توان همسایگی یک فرد را افراد موجود در شعاع یک متری آن فرد در صفحه ی مختصات در نظر گرفت. به عبارتی برای هر فرد با نمایش (x , y) ، 8 همسایه با نمایش های $(x+1 , y)$ ، $(x , y+1)$ ، $(x+1 , y+1)$ ، $(x-1 , y+1)$ ، $(x , y-1)$ ، $(x-1 , y)$ ، $(x+1 , y-1)$ و $(x -1, y-1)$ وجود خواهد داشت. لذا با توجه به این همسایگی بیان شده، و با توجه به خاصیت الگوریتم SA ، که در ابتدای کار رویکرد تصادفی دارد و رفته رفته greedy می شود، احتمال توقف در بهینه ی محلی کاهش پیدا می کند. چرا که در صورت قرار گرفتن فرد در موقعیتی مانند آنچه در مثال قبل بیان شد، همسایه های معتبر دیگری برای فرد وجود دارد، که از خود فرد fitness بدتری دارند (با توجه با تابع fitness بیان شده در بخش <الف>) ، ولی موجب فرار از بهینه ی محلی می شوند.

اما خب اگر گیر کردن در بهینه ی محلی در آخر کار الگوریتم که رویکردش greedy می شود، اتفاق بیفتد، در آن متوقف می شود.

(اگر برای الگوریتم hill climbing این همسایگی را در نظر می گرفتیم، در عمل فقط همان 3 همسایه ی بیان شده در بخش <الف> شانس انتخاب شدن را داشتند و بقیه ی همسایه ها عملا شانس برنده شدنشان برابر با 0 می شد. لذا در آن بخش، از همان ابتدا، فقط همان 3 همسایه را به عنوان همسایه ی یک فرد معرفی کردم.)

5)

الف- در حل این مسئله برای نشان دادن جواب، از نمایش رشته ی باینری استفاده می کنم و هر یک از اعضای فضای جستجو را به صورت یک رشته باینری نمایش می دهم. به این صورت که به هر کدام از یال های گراف، یه بیتِ نظیر در رشته باینری اختصاص می دهم. لذا این رشته ی باینری طولش به اندازه ی تعداد یال های گراف است. ۰ یا ۱ بودن بیت نظیر هر کدام از یال ها به معنای این است که دو سر انتهایی آن یال در یک دسته قرار گرفته اند یا خیر. به این ترتیب که بیت هایی که مقدارشان ۱ است، دو node انتهایی یال نظیرشان، در یک دسته قرار گرفته اند؛ و آن هایی که مقدارشان ۰ است، دو node انتهایی یال نظیرشان ، در دسته های متفاوتی قرار گرفته اند.

ب- همانطور که در قسمت قبل بیان شد، طول رشته ی باینری که برای نمایش هر یک از اعضا به کار می رود، برابر با تعداد یال ها یعنی m است. بنابراین 2^m رشته ی باینری متفاوت خواهیم داشت. لذا تعداد اعضای فضای جستجو 2^m تا است.

پ- برای تعریف همسایگی این مسئله از همسایگی معروف 1-flip ، که معمولاً برای تعریف همسایگی مسائلی که با رشته باینری نمایش داده می شوند استفاده می شود، استفاده می کنم. بدین شکل که هر بار با flip کردن هر یک از بیت های رشته ی باینری، یک همسایه تولید می کنم. بدین معنی که مثلاً اگر دو راس انتهایی یک یال در یک دسته قرار گرفته اند، حالا با flip کردن بیت نظیرش در رشته ی باینری، هر راس انتهایی آن در دسته های متفاوتی قرار می گیرد، و بالعکس. لذا تعداد همسایه های هر یک از اعضا، برابر است با طول رشته ی باینری.

ت- با توجه به تعریف همسایگی که در بخش قبل بیان شد، تعداد همسایه های همه ی اعضا، یکسان، و برابر با طول رشته ی باینری است؛ که در واقع همان تعداد یال های گراف یا m است.

ث- بله؛ یک روش برای حل مسئله خوشه بندی گراف ها، دسته بندی با اندازه گیری quality و کیفیت است؛ که از maximum spanning tree مربوط به گراف های وزن دار استفاده می کند. نحوه ی گروه بندی balance شده ی این الگوریتم، با هدف max یا min کردن است. این راه حل که از dynamic programming استفاده می کند، در polynomial time مسئله را حل می کند. و time complexity آن برابر با $O(k^2n^3)$ است.
در این مقاله تشریح شده است:
[A Polynomial Algorithm for Balanced Clustering via Graph Partitioning]