הפרויקט שלנו עוסק במאמר Learning rate branching heuristic for SAT solvers. במהלך הפרויקט מימשנו את האלגוריתמים המתוארים במאמר במסגרת של מימוש של Edusat solver בסיסי עם היוריסטיקה VSIDS ובדקנו האם ישנו שיפור בביצועים.

שי כליפה, יונתן להט

דו"ח סיום פרויקט באלגוריתמים בלוגיקה

Learning rate branching heuristic for SAT solvers by Jia Hui Liang, Vijay Ganesh, Pascal Poupart, and Krzysztof Czarnecki University of Waterloo, Waterloo, Canada

תוכן

[הצגת עקרונות ומושגים רלוונטיים להמשך: 2](#_Toc65079271)

[MAB algorithm called exponential recency weighted average (ERWA): 2](#_Toc65079272)

[Conflict side and Reason side: 2](#_Toc65079273)

[Interval: 2](#_Toc65079274)

[Learning Rate: 2](#_Toc65079275)

[Reason side rate: 3](#_Toc65079276)

[Locality: 3](#_Toc65079277)

[אלגוריתמים ומימושים שלהם: 4](#_Toc65079278)

[ERWA: 4](#_Toc65079279)

[ERWA + RSR: 5](#_Toc65079280)

[LRB 6](#_Toc65079281)

[קוד משותף לשלושת הHeuristics והסברים: 6](#_Toc65079282)

[תוצאות: 9](#_Toc65079283)

# הצגת עקרונות ומושגים רלוונטיים להמשך:

## MAB algorithm called exponential recency weighted average (ERWA):

אלגוריתמי MAB או multi armed bandit הינם אלגוריתמי בינה מלאכותית שמיועדים לקביעת הפעולה שתביא למקסימום את תוחלת הרווח של שחקן. הדוגמה הנפוצה שניתנת באלגוריתמים מן הסוג הנ"ל היא ניסיון למצוא את המכונה בעלת תוחלת הרווח הכי גבוהה בקזינו. האלגוריתם מתחלק לשני שלבים exploration ו exploitation. בשלב ה exploration האלגוריתם מנסה מכונות באקראי ומחשב לכל מכונה ממוצע במקרה שלנו של exponential recency weighted average אנו מקטינים באופן אקספוננציאלי את המשקל שניתן לתצפיות ישנות על ידי עדכון הממוצע על ידי כפל הערך הישן פי ותצפית חדשה פי *. ככל שנראה יותר דוגמאות נוכל לסמוך יותר על נכונות הממוצע שחישבנו ולכן נקטין את לאורך הריצה. בשלב הexploitation האלגוריתם מנצל את מה שלמד ובוחר במכונה שחושב לה הממוצע הכי גבוהה ומשתמש בה ומוסיף את התוצאה לממוצע כמו בשלב הexploration. במהלך הריצה מבצעים את שני השלבים לחילופין בהתחלה יותר exploration כדי ללמוד את הסביבה ואז יותר exploitation כדי להשתמש במה שנלמד לבחירת אסטרטגיה. במקרה של ההיוריסטיקה המתוארת במאמר הexploration מבוצע על ידי propagation , הexploitation מבוצע על ידי בחירת המשתנה עם ערך היוריסטיקה הגבוהה ביותר בDecide.*

## Conflict side and Reason side:

המאמר מציג חלוקה של הimplication graph לשני צדדים צד הconflict וצד הreason. הגרף מחולק לפי הlearnt clause כאשר כל מי שנמצא מימין לlearnt clause בגרף כלומר כל מה שנגרר על ידי הlearnt clause כולל הconflict נקרא conflict side וכל מה שהשתתף בהסקת הlearnt clause נקרא Reason side.

## Interval:

מוגדר במאמר כמספר הפסוקיות שנלמדו מהרגע שלמשתנה הושם ערך עד הרג שבו המשתנה איבד את ערכו.

## Learning Rate:

מוגדר במאמר להיות מספר הפעמים בהם משתנה היה חלק מה Conflict side במשך הזמן בו היה לו ערך לחלק למספר הפסוקיות שנלמדו כאשר היה לו ערך. הנ"ל משערך כמה המשתנה משמעותי בלמידת Conflicts במטרה למצוא משתנים שיכולים לייצר הרבה Conflicts ושהשמה טובה שלהם יכולה לעזור ללמוד מהר יותר.

## Reason side rate:

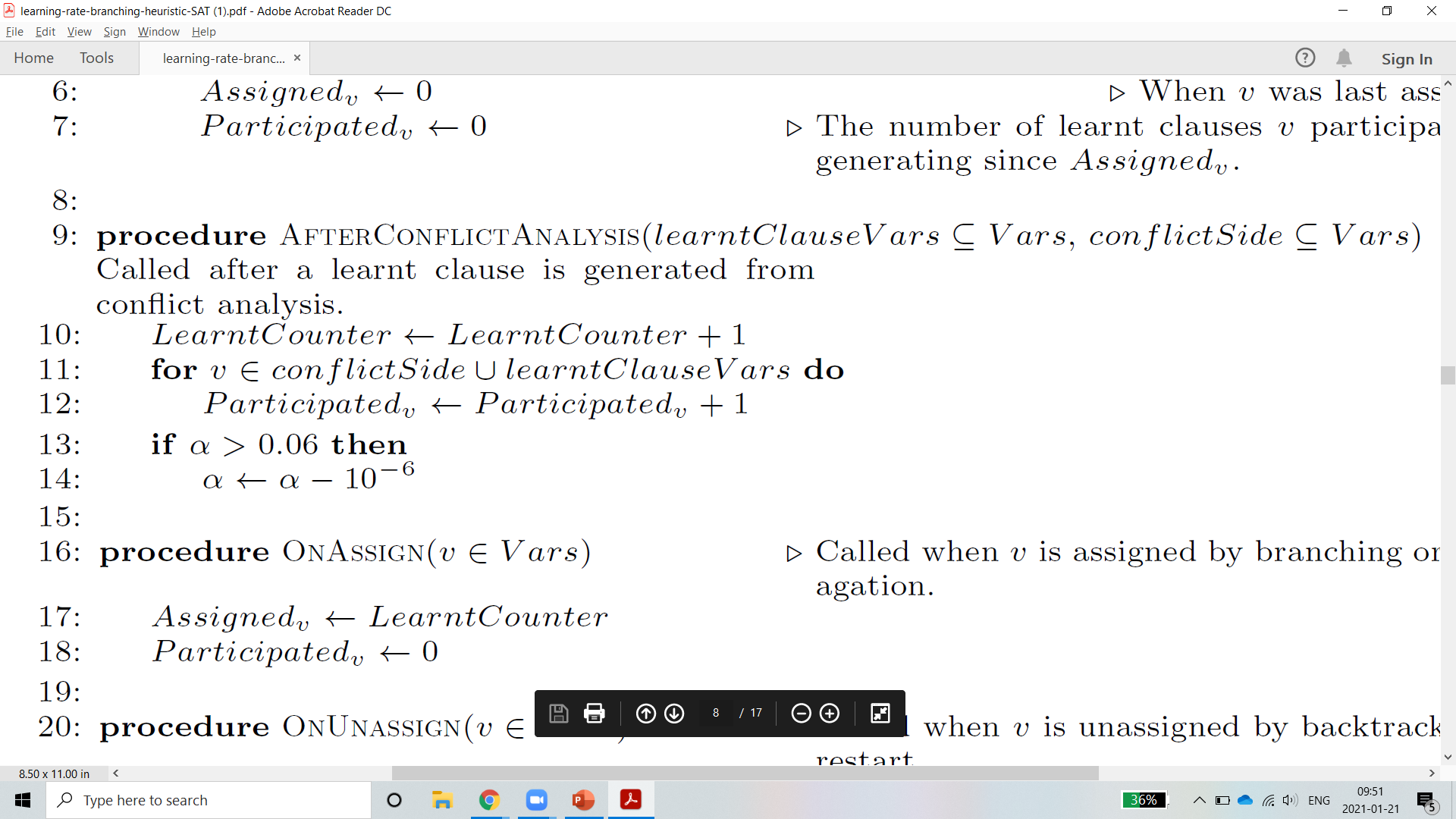
מוגדר במאמר להיות מספר הפעמים בהם משתנה לקח חלק בהסקת ה learnt clause (היה שייך ל (Reason sideבמשך הזמן בו היה לו ערך לחלק במספר הפסוקיות שנלמדו כאשר היה לו ערך. הנ"ל מגדיל את הציון של משתנים שמשתתפים בהסקת הlearnt clause כהערכה שהשמה מוקדמת וטובה שלהם יכולה לעזור ללמוד מהר יותר.

## Locality:

קיים יתרון מובהק לאלגוריתם שנותן עדיפות לחיפוש באזור שבו הוא מחפש כרגע וממצה אותו עד תום על פני אלגוריתמים שנעים וזזים מאזור אחד במרחב החיפוש למשנהו. כדי לממש את העיקרון הזה בכל conflict מכפילים את הציון של כל המשתנים שאין להם השמה בdiscount factor במאמר 0.95 וזה למעשה נותן יתרון למשתנים שאכן יש להם השמה באותו שלב.

# אלגוריתמים ומימושים שלהם:

## ERWA:

מתוך המאמר:

משתנים:

double alpha; // learning rate for ERWA

vector<double> m\_Q; // Var => Qv score

double m\_curr\_Q; // holds the max Qv score of unassigned variable

vector<int> time\_assigned; // # of learnt clauses at assignment

vector<int> participated; // # of conflicts the variable participated in

alpha מייצג את  *מהמאמר וקובע בכמה נתקן את הממוצע.*

*m\_Q מחזיק לכל משתנה את הייצוג שלו לפי היוריסטיקה.*

*m\_curr\_Q מחזיק את הציון של המשתנים שכרגע נרצה לבחור בDecide.*

*Time\_assigned מחזיק לכל משתנה כמה פסוקיות נלמדו כאשר הושם לו ערך.*

*Participated מחזיק לכל משתנה בכמה conflict sides הוא השתתף מאז שקיבל ערך*

*חלקי מימוש שמתחזקים נכונות:*

if (!marked[v]) {

++participated[v];

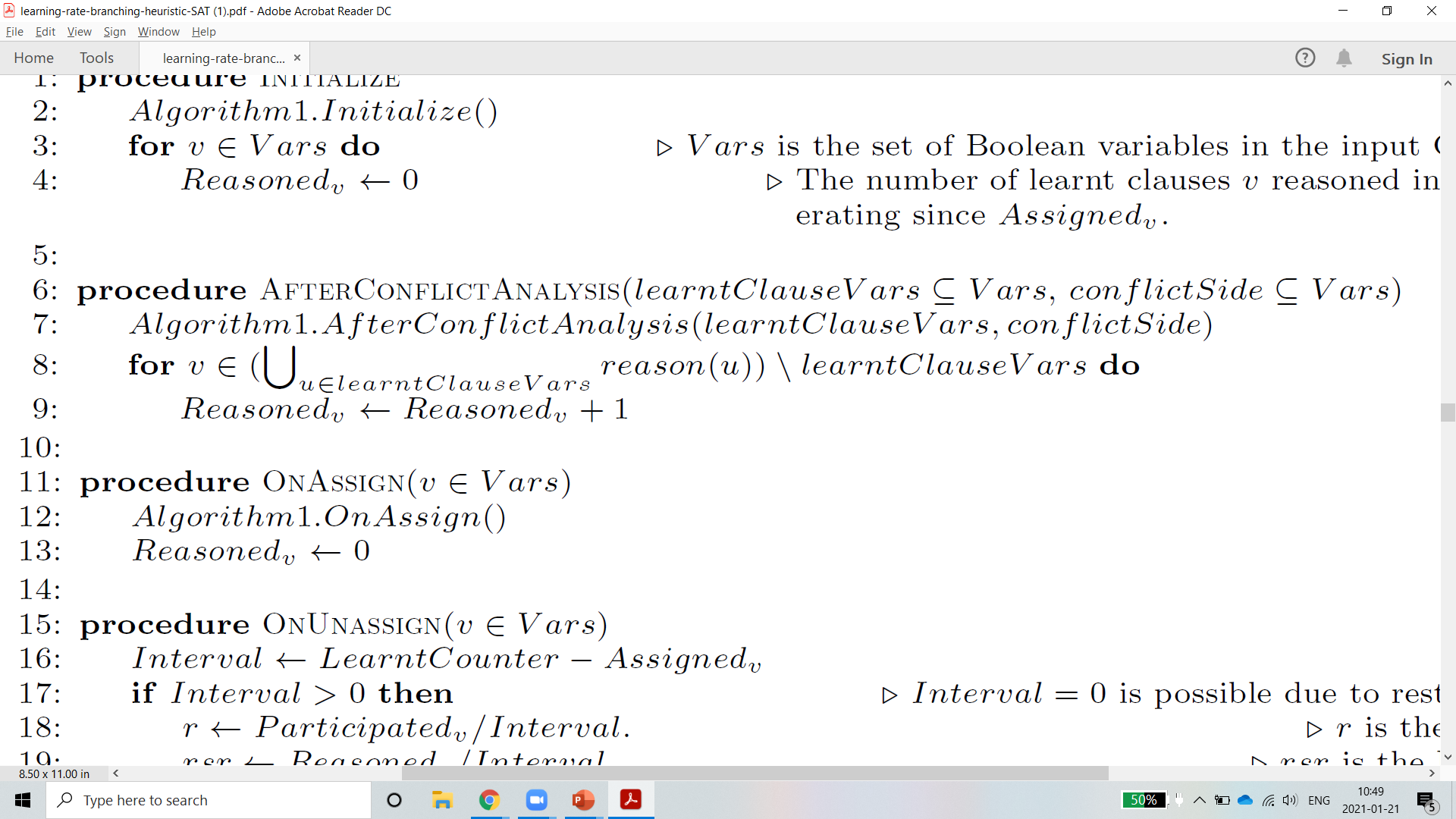
*כל המשתנים שנעבור עליהם בחישוב הlearnt clause בanalyze שייכים לconflict side וmarked דואג שלא נעבור על אותו משתנה פעמיים.*

if (VarDecHeuristic != VAR\_DEC\_HEURISTIC::MINISAT) if (alpha > 0.06) alpha -= 1e-6;

*בכל שלושת הHeuristics נרצה להקטין את ב כפי שעושים במאמר.*

*שני קטעי הקוד הנ"ל נלקחו מanalysis (הפונקציה שמנתחת conflicts) ומבוצעים תוך כדי ניתוח הקונפליקט ולא ממש אחריו כדי לחסוך זמן ריצה. שאר הפונקציות והמימושים הרלוונטיים משותפים לשלושת הHeuristics ונדון בהן בנפרד.*

## ERWA + RSR:

מתוך המאמר:

תוספת למשתני ERWA שרלוונטיים לאלגוריתם הנוכחי:

vector<int> reasoned; // # of conflicts the variable reasoned in

reasoned מחזיק לכל משתנה את מספר הlearnt clauses שהוא השתתף ביצירתן.

*חלקי מימוש שמתחזקים נכונות:*

if (VarDecHeuristic == VAR\_DEC\_HEURISTIC::ERWA\_RSR || VarDecHeuristic == VAR\_DEC\_HEURISTIC::LRB)

{

for (clause\_it it = new\_clause.cl().begin(); it != new\_clause.cl().end(); ++it)

{

Lit l = \*it;

Var va = l2v(l);

int ant = antecedent[va];

if (ant == -1) continue;

Clause ant\_clause = cnf[ant];

for (clause\_it it2 = ant\_clause.cl().begin(); it2 != ant\_clause.cl().end(); ++it2)

{

Lit l2 = \*it2;

Var va2 = l2v(l2);

if (va2 != va && !in\_learnt\_clause[va2])

reasoned[va2] += 1;

}

}

בחלק זה של הקוד אנו עוברים על הlearnt clause בודקים לכל משתנה בה האם הוא הוסק על ידי propagation אם כן אנו מוסיפים כל משתנה בפסוקית ממנה עשינו propagation לreason של הlearnt clause כפי שמתואר באלגוריתם במאמר.

גם חלק קוד זה נלקח מanalysis ומבוצע אחרי בניית הlearnt clause.

## LRB

תוספת למשתנים קודמים שרלוונטית לאלגוריתם הנוכחי:

vector<double> factors; // # factor to double Q in LRB var => factor

factors מכיל לכל משתנה פי כמה להכפיל את הציון הקודם שלו לפני תיקון הציון.

if (VarDecHeuristic == VAR\_DEC\_HEURISTIC::LRB) {

double factor = 0.95;

for (Var i = 1; i < nvars + 1; i++)

{

if (state[i] == VarState::V\_UNASSIGNED)

{

factors[i] \*= factor;

}

}

}

בחלק זה של הקוד אנו מכפילים לכל משתנה שאין לו השמה את הפקטור פי 0.95 כדי שכאשר נשנה את הציון נקטין אותו פי הפקטור הנכון.

גם קטע הקוד הזה נלקח מanalysis ומבוצע לאחר בניית הlearnt clause.

## קוד משותף לשלושת הHeuristics והסברים:

אתחול המשתנים המוזכרים לעיל:

time\_assigned.resize(nvars + 1);

participated.resize(nvars + 1);

reasoned.resize(nvars + 1);

factors.resize(nvars + 1);

m\_Q.resize(nvars + 1);

m\_curr\_Q = 0.0f;

for (int v = 1; v <= nvars; ++v) {

m\_Q[v] = 0;

time\_assigned[v] = 0;

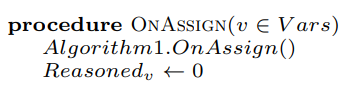
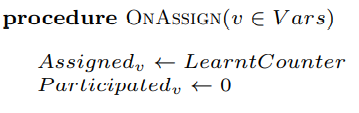
participated[v] = 0;

reasoned[v] = 0;

factors[v] = 1;

}

לקוח מתוך הפונקציה initialize.

מתוך המאמר:

ממומשים על ידי:

void Solver::reset\_after\_assigment(Var v)

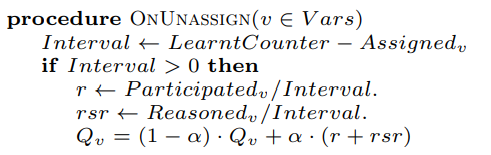
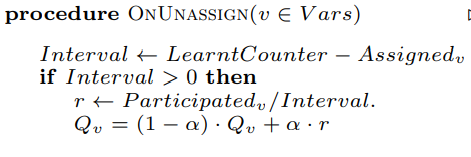
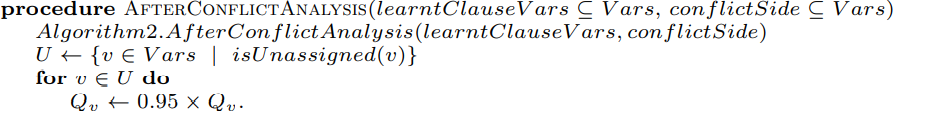
{

participated[v] = 0;

reasoned[v] = 0;

time\_assigned[v] = num\_learned;

}

מתוך המאמר:

ממומשים ביחד ב:

void Solver::bumpQScore(int var\_idx)

{

double new\_score;

double score = m\_Q[var\_idx];

if (score > 0) {

Assert(m\_Q\_Score2Vars.find(score) != m\_Q\_Score2Vars.end());

m\_Q\_Score2Vars[score].erase(var\_idx);

if (m\_Q\_Score2Vars[score].size() == 0) m\_Q\_Score2Vars.erase(score);

}

double LR;

double RSR;

double interval = num\_learned – time\_assigned[var\_idx];

if (num\_learned == 0)

{

LR = 0;

RSR = 0;

}

else if (interval != 0)

{

LR = participated[var\_idx] / interval;

RSR = reasoned[var\_idx] / interval;

}

else return;

new\_score = (1 – alpha) \* factors[var\_idx] \* score + alpha \* (LR + RSR);

m\_Q[var\_idx] = new\_score;

factors[var\_idx] = 1;

if (m\_Q\_Score2Vars.find(new\_score) != m\_Q\_Score2Vars.end())

m\_Q\_Score2Vars[new\_score].insert(var\_idx);

else

m\_Q\_Score2Vars[new\_score] = unordered\_set<int>({ var\_idx });

}

כאשר ברירת המחדל של reasoned, factors היא ניטרלית (0 ו 1 בהתאמה) כך שהפונקציה מחשבת נכון גם אם נשתמש בERWA שלא משתמש במשתנים אלה. אנו משנים את reasoned, factors מברירות המחדל שלהם רק אם משתמשים בHeuristic שצריכה אותם לחישוב. כמו כן אנו קוראים לפונקציה לאחר שהמשתנה מאבד את השמתו.

דוגמה לDecide והסברים עליו:

case VAR\_DEC\_HEURISTIC::ERWA: {

if (m\_should\_reset\_iterators) reset\_iterators\_Q(m\_curr\_Q);

Var v = 0;

int cnt = 0;

if (m\_Q\_Score2Vars\_it == m\_Q\_Score2Vars.end()) break;

while (true) { // scores from high to low

while (m\_VarsSameQ\_it != m\_Q\_Score2Vars\_it-> second.end())

{

v = \*m\_VarsSameQ\_it;

++m\_VarsSameQ\_it;

++cnt;

if (state[v] == VarState::V\_UNASSIGNED)

{ // found a var to assign

m\_curr\_Q = m\_Q\_Score2Vars\_it->first;

assert(m\_curr\_Q == m\_Q[v]);

best\_lit = getVal(v);

goto Apply\_decision;

}

}

++m\_Q\_Score2Vars\_it;

if (m\_Q\_Score2Vars\_it == m\_Q\_Score2Vars.end()) break;

m\_VarsSameQ\_it = m\_Q\_Score2Vars\_it->second.begin();

}

break;

}

כדי שנוכל לבצע השוואה apples to apples ניסינו לשמור ככל הניתן על המבני נתונים שהאלגוריתם השתמש בהם בVSIDS כך שלא יהיה לנו overhead מצד אחת הHeuristics במהלך Decide שלא קשור לערך עצמו.

לכן פונקציית הDecide זהה ביצועית לפונקציה בVSIDS רק עם מבני נתונים שמותאמים אליה.

הפונקציה עובדת על ידי תחזוקת מילון שמפתחותיו ציונים וערכיו הם קבוצת משתנים עם הציון הזה. הפונקציה עוברת על המילון עד שהיא מוצאת משתנה ללא השמה (בעל הציון הכי גבוהה) ומחזירה אותו.

Decide עובדת אותו דבר לשלושת הHeuristics הקוד שוכפל לנוחות שינויים. באם היינו צריכים לשנות לאחת מהHeuristics את הDecide למרות הרצון שלנו לא לעשות זאת רצינו שהקוד יאפשר זאת בקלות.

# תוצאות:

אלו טבלאות של התוצאות על פני הסטים השונים:

זו הטבלה עבור היורסטיקה Phasesaving :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Average Time | Finished | Num of conflicts |
| ERWA | Easy | 0.03725 | 72 | 23.13888888888889 |
|  | Sat | 166.14992857142857 | 13 | 3396.0 |
|  | Unsat | 62.18919999999999 | 5 | 488.6 |
|  | 2002 – beta | 94.63805276381903 | 375 | 4996.095477386934 |
| RSR | Easy | 0.036847222222222226 | 72 | 23.75 |
|  | Sat | 192.4457142857143 | 13 | 3367.5714285714284 |
|  | Unsat | 33.4746 | 5 | 334.2 |
|  | 2002 – beta | 93.2177763819095 | 385 | 6040.228643216081 |
| LRB | Easy | 0.040638888888888884 | 72 | 25.055555555555557 |
|  | Sat | 187.1147142857143 | 13 | 3310.714285714286 |
|  | Unsat | 38.765600000000006 | 5 | 333.6 |
|  | 2002 – beta | 108.90097236180914 | 378 | 6591.459798994975 |
| VSIDS | Easy | 0.07333333333333335 | 72 | 45.5 |
|  | Sat | 202.09349999999998 | 14 | 5318.285714285715 |
|  | Unsat | 34.365 | 5 | 279.8 |
|  | 2002 – beta | 92.54488944723622 | 378 | 6843.934673366834 |

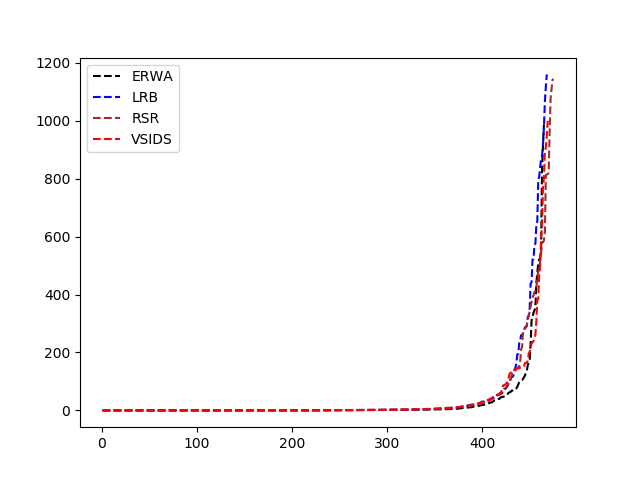
זו הטבלה עבור היורסטיקה Litscore :

|  |  |  |  |
| --- | --- | --- | --- |
|  | Average Time | Finished | Num of conflicts |
| ERWA | 130.68304907975448 | 450 | 6188.668711656442 |
| RSR | 126.80247852760735 | 458 | 5917.067484662577 |
| LRB | 110.8268425357873 | 464 | 5057.231083844581 |
| VSIDS | 116.50357055214722 | 455 | 5812.963190184049 |

עבור היוריסטיקה Litscore הרצנו את כל הסטים ביחד ולכן אין הפרדה בטבלה של התוצאות.

אלו הגרפים שמשווים בין היוריסטיקות השונות:

זה הגרף של Phasesaving :



זה הגרף של Litscore :

