Haplotype Inference by parsimony

- **مقدمه :**

ابتدا با مرور چند مفهوم، المان‌های مسأله را معرفی می‌کنیم. سپس به شرح دقیق مسأله می‌پردازیم.
هر انسان حدود ۲۵ ,۰۰۰ تا ۳۰ ,۰۰۰ ژن دارد. معمولاً، هر کدام از این ژن‌ها چندین آلل در میان افراد مختلف دارند. همین موجب متفاوت بودن ژنوتیپ آن‌ها، و در نهایت گوناگونی می‌شود، چرا که احتمال این که دو فرد برای همهٔ ژن‌های خود یک آلل را به ارث برده باشند تقریباً صفر است! اطلاعات هاپلوتیپ نقش مهمی در مطالعات ژنتیکی، پیدا کردن ژن‌های بیماری‌های سخت، طراحی دارو و ... دارد. اما ژنوتیپ یک موجود زنده ممکن است دقیقاً هاپلوتیپ آن را بیان نکند و تشخیص تجربی هاپلوتیپ‌ها هم پرهزینه و وقت‌گیر است. بنابراین نیازمند تکنیک‌هایی برای استنتاج اطلاعات هاپلوتیپ‌ها از روی ژنوتیپ هستیم.

هاپلوتیپ *h را با یک آرایه‌ی mتایی (شامل m آلل) نمایش می‌دهیم که هر آلل می‌تواند مقدار ۰ یا ۱ داشته باشد.

همچنین **ژنوتیپ g** را با یک آرایه‌ی mتایی نشان می‌دهیم که هر عنصر آن می‌تواند مقادیر ۰ یا ۱ (در حالت‌های غالب) یا ۲ (در حالت مغلوب) را داشته باشد.

عملگر همگشتی ⊕ را به صورت زیر تعریف می‌کنیم:

⊕:{0,1} -> {0,1,2}

⊕	0	1
0	0	2
1	2	1

پس اگر g_j یک ژنوتیپ در سطر jم ماتریس n*mی که مصداق n ژنوتیپ است، باشد، و نیز اگر $h^*_{j,1}$ و $h^*_{j,2}$ دو هاپلوتیپ متناظر با g_j باشند، داریم:

$$g = h_1^* \oplus h_2^* \iff g_j = h_{1,j}^* \oplus h_{2,j}^* \ (j = 1, \ldots, m).$$

مثال این معادله در متن مقاله (Example 1) موجود است.

Maximum Parsimony Principle (اصل خسّت حداکثری):

این اصل بیان می‌کند که یکی از راه‌حل‌های ممکن، مطلوب است اگر تعداد هاپلوتیپ‌های متمایز موجود در مجموعه‌ی $\{h^*_1, h^*_2, \dots, h^*_{2n}\}$ حداقل باشد.
بسیاری از تکنیک‌های استنتاج هاپلوتیپ بر اساس همین اصل است، که هم از لحاظ نتایج تجربی و هم از لحاظ استدلال‌های نظری توجیه‌شده است. هرچند، نشان داده شده است که مسأله‌ی استنتاج هاپلوتیپ بر اساس خسّت (parsimony) یک مسأله‌ی NP-hard است. پس کاربردپذیری تکنیک‌هایی که دقیقاً بر اساس parsimony است بر روی دیتا ست کوچک، محدود است.

- **مسأله:**

  در این پروژه قصد داریم راه‌حلی تا جای ممکن کم‌هزینه بر اساس parsimony ارائه کنیم تا بتواند ماتریس H شامل حداقل هاپلوتایپ‌های متمایز و کافی متناظر با ژنوتیپ‌های ماتریس داده شده (G) را پیدا کند.

- **ورودی:**

ماتریس $G_{n,m}$ شامل n ژنوتیپ به طول m که هر عنصر آن می‌تواند مقادیر ۰ یا ۱ (در حالت‌های غالب) یا ۲ (در حالت مغلوب) را داشته باشد.

- **خروجی:**

ماتریس $H_{p,m}$ شامل p هاپلوتیپ متمایز به طول m که هر عنصر آن می‌تواند مقدار ۰ یا ۱ داشته باشد، و p حداقل مقدار ممکن باشد.

Problem Formulation

Initial State:

۱. ماتریس G شامل n ژنوتیپ که هرکدام m عنصر دارند.
۲. تعداد عناصر مغلوب (مقدار ۲) موجود در هر یک از ژنوتیپ‌های داده شده
۳. تعداد عناصر غالب (مقادیر ۰ و ۱) موجود در هر یک از ژنوتیپ‌های داده شده

States:

هرstate شامل این اطلاعات است:
۱. هرکدام از هاپلوتایپ‌های موجود، در ساختن کدام ژنوتیپ‌های داده‌شده مفید هستند؟ (آرایه‌ای دو بعدی که در بعد اول آن اندیس‌های هاپلوتیپ‌های موجود قرار دارند و در بعد دوم آرایه‌ای از اندیس ژنوتیپ‌هایی که هر هاپلوتیپ مجاز به مشارکت در ساختن آن است)
۲. تعداد هاپلوتیپ‌هایی که در ساختن هیچ ژنوتیپی موثر نیستند (بی‌فایده‌اند).
۳. هر کدام از ژنوتیپ‌های داده‌شده، با چه زوج‌هایی از میان هاپلوتیپ‌های موجود قابل ساخته شدن هستند؟ (آرایه‌ای سه بعدی که در بعد اول آن اندیس‌های ژنوتیپ‌های داده‌شده، در بعد دوم آن هر یک از هاپلوتیپ‌های موجود که امکان ساختن آن ژنوتیپ را دارند و در بعد سوم، هاپلوتیپ مکمل آن قرار دارد)
۴. تعداد ژنوتیپ‌هایی که با هیچ زوج هاپلوتیپ موجودی قابلیت ساخته‌شدن ندارند.
۵. تعداد هاپلوتیپ‌های متمایز موجود

Actions:
هر action میتواند یکی از موارد زیر باشد:
۱. اضافه کردن یک هاپلوتیپ به مجموعه هاپلوتیپ‌های موجود که در تعداد ژنوتیپ‌هایی که با هیچ زوجی از هالوتیپ‌های موجود قابلیت ساخته شدن ندارند، کاهش ایجاد کند.
۲. حذف یک هاپلوتیپ از مجموعه هاپلوتیپ‌های موجود که در تعداد هاپلوتیپ‌هایی که در ساختن هیچ‌یک از ژنوتیپ‌های موجود تأثیر ندارند (بی‌فایده‌اند) کاهش ایجاد شود اما در تعداد ژنوتیپ‌هایی که با هیچ زوجی از هالوتیپ‌های موجود قابلیت ساخته شدن ندارند، افزایشی ایجاد نکند. (درواقع این تعداد تغییر نکند)
۳. حذف هاپلوتیپی که تکراری (نامتمایز از حداقل یک هاپلوتیپ موجود دیگر) است.

Transition Model:
با حذف یا اضافه شدن هر هاپلوتیپ، هرکدام از سه مورد موجود در state می‌تواند تغییر کند. مثلاً اگر داشته باشیم {[1112] ,[1120]} :G و نیز داشته باشیم {[1111] ,[0100] ,[1110]} :H می‌دانیم در stateی قرار داریم که به این شرح است:
.۱
هاپلوتیپ [1110] در ساختن هردو ژنوتیپ داده‌شده مجاز به مشارکت است.
هالوتیپ [0100] در ساختن هیچ یک از ژنوتیپ‌های داده‌شده مجاز به مشارکت نیست.
هالوتیپ [1111] در ساختن ژنوتیپ [1112] مجاز به مشارکت است.

.۲
یک هاپلوتیپ موجود است که در ساختن هیچ‌یک از ژنوتیپ‌های داده‌شده موثر نیست.

.۳
ژنوتیپ [1120] با هیچ یک از زوج‌های هاپلوتیپی موجود قابل ساخته‌شدن نیست.
ژنوتیپ [1112] با زوج هاپلوتیپ [1111] و [1110] قابل ساخته‌شدن است.

.۴
یک ژنوتیپ موجود است که با هیچ زوج هاپلوتیپی موجود قابل ساخته‌شدن نیست.

.۵
تعداد هاپلوتیپ‌های متمایز برابر ۳ است.

تابع Result(s, a) در این صورت به ازای هر یک از دو action ممکن می‌تواند به این شکل جواب دهد:

Result(s, a1):
با اضافه کردن هاپلوتیپ [1100]، ژنوتیپ [1120] قابلیت ساخته‌شدن با یک زوج هاپلوتیپ موجود را پیدا می‌کند و تعداد ژنوتیپ‌هایی که با هیچ زوج هاپلوتیپی موجود قابل ساخته‌شدن نیستند به صفر کاهش پیدا می‌کند.

Result(s, a2):
با حذف کردن هاپلوتیپ [0100]، در تعداد ژنوتیپ‌هایی که با هیچ زوج هاپلوتیپی موجود قابل ساخته‌شدن نیستند تغییری ایجاد نمی‌شود، اما تعداد هاپلوتیپ‌های بی‌فایده به صفر می‌رسد.

توجه داریم که چون action 3 در این حالت به دلیل نبودن هاپلوتیپ تکراری نامتعارف است، Result(s, a3) هم تعریف نشده‌است.

State Space

همه‌ی 2*m هاپلوتیپ‌های سازنده‌ی ژنوتیپ‌ها که m اندازه‌ی هر ژنوتیپ داده‌شده است فضای stateها را تشکیل می‌دهد. بسته به راه‌حل، این فضا می‌تواند 2^x که x تعداد کل نامغلوبی‌ها در ژنوتیپ‌های داده‌شده است، باشد. (x < m)

Goal Test:

زمانی که همه‌ی ژنوتیپ‌های داده‌شده قابل ساخته شدن با زوج‌هایی از میان هاپلوتایپ‌های موجود باشند، و هیچ هاپلوتایپی در مجموعه‌ی موجود بی‌فایده و یا نامتمایز نباشد، ما به یک جواب ممکن رسیده‌ایم.
هرچه تعداد هاپلوتایپ‌های موجود کمتر باشد، این جواب بهتر و بهینه‌تر است. (بر اساس Parsimony)

Step Cost:

از آنجایی که در هر مرحله یک هاپلوتیپ اضافه می‌کنیم، بسته به این که این کار را رندوم انجام می‌دهیم یا آن را از روی یک ژنوتیپ داده‌شده به‌دست می‌آوریم، هزینه‌ی اضافه کردن یک هاپلوتیپ بین ۱ تا 2^x که x مجموع کل تعداد مغلوبی‌های همه‌ی ژنوتیپ‌های داده‌شده‌است، باشد.
اگر قصد حذف یک هاپلوتیپ را داشته باشیم، چون اطلاعات تأثیرگذاری آن‌ها در ساختن ژنوتیپ‌های موجود را داریم، حداکثر با مقایسه‌ی همه‌ی هاپلوتیپ‌ها باهم یعنی هزینه‌ی تعداد هاپلوتیپ‌ها (|H|) می‌توانیم تصمیم بگیریم کدام‌یک را حذف کنیم. (هزینه‌ی حذف کردن در مراحل ابتدایی کمتر از مراحل بعدی‌ست.)

Path Cost:

بسته به تعداد مراحل طی شده تا رسیدن به هدف، هزینه‌ی مسیر می‌تواند یک ترکیب خطی از دو متغیر هزینه‌ی گفته شده در Step Cost باشد. می‌توانیم |H| را 2n در نظر بگیریم (در بدترین حالت با دو برابر تعداد ژنوتیپ‌ها می‌توان به هدف رسید). برای جایگذاری x می‌توانیم از تعداد مغلوبی‌های موجود در هر ژنوتیپ‌های داده‌شده که در بخش initial state موجود است استفاده کنیم. اگر تعداد مراحل تا رسیدن به هدف را s فرض کنیم؛

pathCost << s * (x) + s * (2n)

در ادامه پس از تعریف یک مفهوم کاربردی در الگوریتم‌های پیاده‌سازی‌شده، به شرح مدل به‌کار رفته‌ی مسأله در هر یک از الگوریتم‌ها می‌پردازیم.

Resolution

گفتیم هر ژنوتاییپ دارای m الل است که هریک از آن‌ها می‌تواند غالب (مقدار ۰ یا ۱) یا مغلوب (مقدار ۲ باشد). اگر تعداد اللهای مغلوب یک ژنوتایپ را k (که $k > 0$) در نظر بگیریم، این ژنوتایپ دارای 2^{k-1} رزولوشن است؛ یعنی این تعداد «جفت هاپلوتایپ» موجود است که از ترکیب هر جفت از آن‌ها، این ژنوتایپ ایجاد می‌شود.

Recursive Breadth First Search

در این روش با در نظر گرفتن مقداری به نام f_limit، از گشودن راه‌هایی که نتیجه‌ی بهتری از راه‌حل‌های حال حاضر نخواهند داشت، جلوگیری می‌شود.

initial state در اینجا ژنوتایپ (-1)م است با صفر رزولوشن، path خالی و مقدار f-cost برابر 4n. مقدار اولیه‌ی f-limit را بی‌نهایت قرار می‌دهیم.

Current state یک ژنوتایپ است. ژنوتایپ‌ها در این روش به ترتیب ماتریس ورودی از اندیس 0 تا n-1 بررسی می‌شوند.

Action مورد اجرا روی هر node، درواقع expand کردن آن ژنوتایپ به یک جفت هاپلوتایپ تصادفی در میان مجموعه‌ی رزولوشن‌های آن ژنوتایپ است.

مقدار heuristic در هر node، برابر است با مجموع تعداد هاپلوتایپ‌های متمایزی که تا کنون مشخص شده است با دو برابر تعداد ژنوتایپ‌هایی که هنوز بررسی نشده‌است.

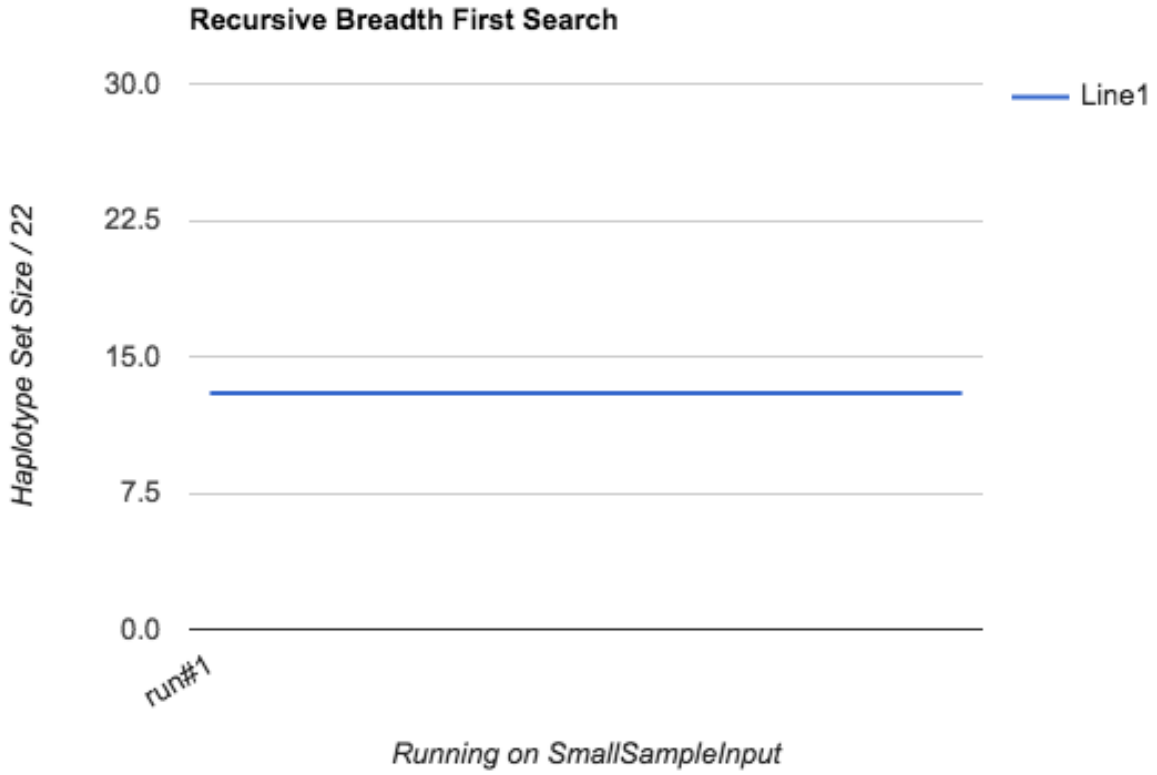
```
successor_heuristic_value = 2*(len(genotypes) - 1 - successor_genotype_index) +
len(set(current_haplotypes))
```

مقدار g-cost را در اینجا ضریبی از اندیس node درحال بررسی در نظر گرفته‌ایم. هرچه‌قدر این ضریب بزرگتر باشد، f هم بزرگتر شده، احتمال نقض f < f-limit بیشتر شده، احتمال عمیق‌تر شدن مسیر کمتر می‌شود و در نتیجه، الگوریتم به BFS شبیه‌تر می‌شود تا DFS.

```
successor_g_cost = path_cost_coefficient*(successor_genotype_index + 1)
```

مقدار f-cost هر node از مجموع دو مقدار h و g آن حاصل می‌شود.

نتیجه‌ی اجرای این الگوریتم بر روی نمونه‌ی SmallSampleInput با حداکثر جواب ۲۲ (worst case)، برابر با ۱۳ است.



Hill Climbing Method

در این روش یک مجموعه هاپلوتایپ پوشش‌دهنده‌ی همه‌ی ژنوتایپ‌های موجود (یک solution) اولیه انتخاب می‌کنیم، سپس به تعداد جایگشت‌هایی که از لحاظ منطق فرم مسأله ممکن هستند، تغییراتی در هاپلوتایپ‌های موجود اعمال می‌کنیم و مدام بررسی می‌کنیم که مجموعه به سمت بهتر شدن پیش رفته‌است یا نه.

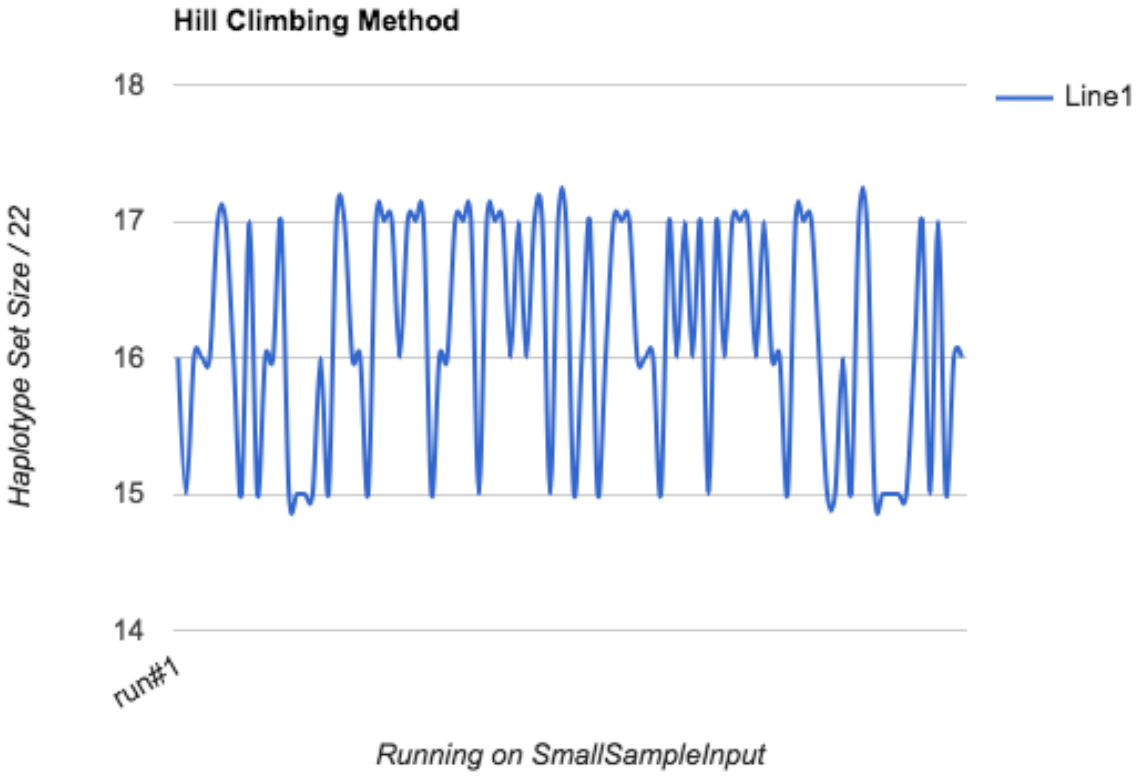
initial state در اینجا یک مجموعه 2n عضوی از هاپلوتایپ‌هایی‌ست که با تابعی ثابت از روی تک تک ژنوتایپ‌های ورودی ساخته شده‌اند. پس حتماً یک راه‌حل است اما لزوماً بهترین جواب نیست.

Current state یک مجموعه از هاپلوتایپ‌ها که راه‌حلی برای مسأله هستند (همه‌ی ژنوتایپ‌ها را پوشش می‌دهند) هرچه اندازه‌ی این مجموعه کوچکتر باشد، راه‌حل بهتری است.

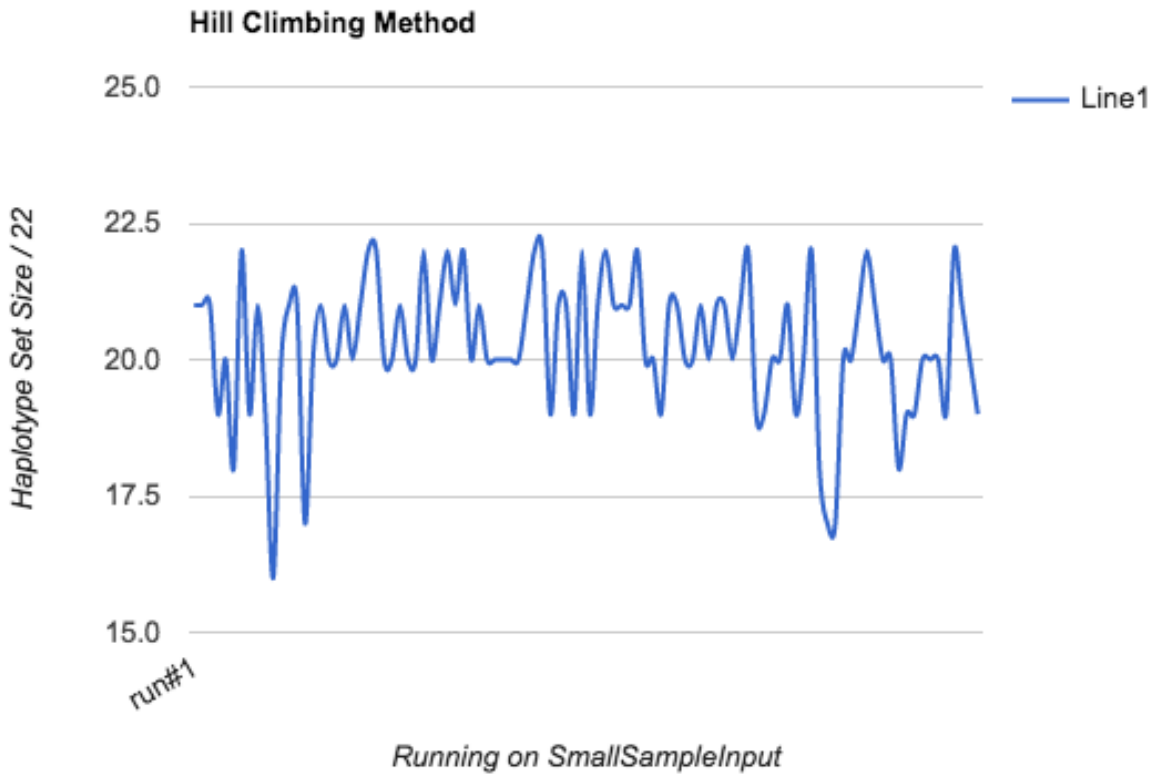
Action در هر مرحله، یکی از ژنوتایپ‌ها (یا به ترتیب ورودی، یا به ترتیبی تصادفی) انتخاب می‌شود، و یکی از الل‌های مغلوب compatible آن، toggle می‌شود. منظور از compatible بودن یک الل این است که وجود دارد الل مغلوب دیگری در ژنوتایپ دیگری که با آن هم‌موقعیت است. پس toggle کردن الل‌های متناظر در هاپلوتایپ‌های هر یک از این دو الل compatible، ممکن است موجب یکسان شدن هاپلوتایپ‌های آن‌ها و درنتیجه کوچکتر شدن مجموعه‌ی هاپلوتایپ‌های کل (راه‌حل) شود.

مقدار heuristic برای هر ژنوتایپ، برابر با تعداد ژنوتایپ‌های non-compatible با آن است. (هرچه این تعداد بیشتر باشد، شانس بهتر شدن راه‌حل پس از toggle کردن الل‌های مغلوب این ژنوتایپ کمتر خواهد بود)

نتیجه‌ی ۱۰۰ بار اجرای این الگوریتم بر روی نمونه‌ی SmallSampleInput با حداکثر جواب ۲۲ (worst case)، به صورت زیر است:



*با عوض کردن حلقه‌ی تکرار الگوریتم Hill Climbing به طوری که به جای انتخاب ژنوتایپ‌ها به ترتیب، ژنوتایپی که در هر مرحله مورد toggle شدن واقع می‌شود به طور رندوم انتخاب شود، در **۱۰۰ بار اجرا** بر روی نمونه‌ی SmallSampleInput با حداکثر جواب ۲۲ (worst case)، به جواب‌هایی به صورت زیر رسیدیم:



Simulated Annealing Method

در این روش با همان تعاریف روش قبل عمل می‌کنیم، اما برای بررسی پیشرفت مسأله و تصمیم‌گیری برای ادامه یا توقف اجرای الگوریتم، از فرمولی احتمالاتی ((exp(-(e' - e) / T) استفاده می‌کنیم. T میزان بهتر شدن راه‌حل را نشان می‌دهد؛ یعنی با طی شدن مراحل بیشتر، باید T کاهش پیدا کند و درنتیجه، احتمال ادامه‌ی اجرا در برخورد با یک عدم بهبودی در راه‌حل، کمتر شود.

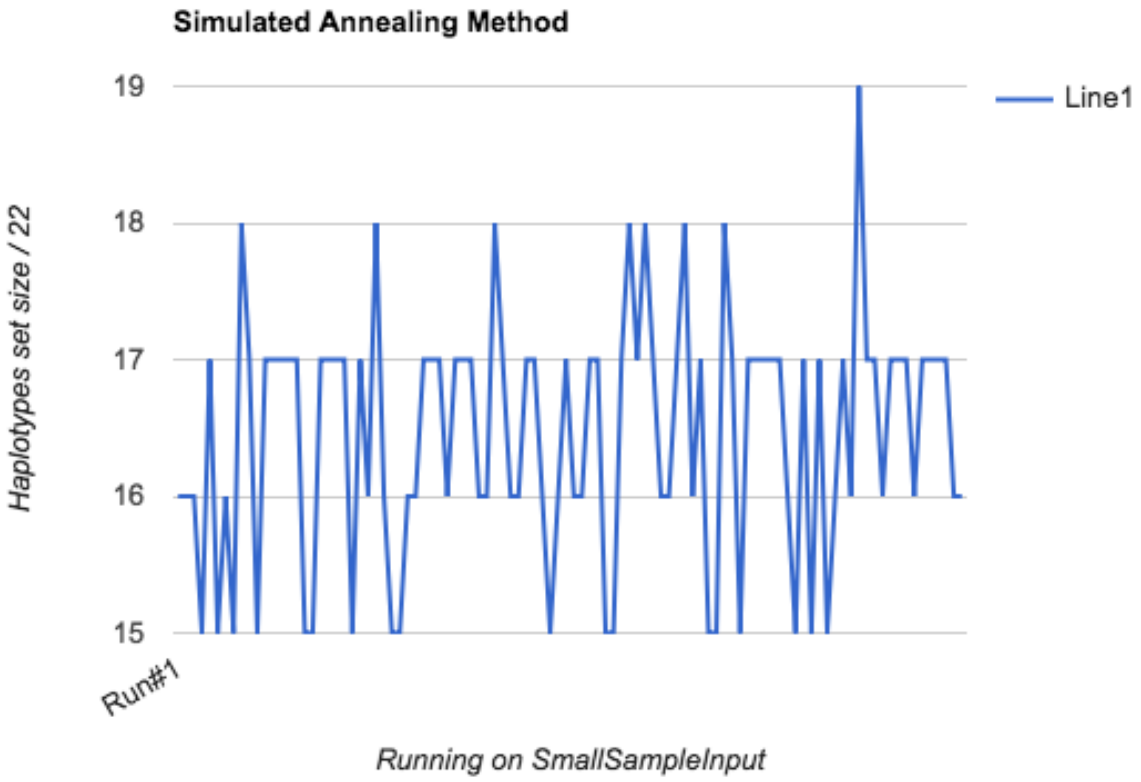
initial state مثل الگوریتم Hill climbing.

Current state مثل الگوریتم Hill climbing.

Action مثل الگوریتم Hill climbing.

مقدار heuristic مثل الگوریتم Hill climbing.

نتیجه‌ی ۱۰۰بار اجرای این الگوریتم در دفعات مختلف بر روی نمونه‌ی SmallSampleInput با حداکثر جواب ۲۲ (worst case)، به صورت زیر است:



Genetic Algorithm

در این روش ابتدا یک جمعیت از مجموعه‌های هاپلوتایپ‌های پوشش‌دهنده از روی تک تک ژنوتایپ‌های موجود می‌سازیم (مجموعه‌ای از راه‌حل‌ها)، سپس با انتخاب و cross-over هر دو راه‌حل، به راه‌حل دیگری می‌رسیم که ممکن است بهتر یا بدتر از راه‌حل‌های قبلی‌اش باشد.

initial state یک جمعیت با اندازه‌ی initial_population_size از مجموعه‌هایی‌ست که هر مجموعه شامل یک راه‌حل تصادفی یعنی مجموعه‌ای تصادفی انتخاب‌شده از بین هاپلوتایپ‌های پوشش‌دهنده‌ی ژنوتایپ‌های موجود است.

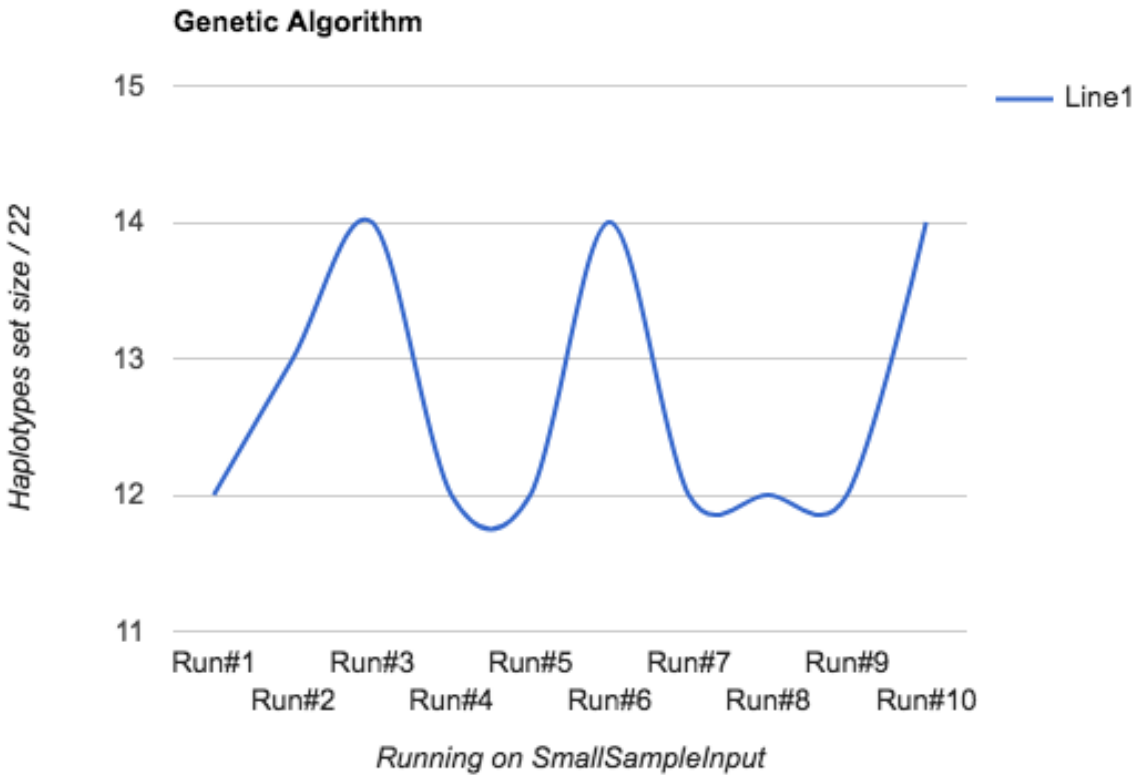
Current state یک جمعیت با اندازه‌ی initial_population_size از مجموعه‌هایی‌ست که هر مجموعه شامل یک راه‌حل یعنی هاپلوتایپ‌های پوشش‌دهنده‌ی ژنوتایپ‌های موجود است. total_score هر جمعیت، مجموع fitnessهای همه‌ی individualهای موجود در آن جمعیت است.

Action توالی اعمالی از selection، cross-over و mutation است. در هر مرحله چند جفت از بهترین individualهای موجود (منظور از بهترین، دارای بیشترین مقدار نسبی fitness است) در جمعیت را انتخاب می‌کنیم و آن‌ها را با هم cross-over می‌کنیم. برای ثابت ماندن اندازه‌ی جمعیت، به همان تعداد از بدترین individualهای موجود (کمترین مقدار نسبی fintness) در جمعیت را پیدا و حذف می‌کنیم. در نهایت با اعمال mutation، شانس جهش در راه‌حل‌ها را به جمعیت می‌دهیم.

مقدار fitness در اینجا اختلاف اندازه‌ی راه‌حل موجود، با بزرگترین راه‌حل ممکن (2n) است. پس هرچه این اختلاف بیشتر باشد، راه‌حل موجود کوچکتر بوده و مطلوب‌تر است.

len(genotypes) - len(set(haplotypes))*2

نتیجه‌ی ۱۰بار اجرای این الگوریتم با اندازه جمعیت ۱۰۰ و پس از ۵۰ نسل با selection ۵ در هر مرحله، بر روی نمونه‌ی SmallSampleInput با حداکثر جواب ۲۲ (worst case)، به صورت زیر است:



Ant Colony Optimisation Algorithm

در این روش ابتدا هر یک از هاپلوتایپ‌های هر ژنوتایپ امتیازی متناسب با اندازه‌ی رزولوشن آن ژنوتایپ می‌گیرد. سپس تعدادی مورچه مأمور می‌شوند به ترتیب به هر ژنوتایپ یک رزولوشن تصادفی آن را انتصاب کنند. این مسیر را در ذهن مورچه نگه می‌داریم. در پایان با توجه به میزان خوب بودن راه‌حل به‌دست آمده، بر می‌گردیم تا امتیاز هاپلوتایپ‌های انتخاب شده طی مسیر را به‌روزرسانی کنیم. پس باید مورچه‌های بعدی قضاوت راحت‌تری داشته باشند و در نتیجه راه‌حل‌های مطلوب‌تری را پیدا کنند.

initial state امتیاز رزولوشن‌ها به صورت (1 / number_of_resolutions) مقداردهی اولیه می‌شود.

Current state برای هر مورچه، در مسیری که در آن حرکت می‌کند، از هر رزولوشن هر ژنوتایپ برای رفتن به node بعدی، بر اساس امتیاز nodeهای پیش رو (با در نظر گرفتن احتمال خوب بودن آن‌ها) تصمیم‌گیری می‌کند.

Action
در حین پیمایش مسیر، node بعدی با استفاده از تابع توزیع نرمال شامل nodeهای بعدی و امتیاز هرکدام انتخاب می‌شود.
با پایان مسیر یک مورچه و رسیدن به یک راه‌حل، به یک total_score هم می‌رسیم که در واقع تفاضل اندازه‌ی راه‌حل موجود با اندازه‌ی بزرگترین راه‌حل ممکن (2n) است.
2*len(genotypes) - len(set(haplotypes))
این مقدار را با استفاده از فرمولی احتمالاتی تقسیم می‌کنیم و به هر یک از رزولوشن‌های انتخاب شده در طول مسیر، اضافه می‌کنیم. به این ترتیب هرچه‌قدر مسیری بهتر باشد، مقدار بزرگتری به nodeهای قرارگرفته روی مسیر اضافه می‌شود.

$$p_{xy}^k = \frac{(\tau_{xy}^\alpha)(\eta_{xy}^\beta)}{\sum_{z \in \text{allowed}_x} (\tau_{xz}^\alpha)(\eta_{xz}^\beta)}$$

مقدار fitness همان امتیاز یک رزولوشن است.

نتیجه‌ی ۱۰۰بار اجرای این الگوریتم با استفاده از ۵۰ مورچه (۵۰ بار پیمایش مسیرهای جواب در هر اجرا) بر روی نمونه‌ی SmallSampleInput با حداکثر جواب ۲۲ (worst case)، به صورت زیر است.