מסמך איפון

רקע:

בעיית ליקוט מוצרים הנה בעיה של איסוף מוצרים לשילוח, כאשר המטרה הינה מזעור סה״כ זמן ליקוט של מספר הזמנות n, אנחנו מניחים שיש פונקציות שמתארת את סה״כ זמן הליקוט אשר תהיה תלויה בכמות המוצרים ללקט ובכמות העצירות שהעובד יצטרך לעשות. ברור להניח שפונקציה זו הינה חיובית עולה.

מתברר לפי (כתבה 1, עמוד 3, וכתבה 3 עמוד 1) שזו בעיה שכרגע נחשבת בעיית np,

עכשיו לבעיה זו יש כמה תתי בעיות קיימים, הבעיות שאנחנו ננסה לפתור יתחשבו בכמה מצבים.

מלקטים בהזמנה: המוצרים של כל הזמנה חייבים להיות תחת אותו מלקט (זה אומר שלא יכול שאדם 1 ילקט חלק מהזמנה x ואדם 2 ילקט חלק אחר מאותה הזמנה x) , דרישה זו הינה סטנדרטית בכל המאמרים, ובבסיסי הנתונים אשר נמצאים להשוואה.

המיקומים של כל המוצרים ידועים לפני.

כיום לבעיה זו היו כמה דרכים אשר מנסים למצוא פתרון ״טוב״ (הכוונה לפתרון אשר קרוב לפתרון למקרה הטוב ביותר), הדרכים הנפוצות שנמצאו הינם:

אלגוריתמים כגון

k-means

metropolis

seed algorithm

וגם אלגוריתמים מותאמים לחברות ספציפיות (מאמר 2, עמוד 4 CCA

תכנות לינארי

תכנות דינאמי .

עכשיו לכל אחד מהמקרים האלה היו חסרונות, (כמובן כי אף אחד לא הגיע לפתרון האידאלי).

המקורות שלנו לבדיקות הולכים להיות כמה מסדי נתונים אשר נותנים מידע על בעיות

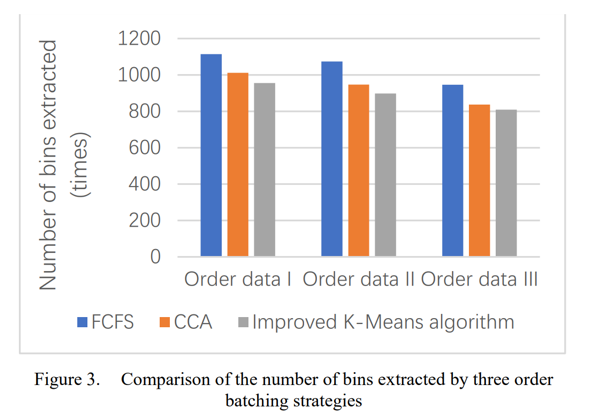
OBP = Order batching problem

בסיס נתונים זה הוזכר בכתבה 3 עמוד 6,

<https://github.com/johanoxenstierna/OBP_instances>

בסיס נתונים זה יאפשר לנו לבדוק את הפתרונות שלנו לעומת הפתרונות האידאלים.

במחקר 2, טוענים אשר גרסה משופרת של k-means מביא לתוצאה הטובה ביותר מבין האופציות הבאות, (כתבה 2, עמוד 5)



יש לציין שכיום החברה שאנחנו בקשר איתה משתמש בגרסה משופרת של k-means.

שזה אומר שבאופן יחסי הם בזמן שהוא אמור להיות לא גבוהה ביחס לכמות האופטימיזציה שמבצעים,

**תהליך הליקוט:**

נסביר את התהליך, תהליך הליקוט הינו תהליך שמגיע בכל מיני דרכים, תלוי מאוד באופי העסק.

בכול העסקים כאשר מדברים על ליקוט מדובר על לקיחת מוצר או מוצרים מהמחסן או מחסנים על מנת לשלוח אותם בדרך כזו או אחרת ללקוח.

עכשיו בכל עסק תהליך הליקוט עובד בצורה קצת שונה, אם העסק עוסק בהזמנות בזמן אמת, האם הוא אוגר הזמנות, זמן אספקה, ימי אספקה וכו.

(דורש להסביר שהבדל זה מביא לקושי בבחינה של ההבדלים בין הביצועים של האלגוריתמים)

בחברה שלנו

המקרה שבה אנחנו מסתכלים הוא על הזמנות, אשר אינם פועלות בזמן אמת, אלא נאגרות ברמת היום (שזה אומר שכל יום עובדים על ההזמנות של היום שלפניו).

המקרה שלנו הוא ליקוט על ידי בני אדם, המחסן בנוי בצורת ״ח״ , ללא מעברים (מסדרונות), במהלך המחסן.

**הפתרון של היום**

כיום הפתרון שהם מבצעים הינו גרסה רקורסיבית שאינה יעילה של אלגוריתם k-means, נסביר

האלגוריתם עובד לפי הצורה הבאה

Function create\_batches (ORDERS) {

If (center.len > 25) {

centers.replace(…create\_batches(ORDERS))

}

}

מה שהאלגוריתם עושה הוא לוקח את ההזמנות מבצע k'means ואז בעצם בודק אם יש לו חריגה מהכמות המקסימלית של המוצרים הוא לכל מרכז כזה הוא מריץ שוב את k 'means עד אשר ההזמנות בכל batch קטן מהכמות המקסימלית.

עכשיו באלגוריתם זה יש כמה בעיות,

1. פיצול יתר – כאשר מפעילים את האלגוריתם יכול להיות שהאלגוריתם יפצל הזמנות ליותר מידי מלקטים במקום לאחד אותם. דוגמא: 50 הזמנות יכולות להגיע במקום לשני מלקטים, ל 3 מלקטים כאשר המלקט הראשון 20 מלקט השני 10 ומלקט האחרון 20 הזמנות, עכשיו לדרך ליקוט זו יש כמה חסרונות, זמן set up, שאינו בא לידי ביטוי באלגוריתם. קושי בשינוים קטנים.
2. קושי בהוספת משתנים – במקרה זה קשה מאוד להתחבר לעוד פרמטר מלבד מרחק, הסיבה לכך היא שהגרסה הנוכחית של האלגוריתם יודעת לעבוד רק עם מרחקים ובעצם לנסות למזער אותה.
3. רחוק מהפתרון האופטימלי, וקושי לדעת כמה אנחנו רחוקים באחוזים מהפתרון האופטימלי.

על הארגון:

הארגון הינו חברת ייעוץ של תהליכים רבים בניהם תהליכי ייצור, אחד מהלקוחות שלהם זה החברה שאיתה אנחנו מנסים לעזור.

החלק בחברה שבה אנחנו מנסים לעזור זה בתהליך הייצוא של הבגדים לפי ההזמנה. הכוונה לכך שנכנסות הזמנות לבגדים, והמחסן צריך ללקט את המוצרים ושלוח אותם. במחסן יש שני בעיות בעיה ראשונה הינה מיקום המוצרים בהתחלה במחסן, ובעיית חלקות ההזמנות לחלקים.

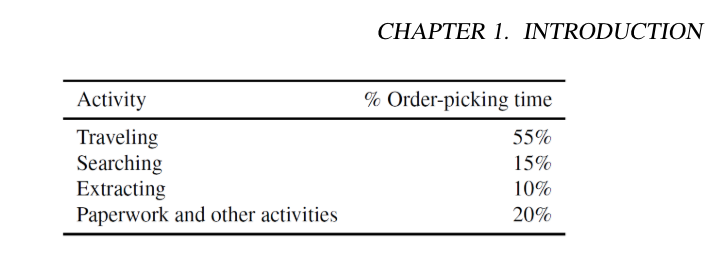
**מיקום המוצרים במחסן:** מטרתו הינו לחסוך את הפיזור בין המוצרים, וכך להשים אחד ליד השני מוצרים שיכולים ללוקט ביחד.

**בעיית חלוקת ההזמנות:** המטרה שלנו פה היא בהינתן מיקום המוצרים במחסן, לנסות לחלק את ההזמנות כך שנוכל עם n מלקטים ללקט סה״כ מקסימום מוצרים (והזמנות), ואו למזער את הזמן לביצוע של הליקוט הכולל.

במחקר הראשון מצוין ש”עומס העבודה שלה מהווה כ-60% מעומס העבודה הכולל של מרכז ההפצה, וזמן הפעולה שלה מהווה יותר מ-40% מזמן הפעולה הכולל של מרכז ההפצה.”

(תרגום שלי) מאמר ראשון.

מאמר 4, מראה ממש טבלה של הזמן ליקוט (עמוד 18), והוא גם מציין את אותו נתון על העומס עבודה שמאווה כ 60 אחוז (עמוד 17).



לכן הבעיה הכפולה הזאת היא מאוד ״מעניינת״ כי בסוף היא משפיעה על הרבה כסף.

**הבעיה שלנו:**

אז הבעיה שבה אנחנו עוסקים היא רק בחלק השני בהינתן סידור מוצרים איך לחלק את המוצרים על מנת לחסוך זמן הליכה , ולהגדיל את כמות הליקוטים.

**מדדים קיימים:**

@שוהם צריך להוסיף פה

**סקר ספרות:**

מאמר ראשון: מאמר מ 2012,

https://www.sciencedirect.com/science/article/abs/pii/S0305054812000020

עוסק בתחום הליקוט בשלושה אלגוריתמים בתחום הליקוט,

FCFS - first come first serve , Saving Algorithm (C&W(ii)) , ILS

לצערנו האלגוריתמים האלו אינם מתאימים לנו מכיוון שהם מותאמים למקרים בהם ההזמנות נכנסות בזמן אמת, וזמן האספקה הוא שונה בין ההזמנות.

כתבה אחת שנתנה לנו המון אינפורמציה בנוגע לאיך לגשת לבעיה, ופתרונות אפשריים מתעסקת בגרסה מורחבת של הבעיה שאינה מתאימה לנו, הזמנות בזמן אמת ומידע מצומצם.

מאמר 2 –

<https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9421986>

<https://ieeexplore.ieee.org/abstract/document/9421986?casa_token=mqKfSUdrgv4AAAAA:I4wneXke2XH4fqr-SrdT-W8gjAD9im_gexWRB2JpVRso4smbiHe4YBO_LtJLh5ul8ofBxug>

מאמר זה מדבר על הנושא ליקוט מוצרים, כאשר פונקציית המטרה הינה מזעור כמות הליקוטים אשר דרושים, (הכוונה זה כמות הפעמים שמלקט צריך ללכת ללקט).

האלגוריתם העיקרי שהם משתמשים בה הינו אלגוריתם k-means, עם התאמה לליקוט מוצרים במחסן.

הם משווים בין אלגוריתם שלהם CCA (אפרט למטה), FCFS, ו k-means עם שילוב של אלגוריתם גנטי, שבמקרים האלו הביאו את התוצאה הכי טובה להם.

מציין גם את החיסכון שלנו, של k-means הכי טוב, ואז האלגוריתם של החברה ,ואז בתוצאה הגרועה ביותר הינו FCFS,

CCA – הינו אלגוריתם אשר מבוסס על מידע העבר (בעיות data driven),

שלב ראשון – מטפלים בכל ההזמנות אשר כוללים רק פריט אחד.

שלב שני – כל ההזמנות אשר יש להם את אותה כמו של מוצרים

שלב שלישי – כל שאר ההזמנות מסווגות בצורה היררכית ומטופלות בסדר אשר יוצא בשלב זה .

המשופר, שזה בעצם תהליך של כמה שלבים על מנת לחבר את האלגוריתם של k-means

כך שיתאים למידע שלנו.k-means

שלב ראשון:

קביעת כמות ההזמנות (אצלנו נתון)

קביעת גודל ה batch, (אצלנו קבוע נתון מראש) [k]

שלב שני:

בוחרים k הזמנות בצורה רנדומלית, והם עכשיו המרכזים שלנו.

שלב שלישי:

לכל נקודה נחשב את המרחק שלה מאותה מרכז. ונחשב את המרכז מחדש לאותה ממוצע.

שלב רביעי:

תחבר כל הזמנה, להזמנה עם המרחק הקטן ביותר שלה,

אם לאחר פעולה זו קיים batch שבה כמות ההזמנות לליקוט הינה גדולה יותר, מהכמות המקסימלית אשר מותר לנו אזי נחבר את המרכז הקרוב לאותו מרכז, וננסה לחשב מחדש.

שלב 5:

תבדוק האם יש הזמנות לא מטופלות, או בגלל ששלב 4 לא הצליח ואז נעשה אותו שוב,

או כי הזמנות קטנות מידי.

אם לא נמשיך

שלב 6:

נחשב את המרכזים החדשים,

שלב 7, נבצע שוב את שלב 2 עד 5 , כל עוד השיפור שלנו קטן מאפסילון מסיום או חצה זה מסוים.

מאמר 3 –

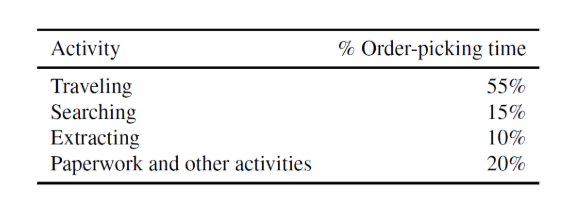
המאמר השלישי מדבר על זמני ריצה של שני אלגוריתמים

שני הגישות מתעסקות באיך לשפר את הזמני ריצה של אלגוריתמים קיימים.

מאמר – 4

https://arxiv.org/pdf/2002.00167

מאמר מאוד ארוך זה עוסק, בהמון חלקים שקשורים בליקוט, בניהם הוא מראה נקודה מעניינת על החלקות זמן שבעצם לוקחת הליקוט עצמו



עמוד 18 ,

שזה מראה לנו שהשני חלקים המעניינים ביותר הם זמן הליכה ממיקום 1 ל 2,

ולאחר מכן החיפוש בתוך המחסן.

עכשיו אנחנו נרצה לנסות ולשפר את הזמן אשר מתבזבז על פעולות אלה, שזה אומר שאם נצליח לשפר זמן ההליכה ל 45 אחוז ונעלה את הזמן שבה מבצעים הוצאות (בפועל נלקט יותר מוצרים) אז הצלחנו.

מה שמאוד מעניין על המאמר הזה הוא שהוא רושם על כל אלגוריתם את היתרונות והחסרונות שלו.

בנוגע לשיטות יוריסטיות, הוא מתאר שסה״כ אפשר להגיע לביצועים שהם קרובים לאופטימליות ובאמת בה לידי שימוש כאשר קירוב לפתרון האופטימלי הוא מספיק טוב.

מציינים שם גם שבדרך כלל יש דרישה להתאים את האלגוריתם למקרה שלך ספציפית.

יש גישה שאנחנו ננסה להתאים אלינו של

Ratliff and Rosenthal

אשר מתארים דרך פולינמלית למזער את המסלול כאשר יבחר על ידי בעיית מוכר העיתונים, הכמות על ידי תכנות לינארי, יש לציין שהאלוגיריתם בתכנות לינארי אינו ייפתר בזמן מהיר כאשר יש כמה אגפים.

האלגוריתם זה

Graphical TSP (GTSP)

על מנת לחבר את הבעיה הזאת עם בעיית האיחוד הזמנות אנחנו צריכים להסתכל על החלק במאמר שמדבר על

Joint Order Batching and Picker Routing Problem (JOBPRP)

שיטה זו מנסה למזער את הכמות הליכה כאשר אנחנו רוצים למזער את הסה״כ הליכה שיש ללכת, כאשר אנחנו בוחרים את איחוד ההזמנות בצורה מסוימת.

האלגוריתם נותן לנו גם את הפתרון של הדרך הליכה עצמה.

היתרון של האלגוריתם שמשתמש בסופו של דבר בתכנות לינארי בשלמים הינו שהוא נותן פתרון אופטימלי.

החיסרון – שכאשר הריצו בתזה את האלגוריתם זמן הריצה היה 720 שעות, שכמובן אצלנו זה זמן שהוא גבוהה בשבילנו, כמובן שאנחנו נצטרך לבדוק עם ההתאמה אלינו תהיה עם אותו יחס זמן.

**כיוונים ומתודולוגיה לפתרון הבעיה :**

כיום יש לנו שני כיוונים לפתרון הבעיה, שני הדרכים באים מהתזה שקראנו (מאמר 4),

כיוון ראשון:

על ידי שיטה ששמה הינה seed huristic שמטרתה הינה לספק קירוב טוב לפתרון בסה״כ זמן טוב.

כיון שני:

על ידי תכנות לינארי ולראות אם נוכל להתאים את האלגוריתם כך שהזמן ריצה יהיה נמוך יותר ממה שמוצג במאמר הרביעי.

**כיוונים ומתודולוגיה מוצעים לפתרון הבעיה**

אלגוריתם 1 – הינו מותאם יותר למקרה שלנו, שהוא בעצם של מה שלמדנו במאמרים האחרים.

הוא אלגוריתם יוריסטי (שאנחנו כמובן ננסה לשפר) , הוא seed heurisic , שזה בעצם אומר שאנחנו מתחילים עם הזמנה אחת ואנחנו מנסים להתאים לה את ההזמנות שהכי דומות לה.

כתבנו קוד לדוגמא של איך הוא אמור להיות, כמובן שעדיין לא סיימנו.



עדיין לא קוד סופי, אבל נסביר אותו

המטרה שלנו אנחנו מתחילים עם כל ההמנות שלנו, עכשיו לכל הזמנה יש את המידע הבא,

מספר הזמנה

מוצרים

כאשר לכל מוצר יש את המידע הבא

כמות (שהיא במקרה שלנו לא מעניינת אותנו יותר מידי)

מיקום שהיא מאוד מעניינת,

עכשיו הגדרנו את המקסימום הזמנות ב batch כ 25, כמקרה פרטי כמובן אפשר לשנות

עכשיו אנחנו מתקדמים על ידי בחירת הזמנה רנדומלית והוספה שלה אל ה batch, לאחר מכן אנחנו מתחילים לחפש את ההזמנה הכי טובה להתאים אותה אליה, שזה בעצם אומר מי ההזמנה הכי דומה אליה, שיש לה הכי הרבה התאמה והכי פחות חוסר התאמה.

(קבלנו מידע מהחברה שאצלם העצירות לוקחות הרבה מאוד זמן ולכן הדגשנו את הצורך במינימום עצירות),

לאחר מכן אנחנו מוסיפים את זה למערך מיקומים שלנו וממשיכים בכך כל עוד אנחנו עם פחות אברים מהכמות המקסימלית של ה batch.

מדדים להצלחה:

??

תוכנית עבודה:

??

Article 1

<https://www.sciencedirect.com/science/article/pii/S0305054812000020>

article 2

\

<https://scholar.sun.ac.za/server/api/core/bitstreams/4fe4f838-1956-4829-9241-3686df6476d7/content>