עבודת גמר באופטימיזציה

דוד חי רוס

9 במרץ 2021

שאלה 1

לשיטת gradient descent יש כמה חסרונות כגון:

1. יתכן שאולי לא יודעים לגזור את f, למשל אם היא מחושבת על ידי סימולציה ולא נתונה על ידי נוסחה מפורשת ולכן לא ניתן לחשב את ∇f במפורש.
   פתרון לבעיה זו היא קירוב של ∇f למשל על ידי הפרש קדמי/אחורי/מרכזי.

2. השימוש בגרדיאנט לא פותר את הבעיה של ההתקדמות במדרגות ולכן ההתכנסות של אלגוריתם זה יכול להיות מאוד איטית, אפילו במקרים פשוטים כמו פרבולה מוטה צרה.

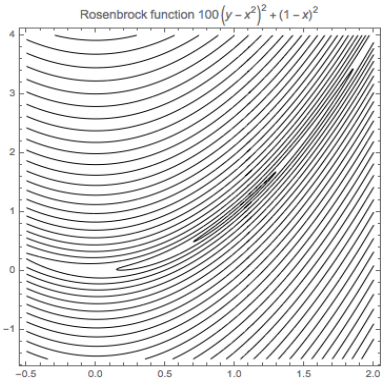
   הסבר: נניח שבחרנו כיוון כלשהוא $\hat{p}$ והגענו למינימום האמיתי של $g(\alpha)$. נסמן אותו ב α_1 ואת $x_1 = x_0 + \alpha_1 \hat{p}$.
   אז, $g'(\alpha_1) = 0$. ולפי כלל השרשרת, $0 = g'(\alpha_1) = \nabla f(x_0 + \alpha_1 p) \cdot p = \nabla f(x_1) \cdot \hat{p}$. כלומר $\hat{p} \perp \nabla f(x_1)$. במילים אחרות, המסלול של האלגוריתם מתקדם בפניות של זוויות ישרות. הבעיה של המדרגות עדיין קיימת וצריך הרבה מאוד מחזורים כדי להתקרב למינימום.

   (במקרה שבו מסתכלים על גירסה שקולה של gradient descent כאשר מבצעים $x^i = x^{i-1} - t\nabla f(x^{i-1})$
   עבור האיטרציה ה i, האלגוריתם מתכנס בקצב $O(1/k)$ ומתקיים: $f(x^k) - f(x^*) \leq \frac{\|x^0 - x^*\|^2}{2tk}$ כאשר k זו מספר האיטרציות.
   ולכן, כדי לקבל $f(x^k) - f(x^*) \leq \epsilon$, נצטרך $O(1/\epsilon)$ איטרציות.)

3. באופן דומה לסעיף הקודם, האלגוריתם יכול להתקרב למינימום אבל אף פעם לא להתכנס אליו בדיוק.

4. האלגוריתם יכול להתקע במינימום לוקאלי ולכן לא נצליח למצוא את המינימום הגלובלי של הפונקציה.

כעת נסתכל על הפונקציה הבאה:

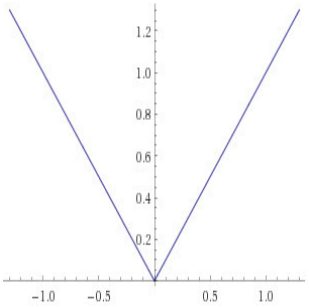


זאת פונקציית Rosenbrock המוגדרת על ידי $f(x,y) = (a-x)^2 + b(y-x^2)^2$, כאשר במקרה שלנו בחרנו $a = 1, b = 100$. יש לפונקציה הזאת מינימום גלובלי בנקודה $(x,y) = (1,1)$ ומתקיים $f(1,1) = 0$.
המינימום הגלובלי של פונקציה זו נמצא בתוך "בקעה" צרה וארוכה שצורתה פרבולית. מציאת ה"בקעה" היא משימה פשוטה יחסית, אולם בגלל צורת ה"בקעה" gradient descent מתכנס לאט למינימום הגלובלי עם צעדים קטנים ולכן נדרוש מספר גדול של איטרציות כדי למצוא את המינימום הגלובלי.

שאלה 2

סעיף א

נוכיח שלא כל החרוטים קמורים.
נסתכל על החרוט הבא:



שזו הגרף של הפונקציה $y = |x|$ כלומר $C = \{(x, y) \mid y = |x|\}$. ברור שזו היא חרוט מכיוון שמתקיים

$$\forall (x, y) \in C,\ and\ \theta \geq 0,\ \theta(x, y) = (\theta x, \theta|x|) = (\theta x, |\theta x|) \in C$$

אבל נשים לב שהקבוצה הזו לא קמורה. למשל ניתן לקחת את הנקודות: $(1, 1), (-1, 1) \in C$ אבל $(1-0.5)\cdot(-1, 1)+0.5\cdot(1, 1) = (1-\alpha)\cdot(-1, 1)+\alpha\cdot(1, 1) \notin C$ $\exists \alpha \in [0, 1],\ \alpha = 0.5$. לדוגמא, נוכל לבחור $(0, 1) \notin C$.

סעיף ב

נוכיח שהחרוט הנוצר על ידי קבוצת ווקטורים $x_1, ..., x_k \in \mathbb{R}^k$ שהוא

$$Cone(x_1, ..., x_k) \coloneqq \{x = \alpha_1 x_1 + ... + \alpha_k x_k \mid \alpha_1, ..., \alpha_k \geq 0\}$$

הוא קבוצה קמורה.

תהיו $x, x' \in Cone(x_1, ..., x_k)$ ונוכיח שלכל $\theta \in [0, 1]$ מתקיים $(1 - \theta)x + \theta x' \in Cone(x_1, ..., x_k)$.
נשים לב שמתקיים:

$$(1 - \theta)x + \theta x' \coloneqq (1 - \theta)(\alpha_1 x_1 + ... + \alpha_k x_k) + \theta(\beta_1 x_1 + ... + \beta_k x_k) = \sum_{i=1}^{k} \big((1 - \theta)\alpha_i + \theta\beta_i\big)x_i$$

כאשר $\forall i\ \alpha_i, \beta i \geq 0$. אבל, מכיוון $\theta \in [0, 1]$ מתקיים $(1 - \theta), \theta \geq 0$ ולכן נקבל $(1 - \theta)\alpha_i + \theta\beta_i \geq 0$

$$\implies \sum_{i=1}^{k} \big((1 - \theta)\alpha_i + \theta\beta_i\big)x_i \in Cone(x_1, ..., x_k)$$

ולכן קיבלנו שלכל $\theta \in [0, 1],\ x, x' \in Cone(x_1, ..., x_k)$ מתקיים $(1 - \theta)x + \theta x' \in Cone(x_1, ..., x_k)$, כלומר הקבוצה שלנו אכן קמורה.

עוד דרך:
קודם כל, נוכיח את הלמה הבא.

למה 1.1

$A \subseteq \mathbb{R}^n$ קמורה $\iff$ לכל $k \geq 2$ ולכל מספרים האי שליליים $\alpha_1, ..., \alpha_k \geq 0$ כך ש $\alpha_1 + ... + \alpha_k = 1$ ולכל $x_1, ..., x_k \in A$ מתקיים:

$$\alpha_1 x_1 + ... + \alpha_k x_k \in A$$

הוכחה:

$\Longrightarrow$
ברור, מכיוון שההגדרה של קמורה היא בדיוק מה שכתוב באגף שמאל עבור $k = 2$ ואנחנו מניחים נכונות עבור כל $k \geq 2$.
$\Longleftarrow$
נוכיח באינדוקציה:
עבור $k = 2$ זו בדיוק ההגדרה של קמורה ואנחנו מניחים את נכונותה.
כעת נניח את נכונות עבור $k - 1$ ונוכיח עבור k.
יהיו $\alpha_1, ..., \alpha_k \geq 0$ כך ש $\alpha_1 + ... + \alpha_k = 1$ ו $x_1, ..., x_k \in A$
נתבונן ב:

$$\alpha_1 x_1 + ... + \alpha_{k-1} x_{k-1} + \alpha_k x_k$$

הערה: נניח כי $\alpha_k \neq 1$ אחרת בוודאי ש $\alpha_1 x_1 + ... + \alpha_k x_k \in A$

$$= (\alpha_1 + ... + \alpha_{k-1}) \underbrace{(\frac{\alpha_1}{\alpha_1 + ... + \alpha_{k-1}} x_1 + ... + \frac{\alpha_{k-1}}{\alpha_1 + ... + \alpha_{k-1}} x_{k-1})}_{\bigstar} + \alpha_k x_k$$

כעת, נתבונן ב $\bigstar$
נשים לב שמתקיים

$$\forall i : \frac{\alpha_i}{\alpha_1 + ... + \alpha_{k-1}} \geq 0, \quad \frac{\alpha_1}{\alpha_1 + ... + \alpha_{k-1}} + ... + \frac{\alpha_{k-1}}{\alpha_1 + ... + \alpha_{k-1}} = 1$$

ולכן נקבל ש $\bigstar \in A$. אבל $x_k \in A$ וגם $\alpha_1 + ... + \alpha_k = 1$ ולכן לפי ההגדרה של קמירות מתקיים:

$$(\alpha_1 + ... + \alpha_{k-1}) \bigstar + \alpha_k x_k \in A$$

ובכך סיימנו את הוכחת האינדוקציה.

כעת, נשים לב שלפי הגדרת $Cone(x_1, ..., x_k)$, לכל $k \geq$ ולכל מספרים האי שליליים $\alpha_1, ..., \alpha_k \geq$ כך ש $\alpha_1 + ... + \alpha_k = 1$ ולכל $y_1, ..., y_k \in Cone(x_1, ..., x_n)$ מתקיים

$$\alpha_1 y_1 + ... + \alpha_k y_k \in Cone(x_1, ..., x_n)$$

ואז לפי הלמה שהוכחנו לעיל נקבל שהחרוט $Cone(x_1, ..., x_n)$ הוא קבוצה קמורה.

שאלה 3

נתבונן בבעיית האופטימיזציה הבאה:

$$\begin{aligned}
&\min_x (x_1 - \frac{3}{2})^2 + (x_2 - \frac{1}{2})^4 \\
&\text{s.t. } 1 - x_1 - x_2 \geq 0 \\
&\quad 1 - x_1 + x_2 \geq 0 \\
&\quad 1 + x_1 - x_2 \geq 0 \\
&\quad 1 + x_1 + x_2 \geq 0
\end{aligned}$$

נבדוק שתנאי KKT מתקיימים בנקודה $x^* = (1,0)$.
הגרדיאנט של f הוא:

$$\nabla f(x) = \begin{pmatrix} 2(x_1 - \frac{3}{2}) \\ 4(x_2 - \frac{1}{2})^3 \end{pmatrix} \Longrightarrow \nabla f(x^*) = \begin{pmatrix} -1 \\ -\frac{1}{2} \end{pmatrix}$$

האילוצים c_1 ו c_2 פעילים ומתקיים:

$$\nabla c_1(x^*) = \begin{pmatrix} -1 \\ -1 \end{pmatrix},\ \nabla c_2(x^*) = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

ולכן נקבל ש $\nabla c_1(x^*),\ \nabla c_2(x^*)$ בת"ל $\Longleftarrow$ תנאי LICQ מתקיים.
האילוצים c_3 ו c_4 אינם פעילים ולכן $\lambda_3^* = \lambda_4^* = 0$.
הלגרנג'יאן הוא $\mathcal{L}(x,\lambda) = f(x) - \sum_{i=1}^4 \lambda_i c_i(x)$ עם גרדיאנט

$$\begin{aligned}
&\nabla_x \mathcal{L}(x,\lambda) = \nabla_x f(x) - \sum_{i=1}^4 \lambda_i \nabla_x c_i(x) \\
&\nabla_x \mathcal{L}(x^*,\lambda^*) = \nabla_x f(x^*) - \sum_{i=1}^4 \lambda_i^* \nabla_x c_i(x^*) = \begin{pmatrix} -1 \\ -\frac{1}{2} \end{pmatrix} - \lambda_1^* \begin{pmatrix} -1 \\ -1 \end{pmatrix} - \lambda_2^* \begin{pmatrix} -1 \\ 1 \end{pmatrix}
\end{aligned}$$

נפתור את $\nabla_x \mathcal{L}(x^*,\lambda^*) = 0$ ונקבל ש:

$$\begin{aligned}
&\begin{pmatrix} -1 \\ -\frac{1}{2} \end{pmatrix} - \lambda_1^* \begin{pmatrix} -1 \\ -1 \end{pmatrix} - \lambda_2^* \begin{pmatrix} -1 \\ 1 \end{pmatrix} = 0 \\
&\lambda_1^* = \frac{3}{4},\ \lambda_2^* = \frac{1}{4}
\end{aligned}$$

כלומר $\lambda_1^*,\ \lambda_2^*$ אכן אי שליליים. לכן קיבלנו שהתנאי KKT מתקיים.

כעת נבדוק האם התנאי ההכרחי מסדר שני מתקיים:
החרוט הקריטי בנקודה $x^* = (1,0)$ נתונה על ידי

$$\begin{aligned}
T(x^*,\lambda^*) &= \left\{ w \;\middle|\; \begin{array}{ll} \nabla_x c_i(x^*) \cdot w = 0 & i \in E \\ \nabla_x c_i(x^*) \cdot w = 0 & i \in I \cap A(x^*) \wedge \lambda_i^* > 0 \\ \nabla_x c_i(x^*) \cdot w \geq 0 & i \in I \cap A(x^*) \wedge \lambda_i^* = 0 \end{array} \right\} \\
&= \left\{ w \;\middle|\; \begin{pmatrix} -1 \\ -1 \end{pmatrix} \cdot w = \begin{pmatrix} -1 \\ 1 \end{pmatrix} \cdot w = 0 \right\} = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right\}
\end{aligned}$$

ולכן נקבל שלכל $w \in T(x^*,\lambda^*)$, $w^T \mathcal{L}_{xx} w = 0 \geq 0$ ולכן התנאי ההכרחי מסדר שני אכן מתקיים בנקודה $x^* = (1,0)$.

הערה: נשים לב כי

$$\mathcal{L}_{xx}(x^*,\lambda^*) = \begin{pmatrix} 2 & 0 \\ 0 & 12(x_2-\frac{1}{2})^2 \end{pmatrix}_{x=x^*} = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}$$

מטריצה חיובית ממש ולכן $\forall z \in \mathbb{R}^2 \backslash 0 : z^T \mathcal{L}_{xx}(x^*,\lambda^*) z > 0$ ובפרט נקבל שהתנאי ההכרחי מסדר שני מתקיים.

כעת נבדוק האם התנאים המספיקים מסדר שני מתקיימים בנקודה $x^* = (1,0)$:
קודם כל נשים לב ש $\nexists\, w \neq 0 \in T(x^*,\lambda^*)$ ולכן נקבל שהתנאי המספיק מסדר שני, גרסה 1 מתקיים באופן ריק.
(כי התנאי אומר שעבור נקודה (x^*,λ^*) בה תנאי KKT מתקיים ובנוסף,

$$\forall w \neq 0 \in T(x^*,\lambda^*),\ w^T \mathcal{L}_{xx}(x^*,\lambda^*) w > 0$$

אז x^* הוא פתרון לוקלאי, וכבר חישבנו ש $T(x^*,\lambda^*) = \left\{ \begin{pmatrix} 0 \\ 0 \end{pmatrix} \right\}$)
בנוסף, $G(x^*)$ מוגדרת להיות מטריצה ששורותיה הם הגרדיאנטים של האילוצים הפעילים בנקודת x^* ולכן מתקיים:

$$G(x^*)^T = \left[\nabla c_i(x^*)\right]_{i \in A(x^*)} = \begin{pmatrix} -1 & -1 \\ -1 & 1 \end{pmatrix} \Longrightarrow dim\ Ker\big[G(x^*)\big] = 0$$

אבל אז נקבל שהתנאי $Z^T \mathcal{L}_{xx}(x^*,\lambda^*) Z \geq 0$, כאשר Z היא מטריצה שעמודותיה בסיס לגרעין של $G(x^*,\lambda^*)$, מתקיים באופן ריק. לכן קיבלנו שהתנאי המספיק מסדר שני, גרסה 2 גם כן מתקיים.

שאלה 4

נתבונן בבעיית התכנות הלינארית הבאה:

$$\begin{aligned}\min z &= 14\lambda_1 + 28\lambda_2 + 30\lambda_3 \\ \text{s.t. } 2\lambda_1 + 4\lambda_2 + 2\lambda_3 &\geq 1 \\ \lambda_1 + 2\lambda_2 + 5\lambda_3 &\geq 2 \\ \lambda_1 + 3\lambda_2 + 5\lambda_3 &\geq -1 \\ \lambda_1, \lambda_2, \lambda_3 &\geq 0\end{aligned}$$

קודם כל נוסיף את ה slack variables - S_i ונקבל:

$$\begin{aligned}\min z &= 14\lambda_1 + 28\lambda_2 + 30\lambda_3 \\ \text{s.t. } 2\lambda_1 + 4\lambda_2 + 2\lambda_3 - s_1 &= 1 \\ \lambda_1 + 2\lambda_2 + 5\lambda_3 - s_2 &= 2 \\ -\lambda_1 - 3\lambda_2 - 5\lambda_3 + S_3 &= 1 \\ \lambda_1, \lambda_2, \lambda_3 &\geq 0 \\ S_1, S_2, S_3 &\geq 0\end{aligned}$$

נשים לב שאם נציב $(\lambda_1, \lambda_2, \lambda_3) = (0, 0, 0)$ נקבל ש $(s_1, s_2, s_3) = (-1, -2, 1)$ שזה לא ייתכן מכיוון שכל ה slack variables הם אי שליליים. לכן, נוסיף artificial variables - A_i ונקבל:

$$\begin{aligned}\min z &= 14\lambda_1 + 28\lambda_2 + 30\lambda_3 \\ \text{s.t. } 2\lambda_1 + 4\lambda_2 + 2\lambda_3 - S_1 + A_1 &= 1 \\ \lambda_1 + 2\lambda_2 + 5\lambda_3 - S_2 + A_2 &= 2 \\ -\lambda_1 - 3\lambda_2 - 5\lambda_3 + S_3 &= 1 \\ \lambda_1, \lambda_2, \lambda_3 &\geq 0 \\ S_1, S_2, S_3, A_1, A_2 &\geq 0\end{aligned}$$

כעת, נשים לב שהבעיה החדשה שלנו שקולה לבעיה המקורית עבור $A_1 = A_2 = 0$ ולכן על מנת לבדוק האם נוכל להגיע למצב הזה, נעשה מינימיזציה לפונקציה $z^A = A_1 + A_2$. כלומר קיבלנו את הבעיית התכנות הלינארית הבאה:

$$\begin{aligned}\min z^A &= A_1 + A_2 \\ \text{s.t. } 2\lambda_1 + 4\lambda_2 + 2\lambda_3 - S_1 + A_1 &= 1 \\ \lambda_1 + 2\lambda_2 + 5\lambda_3 - S_2 + A_2 &= 2 \\ -\lambda_1 - 3\lambda_2 - 5\lambda_3 + S_3 &= 1 \\ \lambda_1, \lambda_2, \lambda_3 &\geq 0 \\ S_1, S_2, S_3, A_1, A_2 &\geq 0\end{aligned}$$

נמיר את הבעיה החדשה שלנו לטבלה

	λ_1	λ_2	λ_3	S_1	S_2	S_3	A_1	A_2	
A_1	2	4	2	-1	0	0	1	0	1
A_2	1	2	5	0	-1	0	0	1	2
S_3	-1	-3	-5	0	0	1	0	0	1
$-z$	14	28	30	0	0	0	0	0	0
$-z^A$	0	0	0	0	0	0	1	1	0

הפעולה הראשונה שנעשה היא חיסור השורות $1,2$ מהשרוה התחתונה כלומר $R_5 = R_5 - R_1 - R_2$ ואז נקבל

	λ_1	λ_2	λ_3	S_1	S_2	S_3	A_1	A_2	
A_1	2	4	2	-1	0	0	1	0	1
A_2	1	2	5	0	-1	0	0	1	2
S_3	-1	-3	-5	0	0	1	0	0	1
$-z$	14	28	30	0	0	0	0	0	0
$-z^A$	-3	-6	-7	1	1	0	0	0	-3

אנחנו מבצעים מינימיזציה ולכן נבחר מספר עם סימן שלילי בשורת z^A. למשל נבחר את λ_3 להיות ה pivot שלנו. מבין המספרים $\frac{1}{2},\ \frac{2}{5},\ -\frac{1}{5}$ השבר $\frac{2}{5}$ הוא הערך האי שלילי הכי קטן ולכן שוב נדרג את הטבלה שלנו ונקבל

	λ_1	λ_2	λ_3	S_1	S_2	S_3	A_1	A_2	
A_1	1.6	3.2	0	-1	0.4	0	1	-0.4	0.2
λ_3	0.2	0.4	$\boxed{1}$	0	-0.2	0	0	0.2	0.4
S_3	0	-1	0	0	-1	1	0	1	3
$-z$	8	16	0	0	6	0	0	-6	-12
$-z^A$	-1.6	-3.2	0	1	-0.4	0	0	1.4	-0.2

כאשר ביצענו את הפעולות:

$$\begin{cases} R_2 = \frac{1}{5}R_2 \\ R_1 = R_1 - 2R_2 \\ R_3 = R_3 + 5R_2 \\ R_4 = R_4 - 30R_2 \\ R_5 = R_5 + 7R_2 \end{cases}$$

עכשיו נבחר את λ_2 להיות ה pivot שלנו. מבין המספרים $\frac{1}{16}, 1, -3$ השבר $\frac{1}{16}$ הוא הערך האי שלילי הכי קטן ולכן שוב נדרג את הטבלה שלנו ונקבל

	λ_1	λ_2	λ_3	S_1	S_2	S_3	A_1	A_2	
λ_2	0.5	$\boxed{1}$	0	-0.3125	0.125	0	0.3125	-0.125	0.0625
λ_3	0	0	1	0.125	-0.25	0	-0.125	0.25	0.375
S_3	0.5	0	0	-0.3125	-0.875	1	0.3125	0.875	3.0625
$-z$	0	0	0	5	4	0	-5	-4	-13
$-z^A$	0	0	0	0	0	0	1	1	0

כאשר ביצענו את הפעולות:

$$\begin{cases} R_1 = \frac{5}{16}R_1 \\ R_2 = R_2 - 0.4R_1 \\ R_3 = R_3 + R_1 \\ R_4 = R_4 - 16R_1 \\ R_5 = R_5 + 3.2R_1 \end{cases}$$

נשים לב שקיבלנו basic feasible solution שזו $(\lambda_1, \lambda_2, \lambda_3, S_1, S_2, S_3) = (0, \frac{1}{16}, \frac{3}{8}, 0, 0, \frac{49}{16})$ לבעיה המקורית שלנו. כעת נוכל לזרוק את העמודות של A_1, A_2 מהטבלה שלנו ונקבל

	λ_1	λ_2	λ_3	S_1	S_2	S_3	
λ_2	0.5	1	0	-0.3125	0.125	0	0.0625
λ_3	0	0	1	0.125	-0.25	0	0.375
S_3	0.5	0	0	-0.3125	-0.875	1	3.0625
$-z$	0	0	0	5	4	0	-13

לא צריכים להמשיך מכיוון שרואים שכל המקדמים ב $-z$ חיובים, כלומר הפתרון הנוכחי $(\lambda_1, \lambda_2, \lambda_3, S_1, S_2, S_3) = (0, \frac{1}{16}, \frac{3}{8}, 0, 0, \frac{49}{16})$ משיגה את הערך המינימלי של $z = 13$.
בנוסף, נשים לב שבעיית LP הזו היא הבעיה הצמודה (dual) לבעיה:

$$\begin{aligned} \max z = x_1 + 2x_2 - x_3 \\ \text{s.t.} \quad 2x_1 + x_2 + x_3 \leq 14 \\ 4x_1 + 2x_2 + 3x_3 \leq 28 \\ 2_1 + 5x_2 + 5x_3 \leq 30 \\ x_1, x_2, x_3 \geq 0 \end{aligned}$$

שאותו פתרנו בהרצאה וקיבלנו שהמקסימום הוא $z = 13$. לכן נקבל שלפי המשפט:

$$P \ finite \iff D \ finite \ and \ their \ optimum \ is \ equal$$

(כאשר P הוא הבעיה המקורית ו D הוא הבעיה הצמודה), שהערך $z = 13$ הוא אכן המינימום של הבעיה שלנו.

הערה: בשאלה זו השתמשנו בשיטת Simplex Two-Phase כדי לפתור את בעיית התכנות הלינארית שלנו.

1 Introduction

1.1 The Traveling Salesman Problem

The traveling salesman problem (TSP) is one of the oldest problems in combinatorics and comes from from as early as 1832 [2]. The problem is defined as "given n cities and the distances between them, find a shortest route that visits each city exactly once".

The TSP is an NP-complete problem, which means that it can not be solved efficiently by any known algorithm [3]. The reason for the TSP being a hard problem is reasonably intuitive just with a quick example: Imagine if we have3 cities. The possible orders in which we visit these cities is that we start with any of the three, then select one of the two remaining ones and lastly visit the only one left. In mathematical terms that would be $3 \times 2 \times 1 = 3! = 6$ possible solutions to search through to find the best one.If the amount of cities increases, the amount of possible routes increases rapidly. Just by doubling the amount of cities we get $6! = 720$ different routes. Just 10 cities yield $10! \approx 3.6$ million different routes. Therefore, searching through all the possible solutions to find the best one quickly becomes unfeasible as the number of cities increase. Instead, we have to resort to optimization algorithms that approximate the best value and limit what part of the solution space to search through by utilizing heuristics and other "smart" problems solving methods.The TSP is one of the most studied optimization problems in the world partly perhaps due to its catchy name, but primarily due to the fact that it is a simple sounding problem but has yet to get a general solution.

The TSP can also be represented by a graph with vertices and edges, which means that it is applicable to many scientific areas such as genetics, neuroscience, telecommunication and computer science, thus furthering its appeal [4].

1.2 Genetic Algorithms

The Genetic Algorithm (GA) is part of an area called evolutionary computation. Evolutionary computation takes inspiration from biological evolution in order to compute solutions for various types of problems [1]. One of the main appeals of evolution when it comes to solving computational problems is the way it can be used as a method of "designing innovative solutions to complex problems" [1].

The GA was presented by Holland in 1975 [1, 5] and in particular takes inspiration from the way genes are passed on by individuals in one generation to their offspring in the next. The idea is that if the best performing members of a generation are allowed to reproduce more into the next generation than worse performers, we iteratively get better and better solutions to our problem each generation. This should then yield iteratively better performing members for each new generation until an optimal or "good enough" solution has been found.

We chose to use GA's to tackle the TSP for a variety of reasons:

- Genetic algorithms are useful for NP-hard problem such as the TSP because, though this method does not ensure optimal solutions, it often gives good approximations to the optimal solutions with relatively fast run-times.
- Genetic algorithms search parallel from a population of points. Therefore it has the ability to avoid being trapped in local optimal solutions of which the TSP has many.
- Genetic algorithms use a fitness score, which is obtained from objective functions, without other derivatives or auxiliary information.
- Genetic algorithms work on the Chromosome which is an encoded version of potential solutions' parameters, rather than the parameters themselves.

1.3 GA Terminology

GA's use a few important concepts. These are described briefly here.

1.3.1 Model

An **individual**, also known as a **chromosome**, is a single instance of a solution to a problem. In the TSP, an individual can be represented as a single route within the solution space. As an example, this 8-city route is one individual:

$$(7, 2, 4, 5, 6, 3, 1, 8)$$

A **gene** is a part of an individual. The gene usually represents a trait within the individual, and is commonly represented with single bits. However, they can be represented in other ways. An example is as letters in a protein structure. In terms of the TSP, a gene can represent a certain city visited at a certain position in the order of cities. As an example, the bolded city 4, in the third position, is a single gene in this individual (as is every other city):

$$(7, 2, \mathbf{4}, 5, 6, 3, 1, 8)$$

A **population** is a collection of individuals in a single generation. As an example, here is a population of size 3:

$$(7, 2, 4, 5, 6, 3, 1, 8)$$
$$(1, 2, 5, 8, 6, 4, 3, 7)$$
$$(5, 3, 8, 7, 2, 1, 4, 6)$$

A **generation** is a single instance of a population within a single iteration of searching the solution space. The first generation is completely random, while the rest of the generations are heavily generated from the previous one. As an example, the example above not only is an example of a population, but also shows how it can look in a single generation.

Execution The **fitness function** is the measurement of how well an individual performs at the objective. What type of value is desired depends on the problem. In the TSP, the total distance traveled is usually used as the measurement of performance. A lower distance traveled equates to better performance since the TSP is a minimization problem.

The **population size** is the amount of individuals in a generation. A bigger population size makes the program run slower, but gives a better chance for a single generation to find better solutions. The trade-off in increasing the population size usually means getting better results but having a slower program which converges to its optimal solution slower with respect to time.

Selection is the process of choosing which individuals in a generation to cross over into new individuals in the next generation. Selection can be implemented in simple ways, such as just selecting a percentage of the best performers within a generation. Selection can also be done in more advanced ways, such as giving each member a portion of a roulette wheel where every members portion size depends on its fitness level, and then spinning the wheel several times in order to select parents.

Retain rate is the portion of individuals selected to be parents for the next generation. If all individuals of a generation gets to reproduce equally, the next generation is basically random.

Crossover is the operation of structurally generating a new individual from parents in a previous generation. There are a substantial number of ways in which one can combine the genes of the parents into a new individual.

Mutation is the operation of randomly changing the genes somehow when generating a new individual or passing an individual from one generation to the next. A mutation within a TSP representation could be swapping the order in which one visits two cities. There are several types of possible mutations. It could be as simple as simply swapping two cities or more complicated, like selecting a small subset of the solution and reversing it.

Mutation rate is the probability of an individual having its genes mutated.The details on how this essay implemented the GA is described in the next section.

1.3.2 Algorithm Description

A short description of a GA can be defined as:

1. Initialize random population from solution space.
2. Calculate the fitness of each individual in the population
3. Reproduce the generation, using a combination of these methods whilst ensuring that the next generation has the same population size:
    - Select a portion of the most fit individual and simply pass them onto the next generation.

- Generate offspring by taking fit individuals and combining them using crossover/
- Generate offspring by taking any individual and mutating it.
- Generate new individuals randomly.

4. If a good enough solution is found, or if another termination criteria such as a time limit or maximum amount of generations is reached, return the result. Otherwise, go to step 2.

2 Method

2.1 Genetic Algorithm Implementation

In order to explore our algorithm, a framework for the GA was built. The programming language used was Python, mainly due to the preference of the authors. In this section, we will describe how the GA was implemented and how we represented the TSP within it.

2.1.1 Setup

The entire project was built from the ground up in Python and is currently available in a GitHub repository [1]. In order to test our GA implementation we chose an existing instances of a TSP problem from TSPLIB.

The chosen TSP instance was:

City	TSP Problem Name	Cities	Optimal Length
Berlin	berlin52	52	7542

A single individual within a generation was represented as a path through these cities, mainly since path representation are probably the most intuitive representation method for a path in the TSP, and also the one preferred by the authors. As an example, a numerical path through a TSP instance of 8 cities would be $(1, 2, 3, 4, 5, 6, 7, 8)$, which means that each city is visited in order,and finally one would return to the first city.

The GA was initialized by randomizing the entire population. Then the fitness of each individual was determined by calculating the inverse of the total distance through the TSP for each individual, where higher was better. We then ranked all the individuals in the generation, saved the result and then reproduced the next generation.

The first generation was completely randomly generated. All of the subsequent generations were built up in two parts, by using the following three steps:

1. First we simply passed on the n best individual of the generation, without mutation or crossover.
2. Then we selected a number of parents from the previous generation, using fitness proportionate selection.
3. Then we generated the individuals for the next generation using crossover on the parents.

After creation, an individual has the chance to randomly mutate. The implementation of mutation is described in section 2.1.4.

The reason for passing on the best individuals from a generation to the next is that one is guaranteed to never forget the best result one has found thus far. If we ensure that our population size is large enough, it seems reasonable to expect these members to not cause premature convergence.

The rest of the new generation was generated by crossing over individuals from the previous generation, where the better performers were more likely to reproduce. This is the core concept of GA's, and the part that keeps it improving iteratively. The method of crossover is described in section 2.1.3.

2.1.2 Selection

We chose to use proportional roulette wheel selection, which is when when each individual gets a portion of a roulette wheel where the portion is entirely proportional to its fitness level. We then compare a randomly drawn number to these portions to select our mating pool.

Additionally, we implemented the use of elitism. With elitism, the best performing individuals from the population will automatically carry over to the next generation, ensuring that the most successful individuals persist.

[1] https://github.com/David-C-Ross/TSP

2.1.3 Crossover

When generating offspring from previous generations, we used the crossover function named OX1. This order crossover operator was first presented by Davis in 1985 [11]. It generates offspring by choosing a part/sub-tour of the first parent and then placing the rest of the cities in the order they appear in the second parent. Here is an example taken from Larrañaga et al. [6]: Consider two parents:

$$(1, 2, 3, 4, 5, 6, 7, 8)$$
$$(2, 4, 6, 8, 7, 5, 3, 1)$$

First, decide where to cut out the sub-tours. We choose to cut between city 2 and 3, and between city 5 and 6:

$$(1, 2 \mid 3, 4, 5 \mid 6, 7, 8)$$
$$(2, 4 \mid 6, 8, 7 \mid 5, 3, 1)$$

Then, we place the sub-string into the offspring at the same position.

$$(*, * \mid 3, 4, 5 \mid *, *, *)$$
$$(*, * \mid 6, 8, 7 \mid *, *, *)$$

And finally, we put the remaining cities that are needed in offspring 1 by starting after the second cutoff point, and placing the cities in the order they are placed in parent 2, and vice versa. Then we get the following children:

$$(7, 8 \mid 3, 4, 5 \mid 1, 2, 6)$$
$$(3, 1 \mid 6, 8, 7 \mid 2, 4, 5)$$

The reason behind choosing OX1 was that it is a classic technique that at the same time seems to perform well at solving the TSP, according to the results of Larrañaga et al. [6].

2.1.4 Mutation

When generating new individuals, they are given a chance to randomly mutate. This helps add diversity to the generation in order to better search the solution space. The selected mutation function we used was ISM. Insertion Mutation (ISM) is credited to a report by Fogel in 1988 and Michalewicz in 1992 [6]. It selects a city in the individual and inserts it randomly somewhere else within the Individual. As an example:

$$(1, 2, 3, \mathbf{4}, 5, 6, 7, 8)$$
$$(1, 2, 3, 5, 6, 7, \mathbf{4}, 8)$$

3 Results

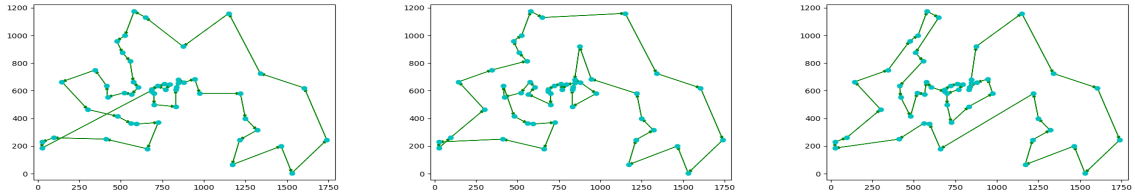
In this section we present the results of our GA applied to the berlin52 data set and compare its performance to both Simulated Annealing (SA) and the basic greedy algorithm approach.

3.1 Optimal Solution

Table 1 shows the best score (minimum route length) achieved on the berlin52 data set for each algorithm. We may notice that our basic greedy algorithm performs the most poorly amongst the three, although its run-time was significantly shorter than the other two algorithms implemented here. Additionally, Simulated Annealing outperforms both the GA and the greedy algorithm with a run-time of approximately double that of the greedy algorithm whilst our GA manages to achieve a roughly equal (albeit slightly worse) result than SA though its run-time is by far the longest, at approximately triple that of the greedy algorithm.

Algorithm	**Distance (meters)**	**Elapsed Time (seconds)**
Genetic	8463.668105	89.118811
Simulated Annealing	8441.988664	52.985237
Greedy	9182.191555	23.738252

Table 1: Best score achieved on the berlin52 data set for each algorithm.

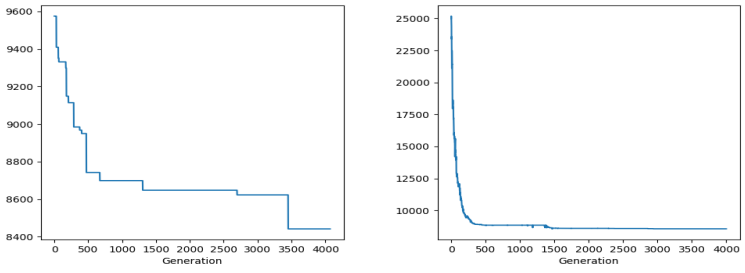


(a) Greedy (b) SA (c) GA

Figure 1: Optimal routes found by each algorithm

3.2 Convergence

In figure 1, the difference in convergence rates between the implementations of our GA and SA is shown. The traditional GA implementation converges much faster than SA until around generation 500, where the convergence rate of our GA then slows dramatically.



(a) (b)

Figure 2: Convergence to local minima visualization: (a) Simulated Annealing (b) Genetic Algorithm

References

[1] Melanie Mitchell. *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press, 1996.

[2] Alexander Schrijver. *"On the History of Combinatorial Optimization(Till 1960)"*. In: *Discrete Optimization*. Ed. by G.L. Nemhauser K. Aardaland R. Weismantel. Vol. 12. Elsevier, 2005, pp. 1–68 Annalen der Physik, 322(10):891–921, 1905.

[3] Jon M. Kleinberg and Éva Tardos. *Algorithm design*. Addison-Wesley,2006.

[4] David L. Applegate et al. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton,NJ, USA: Princeton University Press, 2007.

[5] John H. Holland. *Adaptation in Natural and Artificial Systems*. second edition, 1992. Ann Arbor, MI: University of Michigan Press, 1975.

[6] P. Larrañaga et al. *"Genetic Algorithms for the Travelling Salesma nProblem: A Review of Representations and Operators"*. In: *Artificial Intelligence Review* 13.2 (1999), pp. 129–170.