הטכניון - מכון טכנולוגי לישראל

הפקולטה להנדסת חשמל ע"ש אנדרו וארנה ויטרבי

המעבדה לבקרה, לרובוטיקה וללמידה חישובית

**ספר פרויקט**

**אלגוריתם DQN**

**לבקרת רמזורים**

מבצעים:

נוואף סלמאן Nawaf Salman

נאהל עוידאת Nahel Awidat

מנחה:

דר.איל טייטלר Dr. Ayal Taitler

סמסטר רישום: חורף תשפ"ג

תאריך הגשה:

**תקציר**

ניהול יעיל של תנועה בצמתים עירוניים הוא מרכיב קריטי ביוזמות עיר חכמה מודרניות. פרויקט זה מתמקד באופטימיזציה של מדיניות הרמזורים בצומת יחידה לשיפור זרימת התנועה והפחתת עומסים.  
באמצעות סביבת הסימולציה SUMO (Simulation of Urban MObility) ושפת התכנות Python, יישמנו אלגוריתם DQN כדי ללמוד ולפתח מדיניות בקרה אופטימלית לרמזורים. באמצעות מודול TRACI (Traffic Control Interface), סוכן ה-DQN מתקשר עם סימולציית ה-SUMO, אוסף נתוני תנועה בזמן אמת ומתאים את תזמוני הרמזורים באופן דינמי.  
המטרה המרכזית היא למזער את זמני ההמתנה הממוצעים ואת אורכי התורים של כלי הרכב בצומת.  
תוצאותינו מציגות שיפורים משמעותיים ביעילות התנועה, ומדגימות את הפוטנציאל של טכניקות למידת חיזוק בניהול מערכות תנועה. פרויקט זה סולל את הדרך לפתרונות נרחבים בבקרת תנועה חכמה, ותורם לרשתות תחבורה עירוניות יציבות יותר.

**Abstract**

Efficient traffic management at urban intersections is a critical component of modern smart city initiatives. This project focuses on optimizing traffic light policies at a single junction to enhance traffic flow and reduce congestion.   
Utilizing the SUMO (Simulation of Urban MObility) simulation environment and Python, we implemented a Deep Q-Network (DQN) algorithm to learn and develop an optimal traffic signal control policy.   
Through the TRACI (Traffic Control Interface) module, the DQN agent interacts with the SUMO simulation, gathering real-time traffic data and adjusting signal timings dynamically. The core objective is to minimize average waiting times and vehicle queue lengths at the intersection.   
Our results demonstrate significant improvements in traffic efficiency, showcasing the potential of reinforcement learning techniques in traffic management systems. This project paves the way for scalable solutions in smart traffic control, contributing to smoother and more sustainable urban transportation networks.

**מבוא**

ניהול תנועה יעיל בצמתים עירוניים מהווה אתגר מרכזי בערים המודרניות. עם העלייה המתמדת במספר כלי הרכב בכבישים, בעיות כמו עומסי תנועה, זמני המתנה ארוכים ופליטת מזהמים הולכות ומחריפות. בעיות אלו פוגעות באיכות החיים של התושבים, מגבירות את זיהום האוויר ומובילות לבזבוז זמן ואנרגיה. התמודדות עם האתגרים הללו והפחתת עומסי התנועה הפכה למשימה דחופה במסגרת יוזמות עיר חכמה ושיפור תחבורה בת קיימא.

הבעיה המרכזית בפרויקט זה היא ניהול ואופטימיזציה של רמזורים בצומת יחיד כדי לשפר את זרימת התנועה ולהפחית את זמני ההמתנה של כלי הרכב. צמתים אלו מהווים נקודות חנק בתשתיות התחבורה, ובקרה יעילה של רמזורים יכולה להביא לשיפור משמעותי בזרימת התנועה העירונית.

פתרונות קיימים לניהול רמזורים כוללים מערכות בקרת תנועה קבועות, אשר מבוססות על מחזורים ותזמונים מוגדרים מראש. מערכות אלו אינן גמישות ואינן מסוגלות להתאים את עצמן בזמן אמת לתנאי התנועה המשתנים, דבר שמוביל ליעילות נמוכה ולבעיות עומס. בנוסף, קיימות מערכות חכמות יותר המשתמשות בחיישנים ובאלגוריתמים פשוטים כדי להתאים את תזמוני הרמזורים, אך גם אלו לעיתים קרובות אינן מצליחות להתמודד בצורה מיטבית עם השינויים הדינמיים בתנועה.

**הגדרת בעיה**

לפתח מדיניות בקרה אופטימלית לרמזורים בצומת יחיד, שתוכל להתאים את תזמוני הרמזורים בזמן אמת ולשפר את זרימת התנועה. לשם כך, השתמשנו באלגוריתם DQN ובסימולטור SUMO כדי לבנות מדיניות חכמה לבקרת רמזורים.

מטרת הרשת היא לקבל מידע עדכני על מצב התנועה באופן דינמי, ולחזות את הפאזה האופטימלית של הצומת מסט פאזות מוגדר מראש.

**פתרונות אפשריים, והפתרון הנבחר**

בפרויקט בחנו כמה פתרונות לבקרת רמזורים: A2C ,אלגורתם Max Pressure, ופתרון מבוסס למידה מחיזוקים DQN.

Max Pressure: אלגוריתם Max Pressure הוא אלגוריתם קלאסי לבקרת רמזורים והוא מתבסס על הפחתת הלחץ בצמתים. הלחץ על כל קשת בכביש מחושב על פי ההפרש בין מספר המכוניות הממתינות בכיוון הנכנס לבין מספר המכוניות הממתינות בכיוון היוצא.  
הסוכן בוחר את הפאזה שתפחית את הלחץ בצומת באופן המקסימלי.  
זה אלגוריתם פשוט ליישום, דורש עיבוד קטן יחסית. אבל החסרון שלו שהוא לא גמיש לתנאי תנועה משתנים, ופחות יעיל כאשר יש שינויים מהירים בתנועה, וכדי להיות יעיל דורש תנאי תנועה יציבים ורמת עומס נמוכה.

A2C: אלגוריתם A2C (Advantage Actor-Critic) הוא שיטה בלמידת חיזוק המורכבת משני מרכיבים עיקריים: "Actor" (שחקן) ו-"Critic" (מבקר). ה-Actor בוחר פעולות על פי מדיניות נוכחית, וה-Critic מעריך את ערך הפעולות שנבחרו. בתהליך האימון, ה-Actor בוחר פעולה, מבצע אותה, מקבל תגמול, וה-Critic מעריך את היתרון של הפעולה כדי לעדכן את המדיניות והערכת הפעולות.  
בבעיה שלנו מערכת רמזורים היא מערכת מורכבת עם הרבה מצבים ופעולות אפשריות, בגלל זה האלגוריתם A2C עלול להתקשות להתמודד עם כל המצבים והתרחישים המגוונים באופן יעיל. כמו כן, A2C דורש כוח חישובי רב וזיכרון כדי לעבד את כל המידע ולהתאים את המדיניות, וזה עשוי להיות בעייתי במיוחד בסביבות מורכבות כמו מערכות רמזורים בערים גדולות.

DQN: אלגוריתם DQN (Deep Q-Network) הוא שיטת למידה מחיזוקים שמשתמשת ברשת נוירונים עמוקה כדי ללמוד ולשפר מדיניות לפתרון בעיות קבלת החלטות. DQN מציע יתרונות משמעותיים במונחים של פשטות, יציבות הלמידה, וניהול Replay Memory. אלו גורמים שהופכים אותו לבחירה מועדפת בבעיות רבות, במיוחד כאשר יש מרחב מצב גדול ונדרשת הערכה מדויקת של ערכי Q. במערכת רמזורים, היתרונות הללו יכולים להתבטא ביכולת טובה יותר להתמודד עם מצבים מורכבים ולהביא ללמידה יעילה יותר.

**תהליך החלטה מרקובי:**

תהליך החלטה מרקובי (MDP) הוא הגישה המתמקדת בפרקטיקה מתמטית שמשמשת למידול בעיות החלטה במקרים בהם התוצאה אינה ברורה.

רכיביו המרכזיים של תהליך החלטה מרקובי הם:

מצבים (S): המערכת יכולה להיות במגוון של מצבים או תנאים שונים. המצבים הללו מאפיינים את הקונפיגורציה הנוכחית של המערכת.  
  
פעולות (A): הסוכן הנלמד יכול לבצע פעולות המשפיעות על מעבר של הסביבה ממצב אחד למצב אחר. הפעולות הן ההחלטות או התנועות שסוכן יכול לבצע.

הסתברות מעבר (P): פונקציית הסתברות המעבר מתארת את סיכוי המעבר ממצב נתון למצב אחר בהינתן פעולה מסוימת. היא מייצגת את דינמיקת המערכת.

תגמול (R): הסוכן מקבל רווח מספרי בהתאם למצב הנוכחי ולפעולה שבוצעה. המטרה היא למקסם את הרווח הנצבר לאורך הזמן.

discount factor (): פרמטר בין 0 ל-1 שמייצג את מידת החשיבות של תגמולים עתידיים. ערך גבוה (קרוב ל-1) אומר שתגובות עתידיות חשובות כמו תגמולים מיידיים, בעוד שערך נמוך (קרוב ל-0) מצביע על כך שתגובות מיידיות חשובות יותר. המטרה היא לאזן בין תגמולים בטווח הקצר והארוך.

פונקציית ערך אופטימלית (): מוגדרת ע"י התגמול הממוצע שנצבר לאורך הזמן/המסלול כאשר מתחילים ממצב s ומבצעים פעולות אופטימליות:

פונקצייה Q: דומה לפונקציה V, אבל Q מוגדרת לכל מצב s ופעולה a. ומתארת את התגמול הממוצע שנצבר לאורך הזמן/המסלול כאשר מתחילים ממצב s ועושים פעולה a וממשיכים אחר כך באופן אופטימלי:

מדיניות (): מדיניות היא אסטרטגיה או סט של כללים המנחים את הסוכן בבחירת פעולות בכל מצב. היא מגדירה את המיפוי ממצבים לפעולות. המטרה היא לחשב מדיניות אופטימלית שממקסמת את הפונקצייה Q:

דינמיקת MDP מאופיינת במאפיין המרקובי, לפיו המצב הבא תלוי רק במצב הנוכחי ובפעולה שבוצעה, ולא ברצף המצבים והפעולות שקדמו לו. המטרה ב-MDP אם כך היא למצוא מדיניות אופטימלית שממקסמת את פונקציית הערך.

פתרון של MDP כולל מציאת מדיניות אופטימלית או פונקציית ערך אופטימלית. אלגוריתמים פופולריים לפתרון של MDP כוללים תכנות דינמי, שיטות מונטה קרלו, וטכניקות למידה מחיזוקים.

**למידה מחיזוקים:**

למידה מחיזוקים Reinforcement Learning היא פרדיגמת למידת מכונה השואבת השראה מפסיכולוגיה התנהגותית, ובה סוכן לומד לקבל החלטות על ידי אינטראקציה עם סביבה וקבלת משוב בצורת פרס או עונש. הרעיון המרכזי הוא לאפשר לסוכן ללמוד התנהגויות אופטימליות דרך ניסיון וטעיה, ולא על ידי תכנות מפורש.

ב-RL, תהליך הלמידה מתקיים כאשר סוכן פועל בסביבה, נוקט בפעולות על פי המצב הנוכחי שלה ומקבל משוב בצורת תגמול. מטרתו של הסוכן היא ללמוד מדיניות הממקסמת את סכום התגמולים הצפויים לאורך זמן. הסביבה, במקביל, מגיבה לפעולות של הסוכן ועוברת למצבים חדשים. זהו בעצם יישום של תהליך החלטה מרקובי.

הסוכן חוקר פעולות שונות כדי להבין את ההשלכות שלהן, וכך משפר את אסטרטגיית קבלת ההחלטות שלו.

יכולת הסוכן ללמוד ולהתאים את עצמו למגוון רחב של מצבים ולסביבות דינמיות, הופכת את הלמידה מחיזוקים לכלי עוצמתי בסביבות מורכבות ולא ודאיות, ומדגישה את פוטנציאל השיטה ליישומים בעולם האמיתי.

**Exploration נגד Exploitation:**

אתגר מרכזי בלמידה מחיזוקים הוא חלוקה נכונה בין exploration ל- exploitation.  
Exploration: חקירת הסביבה נחוצה כדי שהסוכן יגלה פעולות אופטימליות שאולי טרם גילה.  
Exploitation: נחוץ לשם בחירת פעולות ידועות על מנת למקסם רווח מיידי.  
שיטה ידועה לבצע את החלוקה היא : בכל צעד נבחר את הפעולה שממקסמת את הפונקצייה Q (פעולה אופטימלית) בהסתברות , ונבחר פעולה אקראית אחרת בהסתברות .  
נתחיל מערך גדול ונקטין אותו בכל צעד. זה מבטיח שהמערכת תתחיל להתמקד ב- exploration והופכת עם הזמן להסתמך יותר על הפונקצייה Q הנלמדת ולהתמקד יותר ב- exploitation.

**רכיבים של DQN**

ב-DQN משתמשים ברשת נוירונים עמוקה Policy Network להערכת פונקציית Q. הרשת מקבלת כקלט את המצב הנוכחי s ומחזירה את ערכי Q לכל הפעולות האפשריות a במצב הזה.

בנוסף לרשת הראשית, יש רשת מטרה Target Network שנעשה לה שינויים קטנים במשקולות (Soft update) . לכל משקל של הרשת הראשית , מעדכנים את המשקל המתאים ברשת המטרה בצורה הבאה:  
שינויים חלקיים וקטנים במשקלות של רשת המטרה עוזרים לשמור על יציבות במהלך הלמידה. זה מקטין את הסיכון לשינויים דרסטיים במדיניות הפעולה ומפחית את התנודתיות בלמידה.  
כמו כן, נשתמש ברשת המטרה כדי לחשב את פונקציית הערך של המצב הבא. בהינתן מצב s נעביר אותו מרשת המטרה (forward pass) וניקח את הערך המקסימלי במוצא:  
נשים לב שלא משתמשים ב- Policy Network כדי לחשב את הערך של המצב הבא כדי לשמור על יציבות בלמידה.

בנוסף, נשתמש בזיכרון (Replay Memory) כדי לשמור את החוויות מהאינטראקציה עם הסביבה. נשמור בכל צעד רביעיה (מצב נוכחי, פעולה, תגמול, מצב הבא). זיכרון זה מאפשר לאלגוריתם ללמוד מהניסיון בצורה יעילה.  
  
בנוסף, נשתמש בשיטת לאיזון ביןexploration ל- exploitation. נתחיל עם ונקטין אותו בכל צעד. לפי הנוסחה (i הוא מספר הצעד):

**מושגים והנחות:**

צומת: צומת רגילה בכביש שמורכבת מכמה רמזורים.

פאזה: מתארת את המצב של הצומת, איזה רמזורים ירוקים, צהובים או אדומים באותו רגע. נתייחס לנתיבים במקום רמזורים בתיאור הפאזה (נתיבים ירוקים, צהובים או אדומים).

סבב: מעבר על כל הפאזות בסדר מסוים כאשר מבקרים בכל פאזה פעם אחת בדיוק. צריך לעבור על כל הפאזות לפני שחוזרים לפאזה שהיינו בה באותו סבב.  
  
פאזה ירוקה: פאזה שיש בה לפחות נתיב אחד ירוק.

פאזה צהובה: פאזה שצומת צריך לעבור בה כאשר עוברים מפאזה ירוקה לפאזה ירוקה אחרת. שוהים בה בדיוק 2 שניות.  
  
פאזה אדומה: פאזה שצומת צריך לעבור לה אחרי פאזה צהובה. שוהים בה בדיוק שניה אחת.

תגמול כולל: סכום התגמולים של כל הצעדים בסימולציה.

הנחה: יש לנו סט פאזות מוגדר מראש ועוברים על הפאזות באותו סדר תמיד.

**האלגוריתם הראשי**

נגדיר DQN עם כל רכיביו כמתואר למעלה.  
  
נתחיל ממבנה ה- MDP:  
  
מצבים (S): המצב הוא שרשור של הוקטורים הבאים:  
- קידוד של הפאזה הנוכחית. למשל אם יש לצומת 4 פאזות והפאזה הנוכחית היא 0 אז הוקטור יהיה [1,0,0,0]  
- כמה זמן שהינו בפאזה הנוכחית בסבב הנוכחי. מספר זה מאותחל להיות 0 כאשר מתחילים פאזה חדשה וגדל ב- 1 בכל צעד עד שעוברים לפאזה הבאה.  
- וקטור בגודל מספר הנתיבים, כך שבמקום ה- i רשום מספר הרכבים על הנתיב ה- i.  
- וקטור בגודל מספר הנתיבים, כך שבמקום ה- i רשום מספר הרכבים העוצרים על הנתיב ה- i ומחכים על הצומת.

פעולות (A): יש 2 פעולות אפשריות, לעבור לפאזה הבאה או להישאר בפאזה הנוכחית  
{לעבור 1, להישאר 0}.

מעברים (P): פונקציית המעבר היא דטרמיניסטית. לכל מצב ופעולה קיים מצב s’שעוברים אליו בהסתברות 1, כלומר אם הוחלט על פעולה היא תתבצע.

תגמול (R): תגמול שלילי על הזמן שכל המכוניות נשארות בסימולציה. Total Travel Time (TTT).  
בפרוייקט נבחן שתי פונקציות תגמול שונות:  
1- TTT.  
2- weighted TTT: ניתן משקל גבוה למכוניות "חשובות" יותר. למשל אוטובוסים. נניח שמשקל של אוטובוס בפונקצית התגמול הוא 10 ומשקל רכב רגיל הוא 1.

discount factor (): נבחר אותו להיות 0.99. חשוב לנו את התגמול הנוכחי יותר מתגמול עתידי. אבל גם התגמול העתידי חשוב מאוד בבעיה זאת כי הפעולה שבוצעה עכשיו משפיעה גם על התגמול שנקבל בעתיד. למשל יכול להיות מקרה שבו הרבה רכבים מחכים על נתיב מסוים שנהפוך אותו לירוק רק בעוד 10 מעברי פאזה. אנחנו רוצים שהתגמול שנקבל בעוד 10 מעברים ישפיע על ההחלטה שמקבלים עכשיו ויגרום לנו במקרה זה לעבור לפאזה הבאה.

פונקציית Q:

*פונקציית* V*:   
מדיניות ():*

מבנה ה- DQN: לכל רשת ב-DQN יש שלוש שכבות נסתרות עם פונקציות הפעלה ReLU ברוחב 128 נוירונים ושכבה רביעית בלי פינקציית הפעלה ברוחב 2 נוירונים.

לכל מצב בכניסה הרשת מחשבת את הערך של כל פעולה. מוצא הרשת הוא וקטור בגודל 2:  
[]

Loss & Optimizer: נשתמש באופטימייזר RMSProp ובפונקצית הפסד Huber Loss. שמאפשרים למודל להתמודד בצורה יעילה עם תנודתיות ושונות גבוהה בערכי Q, לשפר את יציבות האימון, ולהקטין את ההשפעה של נקודות קיצון. יחד, הם תורמים לשיפור הביצועים וההתכנסות של המודל בבעיות RL.

**תהליך האימון**

נבחר בכל צעד את הפעולה האופטימלית לפי שיטת .  
נפעיל את הפעולה ע"י אינטראקציה עם הסביבה, כתוצאה מכך הסביבה תעבור למצב הבא, ונקבל מהסביבה את הרביעיה (מצב נוכחי, פעולה, תגמול, מצב הבא).  
נכניס את הרביעיה ל- Replay Memory.  
בכל צעד נדגום אקראית batch בגודל קבוע מה- Replay Memory. לכל s, a ב- batch:  
- נחשב את פונקציית הערך של המצב הבא ע"י שימוש ב- Target Network כמתואר למעלה.  
- נחשב את הערך הצפוי ל- Q לפי נוסחת ה- Q-Learning:

- נחשב את הערך של Q הנלמד ע"י העברת s ברשת הראשית (forward pass) ולקיחת הערך בויקטור המוצא באינדקס a.

- נחשב את ה- loss בין לבין . נחשב את הנגזרת של ה- loss ע"י backpropagation ונשתמש ב- RMSProp כדי לעדכן את משקולות הרשת הראשית.

- נבצע עדכון Soft update למשקולות של רשת המטרה כמו שמתואר למעלה.

נריץ את כל הצעדים עד שהסימולציה מסתיימת (3600 צעדים לכל סימולציה). וזה נחשב episode אחד.  
נריץ 200 episodes כאלה כדי להבטיח שהתגמול הכולל התכנס לערך האופטימלי ולכן גם המדיניות התכנסה למדיניות האופטימלית.

**TODOs:**

**4 Hyperparameter Tuning**

**5- תוצאות + השוואות**

**6- השוואה בעומס גבוה בין max pressure ל- DQN**

**7- resources**

**8- فهرس**

**כלים**

SUMO-"Simulation of Urban Mobility", היא תוכנת סימולציה פתוחה לתחבורת עירונית המיועדת ליצירת מודלים ולדימוי תנועה בסביבות עירוניות. היא תומכת במיקרו-סימולציה של רכבים פרטיים, תרחישים מרובים-מודאליים (כולל רכבי רגל ותחבורה ציבורית), ובהדמיה של מערכות בקרת תנועה. התוכנה גמישה מאוד ומאפשרת עבודה בסדרי גודל שונים של מערכת, מה שהופך אותה לכלי יקר ערך לחקר זרימת תנועה, לבדיקת שינויי תשתיות, ולהערכת אסטרטגיות ניהול תנועה בסביבות עירוניות שונות.

TRACI-"Traffic Control Interface", היא ספרית פייתון המסופק ת ע"י SUMO. היא מאפשרת לתוכניות וכלים חיצוניים לתקשר בזמן אמת עם סימולציית, ומספקת דרך לשלוט ולעקוב אחר מספר רב של אספקטים בסימולציה. דרך TRACI משתמשים יכולים להריץ ניסויים, לבדוק אלגוריתמים ולנתח תרחישי תנועה בסביבה וירטואלית לפני יישום השינויים בעולם האמיתי.

PyTorch- ספריית למידת מכונה open source שפותחה על ידי מעבדת AI Research של פייסבוק. היא מאפשרת בנייה של מודלים של למידת מכונה, כולל רשתות נוירונים, בצורה דינמית וגמישה. נשתמש ב-PyTorch בעיקר כדי להשתמש במודלים ופונקציות פעולה בנויות כמו NN

ו- ReLU, וכדי לבצע חישובים כבדים על GPU.

TensorFlow - היא ספריית open source שפותחה על ידי google ומשמשת ללמידת מכונה, למידה עמוקה ולניתוח נתונים. הספרייה מאפשרת בניית מודלים מתקדמים ואימון רשתות נוירונים בצורה קלה ואפקטיבית. חלק מרכזי ב- TensorFlowהוא TensorBoard, הוא כלי ויזואליזציה שמסופק יחד עם TensorFlow שמאפשר להציג גרפים של המודלים, מעקב אחר מטריקות של האימון, צפייה בתמונות, היסטוגרמות, ועוד. ניתן לעקוב אחרי מטריקות כמו הפסד (loss), דיוק (accuracy), גרדיאנטים ועוד, במרוצת האימון. זה עוזר לדיבוג, להבין איך המודל משתפר (או למה לא משתפר) עם הזמן.

**הסימולציה:**

את הסימולציה של הצומת העירוני (איור 1) שמהווה סביבה לסוכן הלומד, יצרנו באמצעות SUMO .

הסימולציה בנויה מצומת כאשר כל כניסה לאזור מהווה נקודת התחלה וסיום נסיעה. בצומת יש רמזורים לכל כיווני התנועה, מכיוון הגעה לשאר הכיוונים.  
משך הסימולציה הינו 3600 שניות, המדמות שעה בעולם האמיתי.

לצומת יש רשימת פאזות מוגדרת מראש (איורים 2 ו- 3). לכל פאזה יש אינדקס ומחרוזת שמתארת את הצבעים של כל הנתיבים (למשל לפאזה הראשונה יש אינדקס 0 ומחרוזת GrrrGrrr).

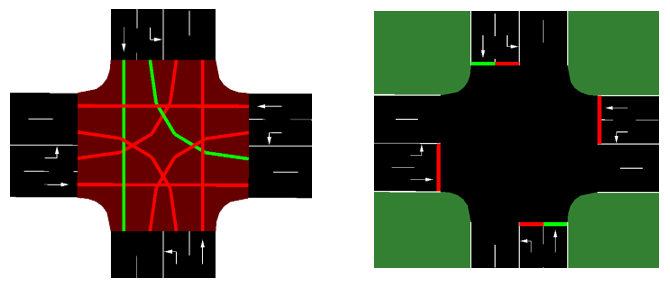
ליצירת התנועה בעיר השתמשנו ברכבים פרטיים ובאוטובוסים, כל מסלולי ועומסי הזרימה מוגדרים בקובץ מסוג rou.xml. כל שורה בקובץ זה מתארת זרימת רכבים מצומת לצומת. למשל, יש שורה מתאימה לזרימת רכבים רגילים מצומת J17 לצומת J21 שמתחילה בזמן 0 ומסתיימת בזמן 3600, שהסתברות הגעת רכבים היא 0.0299 בכל שניה.  
ועבור זרימת אוטובוסים נוסף Type באותה שורה שבעזרתו נוכל להבדיל בין רכב לאוטובוס.

(איור 4).  
  
נשתמש בספריית TRACI ב- python. נטען בה את קובץ הסימולציה מסוג sumocfg וזה טוען אוטומטית את הקובץ שמתאר את עומס הזרימה (מסוג rou.xml).  
בעזרת TRACI נוכל לבצע כל פעולה על הצומת ולקבל מידע על המצב הנוכחי שלו. בכל צעד נשתמש ב- API של TRACI כדי לבצע את הפעולה וכדי לקבל את הרביעיה (מצב נוכחי, פעולה, תגמול, מצב הבא) שתעזור לנו להמשיך את תהליך האימון.

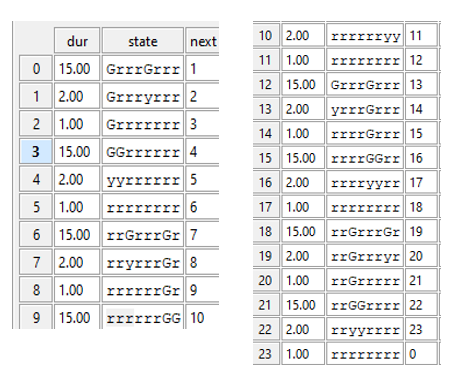
A cross-section of a road

Description automatically generated

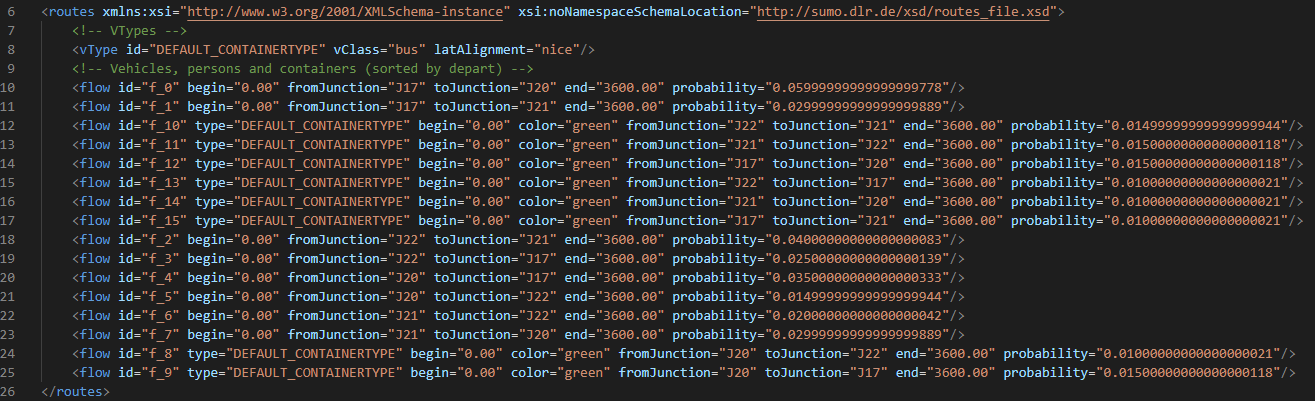
איור 1



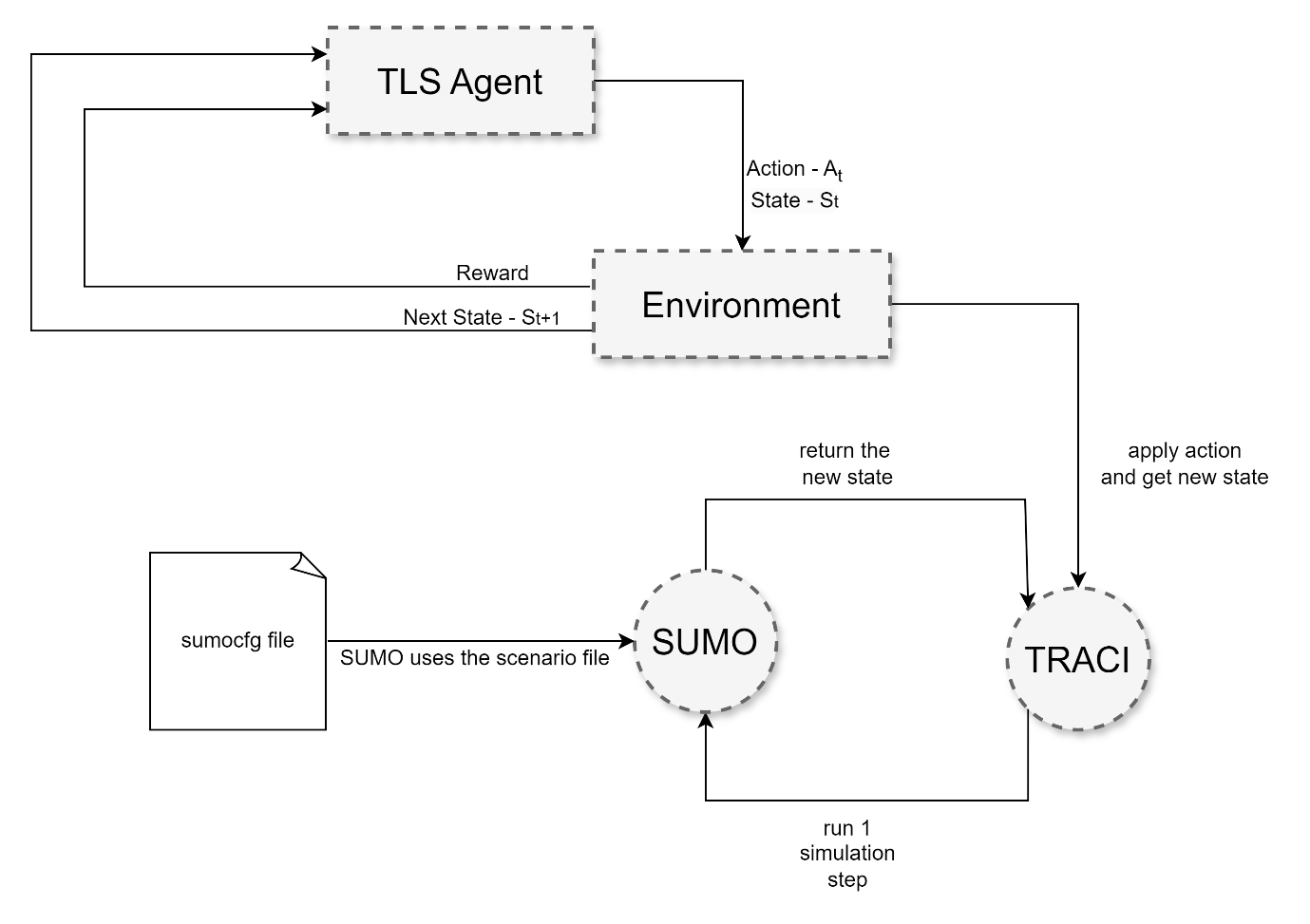
איור 2



איור 3



איור 4

**Block Diagram**

**Sequence Diagram**

