מיני פרוייקט – נושאים באופטימיזציה קומבינטורית : בעיית ספיקות מקסימלית (Max SAT)

מרצה הקורס : פרופ' דניאל ברנד

מגישים : טל יצחק (204260533), חיים סבן (308000371)

כלל הקוד בפרויקט נכתב בJava, וזמין ב-GitHub :

github.com/talitz/Combinatorial-Optimization-MaxSAT

מטרת ונושא הפרוייקט : להבין את הסטטיסטיקה של מופעים אקראיים של בעיית הספיקות המקסימלית. בפרויקט זה נבנה מופעים אקראיים של הבעיה ונבחן ביצועים ממוצעים של אלגוריתמים שונים.

בעיית Max SAT – הגדרה : נתון ביטוי φ בצורת CNF (פסוקיות של ORים וביניהן ANDים) עם n משתנים x1,x2,…,xn. המטרה היא למצוא השמה למשתנים שמביאה למקסימום את מספר הפסוקיות המסופקות.

אלגוריתם קירוב (Approximation Algorithm) : אלגוריתם שמוצא פיתרון שאינו בהכרח פיתרון אופטימלי לבעיה נתונה, אלא פיתרון **שקרוב** לפיתרון אופטימלי. על פי רוב מודדים קירוב של אלגוריתם בהתאם ליחס בין הפיתרון שנמצא ע"י האלגוריתם לבין הפיתרון האופטימלי.

הערה : נבחן את האלגוריתמים שנציע גם מבחינת יעילות זמן ריצה וגם מבחינת אחוז הצלחה גבוה של האלגוריתם.

** נציין שבכל ההרצות, נגדיר את m להיות מספר הפסוקיות ו-n מספר המשתנים. נתעניין במקרים בהם היחס m/n יהיה באיזור 4.26 (מתחת לערך זה, ההסתברות של סיפוק מופע של הבעיה ישאף ל1, מעל לערך זה ההסתברות של סיפוק מופע של הבעיה ישאף ל-0), כמו כן r=3, כלומר אורך הפסוקיות הוא 3.*

בפרוייקט זה נממש מספר אלגוריתמי קירוב לבעיית Max SAT:

1) אלגוריתם א':

באלגוריתם זה נגריל לכל משתנה ערך בוליאני (True או False) בהסתברות שווה.

ניתוח האלגוריתם : נתבונן בפסוקית C_j בעלת k ליטרלים. ההסתברות של פסוקית זו לא להסתפק היא $\frac{1}{2^k}$, לכן ההסתברות המשלימה (ההסתברות של פסוקית זו להיות מסופקת) תהיה בדיוק $1-\frac{1}{2^k}$.

לאלגוריתם זה יחס קירוב בממוצע של 1/2 לבעיית *Max SAT* : אם נגדיר משתנה מקרי *Xi* שמקבל 1 כאשר הפסוקית *i* מסתפקת ו-0 אחרת, נוכל להתבונן בתוחלת של *Xi* :

$$E[Xi] = 0 \cdot \Pr(Xi = 0) + 1 \cdot \Pr(Xi = 1) = 1 - \frac{1}{2^k}$$

נשים לב שככל ש-k גדל, כך התוחלת תגדל, ועבור k=1, התוחלת שווה ל-1/2 בדיוק.

עבור מופע של הבעיה עם m פסוקיות, נקבל מליניאריות התוחלת:

$$E[\sum_{i=1}^{m} \mathrm{Xi}] = \sum_{i=1}^{m} \mathrm{E[Xi]} = m(1 - \frac{1}{2^k})$$

נציג את תוצאות הפלט לאחר ההרצה והמימוש של האלגוריתמים ב-Java:

```
Console   @ Javadoc   Declaration   Problems   Search
<terminated> Test (1) [Java Application] C:\Program Files\Java\jre1.8.0_73\bin\javaw.exe (Jan 3, 2017, 12:23:48 AM)
A algorithm:
Number of literals per clause: 3
Number of variables (n): 10
Number of clauses (m): 42
m/n ratio: 4.2
Number of CNF's: 100
Satisfied clauses (in percentage): 87.28571428571429%
```

```
Console   @ Javadoc   Declaration   Problems   Search
<terminated> Test (1) [Java Application] C:\Program Files\Java\jre1.8.0_73\bin\javaw.exe (Jan 3, 2017, 12:25:55 AM)
A algorithm:
Number of literals per clause: 3
Number of variables (n): 10
Number of clauses (m): 80
m/n ratio: 8.0
Number of CNF's: 100
Satisfied clauses (in percentage): 86.8625%
```

נבחן את התוצאות לעומת ערך התוחלת עבור m=42 וk=3:

כלומר מספר הפסוקיות המסתפקות באחוזים הוא $\left(\frac{36.75}{42}\right) 100 = 87.5\%$, מאוד קרוב למה שקיבלנו בהרצות.

ניתוח זמן הריצה של האלגוריתם: באלגוריתם זה עוברים על כל המשתנים, ומגרילים ערך לכל אחד בהסתברות חצי. כלומר המעבר על הקלט הוא יחיד, ב- $O(n)$.

נרצה לשפר את דיוק האלגוריתם: נריץ את האלגוריתם מספר קבוע של פעמים p, ובכל פעם נבדוק מה הכמות המקסימלית של פסוקיות שהסתפקו, ואותה נחזיר.

זמן הריצה במקרה זה יהיה $O(p \cdot n) = O(n)$.

עבור $p = 100$, נקבל את התוצאות הבאות:

```
Console  @ Javadoc  Declaration  Problems  Search
<terminated> Test (1) [Java Application] C:\Program Files\Java\jre1.8.0_73\bin\javaw.exe (Jan 3, 2017, 12:08:22 AM)
A algorithm:
Number of literals per clause: 3
Number of variables (n): 10
Number of clauses (m): 42
m/n ratio: 4.2
Number of CNF's: 100
Satisfied clauses (in percentage): 98.4047619047619%
```

```
Console  @ Javadoc  Declaration  Problems  Search
<terminated> Test (1) [Java Application] C:\Program Files\Java\jre1.8.0_73\bin\javaw.exe (Jan 3, 2017, 12:06:57 AM)
A algorithm:
Number of literals per clause: 3
Number of variables (n): 10
Number of clauses (m): 80
m/n ratio: 8.0
Number of CNF's: 100
Satisfied clauses (in percentage): 96.0%
```

ניתן לראות שמספר הפסוקיות שמסתפקות עולה משמעותית, ולא שילמנו בזמן ריצה יקר יותר (עדיין ליניארי באורך הקלט).

2) <u>אלגוריתם ב':</u> בהינתן קלט, מגרילים ליטרל l ובודקים איזה שמה ל-l (True/False) תגדיל את תוחלת מספר הפסוקיות המסתפקות. נבצע זאת לכל הליטרלים, עד שלא ישארו ליטרלים לבצע עבורם השמה יותר

<u>דוגמא:</u> עבור ה-CNF בעל הפסוקית הבודדה $\varphi = (x1\ or\ x2\ or\ \neg x3)$, אם נגריל את הליטרל x1, נתבונן ב-2 המקרים:

<u>1)</u> x1=true, ולכן התרומה שלו עבור התוחלת תהיה 1/8.

<u>2)</u> x1=false, ולכן התרומה שלו עבור התוחלת תהיה 0.

כלומר 1/8 גדולה מ-0 ולכן נעדיף לבחור בהצבה x1=true.

```
Console  @ Javadoc  Declaration  Problems  Search
<terminated> Test (1) [Java Application] C:\Program Files\Java\jre1.8.0_73\bin\javaw.exe (Jan 3, 2017, 12:10:32 AM)
B algorithm:
Number of literals per clause: 3
Number of variables (n): 10
Number of clauses (m): 42
m/n ratio: 4.2
Number of CNF's: 100
Satisfied clauses (in percentage): 97.54761904761905%
```

```
Console  @ Javadoc  Declaration  Problems  Search
<terminated> Test (1) [Java Application] C:\Program Files\Java\jre1.8.0_73\bin\javaw.exe (Jan 3, 2017, 12:12:28 AM)
B algorithm:
Number of literals per clause: 3
Number of variables (n): 10
Number of clauses (m): 80
m/n ratio: 8.0
Number of CNF's: 100
Satisfied clauses (in percentage): 95.4125%
```

ניתוח זמן הריצה של האלגוריתם: מימשנו את האלגוריתם באופן הבא: הגרלנו ליטרל באופן אקראי, וחישבנו בנפרד את התוחלת עבור 2 ההשמות האפשריות עבורו: True ו-False: עשינו זאת ע"י מעבר על הקלט פעם אחת. לאחר מכן נמחק את כל הפסוקיות המסופקות, ונמשיך בתהליך עד שלא יישארו ליטרלים יותר להגריל. סה"כ, $O(tn)$, כאשר t זהו מספר הליטרלים, ו-n אורך הקלט.

3) <u>אלגוריתם ג':</u>

1. בהינתן קלט, עבור על כל המשתנים נספור לכל משתנה Xi את כמות ההופעות שלו ושל שלילתו (למשל, המשתנה X5 יכול להופיע 5 פעמים, ושלילתו not(X5) יכולה להופיע 0 פעמים).
2. אם כמות ההופעות של Xi גדולה מכמות ההופעות של not(Xi), בחר Xi=True, אחרת בחר Xi=False.
3. חזור על התהליך עד שלא נשארו יותר משתנים.

```
Console  @ Javadoc  Declaration  Problems  Search
<terminated> Test (1) [Java Application] C:\Program Files\Java\jre1.8.0_73\bin\javaw.exe (Jan 3, 2017, 12:14:46 AM)
C algorithm:
Number of literals per clause: 3
Number of variables (n): 10
Number of clauses (m): 80
m/n ratio: 8.0
Number of CNF's: 100
Satisfied clauses (in percentage): 93.9875%
```

```
Console  @ Javadoc  Declaration  Problems  Search
<terminated> Test (1) [Java Application] C:\Program Files\Java\jre1.8.0_73\bin\javaw.exe (Jan 3, 2017, 12:13:43 AM)
C algorithm:
Number of literals per clause: 3
Number of variables (n): 10
Number of clauses (m): 42
m/n ratio: 4.2
Number of CNF's: 100
Satisfied clauses (in percentage): 94.78571428571429%
```

ניתוח זמן הריצה של האלגוריתם: מימשנו את האלגוריתם באופן הבא: עברנו על הקלט פעם אחת, ויצרנו מערך מונים המכיל לכל ליטרל והיפוכו – את כמות ההופעות שלו (סה"כ $O(n)$, אורך הקלט).

כעת נותר להציב ערכים בליטרלים לפי כמות ההופעות של הליטרל לעומת היפוכו, כפי שמתואר באלגוריתם (סה"כ $O(t)$). בסה"כ נקבל זמן ריצה $O(n+t)$.

4. <u>אלגוריתם ד':</u> האלגוריתם הוא אלגוריתם חמדן, כאשר כלל הבחירה החמדני הוא כדלקמן:
   1. בהינתן קלט, בחר ליטרל l שכמות המופעים שלו בפסוקיות היא מקסימלית.
   2. בחר את הערך של l להיות (True/False) כך שכל הפסוקיות שמכילות את l מסתפקות.
   3. מחק את כל הפסוקיות המסופקות לאחר הצבת הערך של l.
   4. חזור ל1 עד שנסיים לעבור על כל הליטרלים.

```
Console   @ Javadoc   Declaration   Problems   Search
<terminated> Test (1) [Java Application] C:\Program Files\Java\jre1.8.0_73\bin\javaw.exe (Jan 3, 2017, 12:19:41 AM)
D algorithm:
Number of literals per clause: 3
Number of variables (n): 10
Number of clauses (m): 42
m/n ratio: 4.2
Number of CNF's: 100
Satisfied clauses (in percentage): 96.95238095238095%
```

```
Console   @ Javadoc   Declaration   Problems   Search
<terminated> Test (1) [Java Application] C:\Program Files\Java\jre1.8.0_73\bin\javaw.exe (Jan 3, 2017, 12:16:02 AM)
D algorithm:
Number of literals per clause: 3
Number of variables (n): 10
Number of clauses (m): 80
m/n ratio: 8.0
Number of CNF's: 100
Satisfied clauses (in percentage): 95.25%
```

ניתוח זמן הריצה של האלגוריתם: בשלב 1 אנו עוברים על הקלט פעם אחת למציאת הליטרל הנדרש ב- $O(n)$, בשלב 2 נבחר ערך לליטרל ב- $O(1)$. שלב 3 דורש מעבר נוסף על כל הקלט פעם אחת על מנת למחוק את כל הפסוקיות שהסתפקו, ב- $O(n)$ במקרה הגרוע. עד כה הגענו לזמן ריצה של $O(n+1+n)=O(n)$. כעת נצטרך לחזור על התהליך במקרה הגרוע כמספר המשתנים t: $O(tn)$.

<u>הערה:</u> ניתן אף לייעל עוד את האלגוריתם אם נחשב לכל ליטרל והיפוכו את יחס ההופעות שלהן, ונבחר ערך לליטרל בכל איטרציה לפי היחס הכי גדול. כלומר, אם x1 מופיע 10 פעמים ושלילתו מופיעה פעם אחת, אך x2 מופיע 20 פעמים ושלילתו מופיעה 19 פעמים, נעדיף לבחור x1=true (כי היחס 10/1 גדול יותר מהיחס 20/19).

כעת, נבחן היבטים אחרים לבעיית Max SAT. ראשית נגדיר כמה מושגים:

אופטימום לוקאלי (מקומי) של בעיית אופטימיזציה הוא פיתרון שהוא אופטימלי (מקסימלי/מינימלי) בהינתן סביבה מסויימת של פתרונות פוטנציאלים.

בבעיית Max SAT: נוכל להגדיר את האופטימום המקומי להיות השמה ספציפית (נקודה) כך ששום שינוי בודד (עד כדי "ביט" T/F) לא ישפר את כמות הפסוקיות המסתפקות.

כמו כן, נתייחס למושג "שכן" של נקודה כהשמה השונה מההשמה של הנקודה רק בהצבה אחת בודדת. לדוגמה: עבור ההצבה (T,T,F), למשל, שכן אפשרי יכול להיות (T,T,T) (אם נתייחס לוקטור כאל וקטור הצבות של המשתנים).

סטיית תקן – מדד סטטיסטי לתיאור הפיזור של ערכי קבוצת נתונים סביב הממוצע שלהם. במונח "סטייה" מתכוונים למרחק בין ערך בקבוצה לבין הממוצע.

כעת נממש 2 אלגוריתמים למציאת נקודות אופטימום. לכל אלגוריתם שנממש נרצה לענות על השאלות הבאות:

א. כמה נקודות אופטימום יש בהינתן ביטוי φ בצורת CNF כלשהו?
ב. מה הביצועים של נקודות אלו? (ממוצע וסטיית תקן).
ג. כמה הנקודות הללו "דומות" (נבחן זאת ע"י אחוז המשתנים בעלי אותה ההשמה)?

1) <u>אלגוריתם א':</u> נמצא נקודות אופטימום מקומי ע"י יצירת נקודות רנדומליות (1000) ובדיקה האם הם נקודות אופטימום מקומי.

<u>תוצאות הרצה:</u>

```
Search   Console   Progress
<terminated> MainAlgorithmsRunner [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.e
First Local Optimum Algorithm:
Number of literals per clause: 3
Number of variables (n): 10
Number of clauses (m): 42
Number of CNF`s: 1
Number of Local Optimum Points: 26
Local Optimum Points Performance:
Satisfied Clauses (Average): 40.38461538461539
Satisfied Clauses (Standard Deviation): 0.963843391241667
Similarity of Local Optimum Points (percentage): 53.01538461538462%

----------------------------------------
```

```
<terminated> MainAlgorithmsRunner [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javav
First Local Optimum Algorithm:
Number of literals per clause: 3
Number of variables (n): 10
Number of clauses (m): 42
Number of CNF`s: 1
Number of Local Optimum Points: 23
Local Optimum Points Performance:
Satisfied Clauses (Average): 40.21739130434783
Satisfied Clauses (Standard Deviation): 1.019655599961615
Similarity of Local Optimum Points (percentage): 53.4387351778656%

----------------------------------------
```

```
<terminated> MainAlgorithmsRunner [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javav
First Local Optimum Algorithm:
Number of literals per clause: 3
Number of variables (n): 10
Number of clauses (m): 42
Number of CNF`s: 1
Number of Local Optimum Points: 22
Local Optimum Points Performance:
Satisfied Clauses (Average): 39.40909090909091
Satisfied Clauses (Standard Deviation): 1.1143318792846604
Similarity of Local Optimum Points (percentage): 50.12987012987013%

----------------------------------------
```

```
<terminated> MainAlgorithmsRunner [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\javaw.
First Local Optimum Algorithm:
Number of literals per clause: 3
Number of variables (n): 10
Number of clauses (m): 42
Number of CNF`s: 1
Number of Local Optimum Points: 27
Local Optimum Points Performance:
Satisfied Clauses (Average): 40.0
Satisfied Clauses (Standard Deviation): 0.9813067629253163
Similarity of Local Optimum Points (percentage): 53.903133903133906%

----------------------------------------
```

```
<terminated> MainAlgorithmsRunner [Java Application] C:\Program Files\Java\jre1.8.0_121\bir
First Local Optimum Algorithm:
Number of literals per clause: 3
Number of variables (n): 15
Number of clauses (m): 60
Number of CNF`s: 1
Number of Local Optimum Points: 7
Local Optimum Points Performance:
Satisfied Clauses (Average): 57.57142857142857
Satisfied Clauses (Standard Deviation): 0.9035079029052513
Similarity of Local Optimum Points (percentage): 52.38095238095239%

---------------------------------------
```

```
<terminated> MainAlgorithmsRunner [Java Application] C:\Program Files\Java\jre1.8.0_121\bin\
First Local Optimum Algorithm:
Number of literals per clause: 3
Number of variables (n): 15
Number of clauses (m): 60
Number of CNF`s: 1
Number of Local Optimum Points: 8
Local Optimum Points Performance:
Satisfied Clauses (Average): 57.75
Satisfied Clauses (Standard Deviation): 0.82915619758885
Similarity of Local Optimum Points (percentage): 55.23809523809524%

---------------------------------------
```

2) <u>אלגוריתם ב':</u> נמצא נקודות אופטימום מקומי ע"י יצירת נקודות התחלתיות, ומעבר על כל המשתנים באופן רנדומלי, כך שנשנה כל משתנה שיתן עבורנו שיפור בכמות הפסוקיות שמסתפקות, עד שנגיע לאופטימום.

<u>תוצאות הרצה:</u>

```
<terminated> MainAlgorithmsRunner [Java Application] C:\Program Files\Java\jre1.8.0_121\bi
Second Local Optimum Algorithm:
Number of literals per clause: 3
Number of variables (n): 10
Number of clauses (m): 42
Number of CNF`s: 1
Number of Local Optimum Points: 35
Local Optimum Points Performance:
Satisfied Clauses (Average): 41.22857142857143
Satisfied Clauses (Standard Deviation): 0.9880923694737477
Similarity of Local Optimum Points (percentage): 56.80672268907563%

---------------------------------------
```

```
<terminated> MainAlgorithmsRunner [Java Application] C:\Program Files\Java\jre1.8.0_121\bi
Second Local Optimum Algorithm:
Number of literals per clause: 3
Number of variables (n): 10
Number of clauses (m): 42
Number of CNF`s: 1
Number of Local Optimum Points: 58
Local Optimum Points Performance:
Satisfied Clauses (Average): 40.94827586206897
Satisfied Clauses (Standard Deviation): 0.5389137795892341
Similarity of Local Optimum Points (percentage): 55.90441621294616%

----------------------------------------
```

```
<terminated> MainAlgorithmsRunner [Java Application] C:\Program Files\Java\jre1.8.0_121\b
Second Local Optimum Algorithm:
Number of literals per clause: 3
Number of variables (n): 15
Number of clauses (m): 60
Number of CNF`s: 1
Number of Local Optimum Points: 188
Local Optimum Points Performance:
Satisfied Clauses (Average): 57.75
Satisfied Clauses (Standard Deviation): 1.0848325687071068
Similarity of Local Optimum Points (percentage): 52.72158379792923%

----------------------------------------
```

```
<terminated> MainAlgorithmsRunner [Java Application] C:\Program Files\Java\jre1.8.0_121\bi
Second Local Optimum Algorithm:
Number of literals per clause: 3
Number of variables (n): 15
Number of clauses (m): 60
Number of CNF`s: 1
Number of Local Optimum Points: 239
Local Optimum Points Performance:
Satisfied Clauses (Average): 58.25523012552301
Satisfied Clauses (Standard Deviation): 0.7746802886047276
Similarity of Local Optimum Points (percentage): 56.798049763838584%

----------------------------------------
```