Софийски университет "Св. Климент Охридски"
Факултет по математика и информатика

Компресирано представяне на бимашина

Дипломна работа на

Иван Димитров Велевски
Факултетен номер: M24467
Специалност "Компютърна лингвистика"
Катедра "Математическа логика и приложенията ѝ"

Научен ръководител: доц. Стоян Михов

- Март 2020г. -

Съдържание

1 Въведение

Често срещана в сферата на компютърната лингвистика е задачата за преписване на текст. Тя се свежда до трансформирането на даден входен текст в изходен такъв, като за целта се прилагат предварително определени правила за заместване. Правилата за заместване могат да бъдат представени по най-прост начин като речник от двойки думи, но в по-изразителната си форма могат да се дефинират като регулярни функции. Като примери за преписване на текст могат да бъдат посочени проблемите за автоматично анотиране на текст с хиперлинкове, добавяне на тагове в текст, привеждане на думи от текст в нормална форма (stemming) и много други.

За решаване на задачи от такъв вид в практиката често се използват машини на крайните състояния. В случай когато правилата на заместване са във формата на речник, може да бъде конструиран подпоследователен преобразувател, реализиращ заместването (например конструкцията от статия [1]). Когато правилата са представени като регулярна функция, може да бъде конструиран съответен двулентов автомат, а от него и бимашина, която да извършва заместването. За формални дефиниции и приложения на гореспоменатите машини може да посочим учебника [2].

Именно бимашината представлява интерес за текущата работа. Сред нейните предимства са сравнително простата ѝ абстракция, детерминистичното ѝ поведение и линейното време за обработване и превеждане на входния текст. Като основен недостатък може да се посочи размерът на заеманата памет при нейното представяне. Най-често паметта е значително по-голяма от паметта за еквивалентния двулентов автомат.

В следващите секции ще разгледаме подход за компресирано представяне на бимашина. При компактната версия ще се стремим да запазим свойството за линейно обработване на входния текст.

2 Дефиниция за бимашина

Нека първо формално да представим дефиницията на бимашина. Бимашина B е наредена тройка $B =< A_L, A_R, \Psi >$, за която е изпълнено:

- $A_L =< \Sigma, Q_L, s_L, Q_L, \delta_L >$ и $A_R =< \Sigma, Q_R, s_R, Q_R, \delta_R >$ са крайни детерминирани автомати, наречени съответно ляв и десен автомат на бимашината.
- $\Psi \colon (Q_L \times \Sigma \times Q_R) \mapsto \Sigma^*$ е частична функция, наречена функция на изходите.

Нека $B =< A_L, A_R, \Psi >$ е бимашина, Σ е азбуката на левия и десния автомат и $w = \alpha_1\alpha_2 \dots \alpha_n$ е дума от $n \geq 0$ символа, като $\alpha_i \in \Sigma$ за $i \in \{1, \dots, n\}$. Когато са дефинирани всички преходи $\delta_L^*(s_L, \alpha_1\alpha_2 \dots \alpha_i)$ и $\delta_R^*(s_R, \alpha_n\alpha_{n-1} \dots \alpha_i)$ (за $1 \leq i \leq n$), получаваме два успешни пътя в A_L и A_R. Да ги наречем съответно π_L и π_R.

$$\pi_L : l_0 \rightarrow^{\alpha_1} l_1 \rightarrow^{\alpha_2} l_2 \dots l_{n-1} \rightarrow^{\alpha_n} l_n$$
$$\pi_R : r_0 \leftarrow^{\alpha_1} r_1 \leftarrow^{\alpha_2} r_2 \dots r_{n-1} \leftarrow^{\alpha_n} r_n$$

При текущите означения $s_L = l_0$ и $s_R = r_n$. Ако функцията на изходите $\Psi(l_{i-1}, \alpha_i, r_i)$ е дефинирана за $1 \leq i \leq n$, думата

$$\mathbb{O}_B(w) = \Psi(l_0, \alpha_1, r_1) \cdot \Psi(l_1, \alpha_2, r_2) \cdot \dots \cdot \Psi(l_{i-1}, \alpha_i, r_i) \cdot \dots \cdot \Psi(l_{n-1}, \alpha_n, r_n)$$

наричаме образ или изход на w в бимашината B. В случай когато $w = \varepsilon$, дефинираме $\mathbb{O}_B(w) = \varepsilon$. Ако в A_L и A_R не съществуват съответни пътища π_L и π_r или $\Psi(l_{i-1}, \alpha_i, r_i)$ не е дефинирана за някое $i \in \{1 \dots n\}$, стойността на $\mathbb{O}_B(w)$ също е недефинирана. Частичната функция $\mathbb{O}_B$ наричаме функцията представена от бимашината B.

От дефиницията се вижда, че пресмятането на $\mathbb{O}_B(w)$ отнема линейно време. Първо използваме обратната на w дума - $\alpha_n \dots \alpha_2\alpha_1$ за да траверсираме десния автомат A_R на бимашината, като запаметяваме получените състояния, формиращи пътя π_R. След това траверсираме A_L, използвайки w, и получаваме π_L. Накрая, ползвайки състоянията в двата пътя и функцията на изходите Ψ, сглобяваме чрез конкатенация търсения образ.

Бимашина $B =< A_L, A_R, \Psi >$ е тотална, когато двата ѝ подлежащи автомата A_L и A_R са тотални и функцията на изходите
$\Psi\colon (Q_L \times \Sigma \times Q_R) \mapsto \Sigma^*$ също е тотална. Очевидно е, че в този случай функцията, представена от B, ще е дефинирана за всяка дума $w \in \Sigma^*$. Т.е. $\mathbb{O}_B$ също е тотална.

Известен резултат (формално описан например в [2]) е, че множеството от всички функции представени от бимашини съвпада с множеството на регулярните низови функции, които изобразяват ε в ε. От теорията за крайните машини на състоянията и по-конкретно обобщената теорема на Клини знаем, че класът от всички регулярни низови релации е представен от класа на всички крайни двулентови автомати, още наричани крайни преобразуватели. В частност крайните функционални преобразуватели представят класа на регулярните низови функции. Добре известни са алгоритми за конструиране на бимашина по зададен функционален преобразувател. Темата е разгледана в [2].

3 Неформално описание на идеята за компресиране

3.1 Бимашини и функционални преобразуватели

Както вече споменахме, голямо предимство на бимашините пред функционалните преобразуватели е детерминистичното им поведение при получаване на изхода, което гарантира линейно време за пресмятането му спрямо дължината на входа и не зависи от броя състояния на двата автомата. Недетерминистичната природа на функционалните преобразуватели предполага траверсиране на множество пътища за получаване на търсения образ, което на практика често води до линейно време спрямо броя на състоянията му за пресмятане на резултата.

От друга страна, за практическото представянето на една бимашина $B =< A_L, A_R, \Psi >$ обикновено е необходима много повече памет отколкото за представянето на еквивалентния функционален преобразувател $T =< \Sigma, Q, I, F, \Delta >$. При подхода за конструиране на B по дадено T, описан в [2], левият детерминиран автомат A_L на бимашината е получен чрез детерминизация на подлежащия по първата лента автомат на T. A_R се строи като се детерминизира обърнатия подлежащ автомат на T. В статията [3] е показано, че в някои от по-лошите случаи сумата от състоянията в A_L и A_R може да достигне $\Theta(|Q|!)$. На практика най-много памет заема функцията на изходите Ψ. Когато е представена директно, се изисква памет пропорционална на $|A_L| \cdot |\Sigma| \cdot |A_R|$.

При конструиране на функцията на изходите Ψ същият подход избира думи от втората лента на изходния преобразувател T за функционални стойности. Това означава, че $|Range(\Psi)|$ е ограничен от $|Proj_3(\Delta)|$ - броя уникални думи по втората лента на T. В практиката често това е много по-малка стойност от $|Dom(\Psi)|$. Получава се така, че много наредени тройки от $A_L \times \Sigma \times A_R$ имат еднакви стойности. Описаният по-долу метод дава добри резултати за компресирано представяне на функции от такъв тип. Като важно изискване е извличането на функционалните стойности от компресираната структура да става за близко до константно време. По този начин се запазват желаните свойства на представената бимашина за линейно обработване на входа.

3.2 Пример за компресия на функция с повтарящи се стойности

За неформалното представяне на избрания подход за компресиране нека първо разгледаме функцията на преходите δ на произволен тотален детерминиран автомат. $\delta\colon Q \times \Sigma \mapsto Q$ е тотална функция. Нека вземем за пример автомат с множество от 3 състояния $Q = \{q_1, q_2, q_3\}$, азбука от 4 символа $\Sigma = \{a, b, c, d\}$ и функция на преходите δ:

$$\begin{aligned}\delta = \{&((q_1, a), q_2), ((q_1, b), q_1), ((q_1, c), q_1), ((q_1, d), q_2),\\ &((q_2, a), q_3), ((q_2, b), q_1), ((q_2, c), q_3), ((q_2, d), q_2),\\ &((q_3, a), q_2), ((q_3, b), q_3), ((q_3, c), q_3), ((q_3, d), q_1)\}.\end{aligned}$$

Да разгледаме δ_{q_1} - проекцията на δ върху състояние q_1. Тя може да бъде представена като четиримерен вектор $\vec{\delta_{q_1}}$ от множеството Q^4, като за първа координата вземаме стойността $\delta_{q_1}(a)$, за втора - $\delta_{q_1}(b)$, и т.н. Аналогично подхождаме за δ_{q_2} и δ_{q_3}. Така получаваме представяне на δ под формата на три вектора:

$$\begin{aligned}\vec{\delta_{q_1}} &= (q_2, q_1, q_1, q_2),\\ \vec{\delta_{q_2}} &= (q_3, q_1, q_3, q_2),\\ \vec{\delta_{q_3}} &= (q_2, q_3, q_3, q_1).\end{aligned}$$

Нека вземем четвърти вектор $\vec{c} = (q_2, q_1, q_3, q_2)$, който ще наричаме Хаминг център на множеството вектори $\{\vec{\delta_{q_1}}, \vec{\delta_{q_2}}, \vec{\delta_{q_3}}\}$. Това е този вектор, който минимизира $\sum_{i=1}^{3} H_d(\vec{c}, \vec{\delta_{q_i}})$. Тук с $H_d(\vec{c}, \vec{\delta_{q_i}})$ бележим разстоянието на Хаминг между $\vec{c}$ и $\vec{\delta_{q_i}}$, пресметнато като намерим броя на координатите, в които векторите имат различни стойности.

Следващата стъпка е да конструираме частична функция $\hat{\delta_{q_1}}$ от компонентите на $\vec{\delta_{q_1}}$, които се различават от съответните компоненти в $\vec{c}$.

$$\hat{\delta_{q_1}} = \{(\sigma, \vec{\delta_{q_1}}_{\sigma}) | \vec{\delta_{q_1}}_{\sigma} \neq \vec{c}_{\sigma}\}$$

Така в $\hat{\delta_{q_1}}$ имаме $H_d(\vec{c}, \vec{\delta_{q_1}})$ на брой стойности. За да получим стойността на δ_{q_1} за даден символ, първо проверяваме дали $\hat{\delta_{q_1}}$ е дефинирана за него. Тогава връщаме дадения резултат. В противен случай стойността е същата като съответната координата на $\vec{c}$.

За представянето на цялата функция δ ползваме $\hat{\delta_{q_1}}$, $\hat{\delta_{q_2}}$, $\hat{\delta_{q_3}}$ и $\vec{c}$, организирани йерархично в дърво T_δ с корен $\vec{c}$.

$$\vec{c} = (q_2, q_1, q_3, q_2)$$
$$\hat{\delta_{q_1}} = ((c, q_1)), \hat{\delta_{q_2}} = ((a, q_3)), \hat{\delta_{q_3}} = ((b, q_3), (d, q_1))$$

Във всяко от листата са кодирани преходите на δ от съответното състояние. Общия брой на експлицитно представените функционални стойности в T_δ е:

$$|\vec{c}| + H_d(\vec{c}, \vec{\delta_{q_1}}) + H_d(\vec{c}, \vec{\delta_{q_2}}) + H_d(\vec{c}, \vec{\delta_{q_3}}) = 4 + 1 + 1 + 2 = 8$$

В случая, когато директно представим δ, пазим 12 функционални стойности - $|Q| \times |\Sigma|$. Към общата памет, необходима за представянето на T_δ, също трябва да отчетем и паметта за указателите от листата към корена - пропорционална на размера на дървото.

Нека отбележим, че за по-ефективно представяне на частичните функции $\hat{\delta_{q_i}}$ могат да бъдат използвани различни видове хеш функции или техники за представяне на разредени таблици - [4].

3.3 Оптимална компресия и дървета на Щайнер

Горният пример има за цел да илюстрира общия подход при компресиране на функции на преходите. В практиката често се налага компресиране на функции от автомати с милиони състояния и азбуки от стотици символи. В такава ситуация подхождаме аналогично на примера. Първо разделяме функцията на преходите като списък от вектори, представящи преходите от всяко състояние с всеки символ. Така получаваме $|Q|$ на брой $|\Sigma|$-мерни вектори (δ-вектори).

В основата на концепцията за намаляване на заеманата от δ памет стои идеята за намирането на нов централен вектор, притежаващ общи компоненти с максимален брой δ-вектори. Вижда се, че колкото по-малка е сумата от Хаминг разстоянията между центъра и първоначалния набор от вектори, съществуват толкова повече общи компоненти между векторите и центъра. Това от своя страна предполага повече функционални стойности, които могат да бъдат представени имплицитно посредством родителски връзки в дървовидната структура. За постигане на максимална компресия не е достатъчно да въведем единствен нов централен вектор. В много случаи оптималния резултат се получава, когато се включат множество нови вектори, организирани помежду си заедно с първоначалното множество в дървовидна структура.

Нека сме конструирали описаната дървовидна структура T с върхове от първоначалните $|Q|$ вектори и K на брой нови вектори. Нека сумата от Хаминг разстоянията между векторите в двата края на всяко ребро е Ф. Всеки от върховете представя функция със Σ на брой двойки от аргумент и стойност. Нека изберем посока на ребрата в даденото дърво T, така че то да стане насочено дърво. За всеки връх u, който е различен

от корена r, можем експлицитно да кодираме функционалните двойки с различни стойности от неговия родител P_u. Така за този връх пазим $H_d(u, P_u)$ двойки. За корена пазим експлицитно всичките $|\Sigma|$ стойности. Тогава броят на функционалните двойки в цялото дърво ще бъде $|\Sigma| + \sum_{u \in V(T) \setminus \{r\}} H_d(u, P_u) = |\Sigma| + \Phi$. Тук с $V(T)$ бележим множеството от върховете на T. За представянето на цялото дърво очевидно ще ни е необходимо да пазим $|Q| + K - 1$ на брой двойки върхове, представящи ребрата. Следователно основната ни цел е да намерим такова дърво, което минимизира $\Phi + K$. Нека на този етап се абстрахираме от броя K на въведените нови върхове и се фокусираме върху дървото, минимизиращо Φ.

Съществува формална дефиниция на такъв проблем - намиране на минимално претеглено Щайнер дърво в Хаминг пространство. За референция - [5]. Нека си представим пълен граф G, чийто върхове представляват всички вектори от множеството $Q^{|\Sigma|}$. На ребрата, свързващи всеки два върха, приписваме за тежест Хаминг разстоянието между съответните два вектора. Целта на задачата е по дадено първоначално множество X от δ-вектори да се намери дърво в G, покриващо всички върхове от X (като може да включва и други върхове извън X), с минимална сума от тегла по ребрата.

В книгата на Гусфилд [5] е обобщено, че проблемът за намиране на минимално дърво на Щайнер е NP-труден. Не ни е известен практически приложим метод за решаване на проблема върху граф G, съдържащ брой върхове от такъв голям порядък - $|Q^{|\Sigma|}|$. Освен това оптималното дърво, постигащо минимална сума от Хаминг разстояния между върховете си, може да има твърде голяма дълбочина. В такъв случай пресмятането на функционалните стойности при някои аргументи би отнело значително по-голямо от константно време.

3.4 Компресия и клъстеризиране на вектори

С оглед на горното, текущият метод за компресия не цели да намери оптималната дървовидна структура. Вместо това ще строим дърво, разделено на няколко нива, започвайки с началното множество от δ-вектори. Те ще са върхове от нулевото ниво или листа. Избираме K на брой нови върха, които ще формират първото ниво и техни наследници ще са върхове от долното ниво. Може да продължим добавяне на нови нива, докато не достигнем определена дълбочина на дървото, като в последното ниво имаме единствен връх - корен. С ограничаване на дълбочината си осигуряваме бърз метод за пресмятане на функционалните стойности.

Основен елемент при описания подход е избирането на новите върхо-

ве, съставящи всяко следващо горно ниво. За целта групираме векторите от текущото ниво в K на брой клъстера. Тогава Хаминг центровете на векторите от всеки клъстер формират върховете от следващото ниво. Като всеки център става родител на векторите от съответния му клъстер. Така свеждаме проблема до решаване на добре известната задача за клъстеризиране или K-means clustering.

В класическата формулировка целта на задачата е да се разделят N на брой точки в d-мерно Евклидово пространство на K групи или клъстера. Като също така се цели сумата на разстоянията от всяка точка до центъра на масите на точките от съответния ѝ клъстер да е минимално. От семантична гледна точка такова клъстеризиране обособява групи от елементи, които са близки в смисъла на Евклидово разстояние помежду си. В общия си вид задачата е NP-трудна, но съществува добре наложен в практиката алгоритъм, известен като K-means, намиращ в много случаи достатъчно приближено решение за приемливо време. За описание на проблема, неговата сложност и алгоритъма може да посочим [6].

За целите на текущата компресия правим модификация по дефиницията на проблема, като го пренасяме от Евклидово в Хаминг пространство. Искаме да получим групиране на $|Q|$ вектора в K клъстера, като целим да минимизираме сумата от Хаминг разстоянията между всеки вектор и Хаминг центъра на неговия клъстер. Нека u е някой вектор и с $Cl(u)$ бележим Хаминг центъра на неговия клъстер. Ако с Q_δ бележим множеството от началните δ-вектори, тогава сумата, която искаме да оптимизираме е $\Phi = \sum_{u \in Q_\delta} H_d\big(u, Cl(u)\big)$. Подобно на горния пример, Хаминг център на някой от формираните клъстери кодира функция на преходите, имаща максимален брой общи стойности с функциите, представени от векторите в клъстера. Така формираме първото ниво от K на брой дървета с корен, център на клъстер, и листа, съответстващи на векторите в клъстера. В листата експлицитно кодираме само преходите, които се различават от тези, кодирани в корена.

Алгоритъмът за клъстеризиране на δ-вектори следва същата конструкция като класическия k-means алгоритъм. На подготвителната фаза искаме да подберем K вектора, които да играят ролята на началното множество от Хаминг центрове. Съществуват различни стратегии за начален подбор на центрове. Един от по-простите подходи е те да бъдат случайно равномерно избрани от даденото на входа множество.

Разполагайки с начално множество от K центъра, продължаваме, като асоциираме всеки δ-вектор с най-близкия по Хаминг разстояние до него център. По този начин получаваме началното разделяне на векторите в K клъстера. Като база за следващата итерация намираме Хаминг центровете на векторите от всеки формиран в тази стъпка клъстер, полу-

чавайки K нови центъра. След това правим следващо клъстеризиране спрямо новото множество от Хаминг центрове. Продължаваме да итерираме, докато сумата Φ спре да намалява. В следващата секция ще докажем, че този процес е сходящ и алгоритъмът винаги приключва изпълнението си.

3.5 Илюстрация на метода за Ψ-функция

Нека се върнем към основния проблем за компресираното представяне на Ψ-функцията на бимашина B. За разлика от функцията на преходите, тя е дефинирана върху $Q_L \times \Sigma \times Q_R$. Това предполага повече възможни начини за нейното представяне като списък от вектори. Един вариант е да я разбием на $|Q_L| \cdot |Q_R|$ на брой $|\Sigma|$-мерни вектора, като получаваме вектор от всяка Ψ_{q_l,q_r} проекция на Ψ, за $q_l \in Q_L$ и $q_r \in Q_R$. За векторен компонент, съответстващ на $\sigma \in \Sigma$, вземаме стойността на $\Psi(q_l, \sigma, q_r)$. Друга възможност за представяне на Ψ е да я разделим на $|Q_L|$ на брой $|Q_R| \cdot |\Sigma|$-мерни вектора. Имаме вектор за всяка проекция Ψ_{q_l}, където $q_l \in Q_L$. За първите $|\Sigma|$ координати вземаме съответните стойности на Ψ за даденото q_l и фиксирано първо състояние от Q_R. Следващите $|\Sigma|$ стойности получаваме при фиксирано второ състояние от десния автомат и т.н.

При клъстеризиране на вектори от първия вид имаме по-добра грануларност при улавяне на регулярности във функционалните стойности, тъй като не обвързваме позициите на векторните компоненти с дадено състояние. От друга страна, така увеличаваме броя на векторите, което води до забавяне на процедурата по клъстеризиране и изисква повече памет за представяне на връзките към родителите при конструиране на дърветата.

След проведени практически опити бяха наблюдавани по-добри резултати при използването на хибриден подход. Първоначално разделяме Ψ-функцията на $|Q_L|$ вектора по втория описан начин. Нека ги наречем върхове от нулево ниво или листа. Построяваме клъстеризиране на векторите на K_1 групи, като за всяка имаме център с размерност $|Q_R| \cdot |\Sigma|$. Ще ги наричаме върхове на първо ниво. След това разделяме всеки център по фиксираните състояния от десния автомат, получавайки общо $K_1 \cdot |Q_R|$ на брой $|\Sigma|$-мерни вектора. Върху новото множество отново прилагаме метода за клъстеризиране и намираме K_2 нови центъра - върхове от второ ниво. За крайното представяне на функцията строим от векторите дървовидна структура на 3 нива подобно на горния пример. Именно този подход ще опишем формално в следващата секция.

Нека отбележим, че ограничаваме текущата структура до 3 нива,

тъй като искаме да запазим близко до константно време за извличане на функционалните стойности.

Друг важен детайл, който трябва да споменем, е изборът на K - броят на клъстерите при групиране на N вектора. Очевидно е, че крайният резултат от компресията силно зависи от правилно подбрана стойност за K. Практическите опити показват, че добри резултати се получават, когато K е близко по стойност до $\sqrt{N}$. Често е достатъчно практично да се изпробва клъстеризиране с различни стойности за K и да се избере това, което дава най-добра компресия.

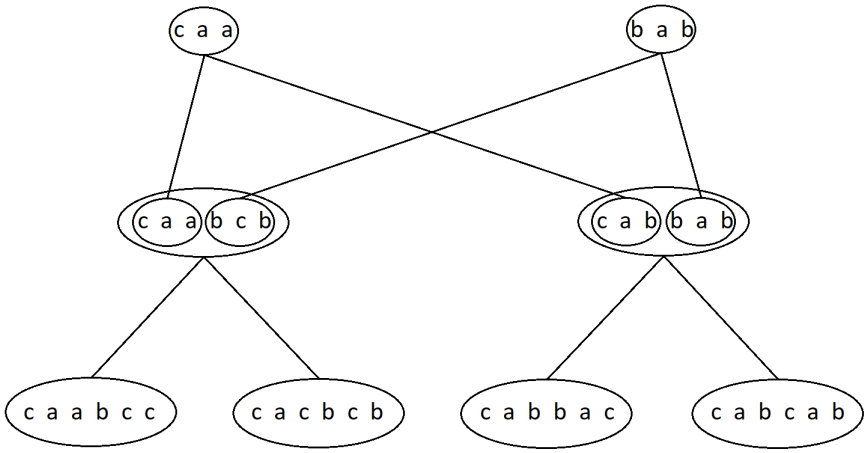
Нека разгледаме пример, демонстриращ представянето на дадена функция на изходите. Да вземем бимашина с азбука $\Sigma = \{a, b, c\}$, състояния на левия автомат $Q_L = \{1, 2, 3, 4\}$ и състояния на десния - $Q_R = \{5, 6\}$. Нека имаме функция на изходите Ψ, където:

$$\Psi(1,a,5) = c, \Psi(1,b,5) = a, \Psi(1,c,5) = a, \Psi(1,a,6) = b, \Psi(1,b,6) = c, \Psi(1,c,6) = c,$$
$$\Psi(2,a,5) = c, \Psi(2,b,5) = b, \Psi(2,c,5) = c, \Psi(2,a,6) = b, \Psi(2,b,6) = c, \Psi(2,c,6) = b,$$
$$\Psi(3,a,5) = c, \Psi(3,b,5) = a, \Psi(3,c,5) = b, \Psi(3,a,6) = b, \Psi(3,b,6) = a, \Psi(3,c,6) = c,$$
$$\Psi(4,a,5) = c, \Psi(4,b,5) = a, \Psi(4,c,5) = b, \Psi(4,a,6) = c, \Psi(4,b,6) = a, \Psi(4,c,6) = b.$$

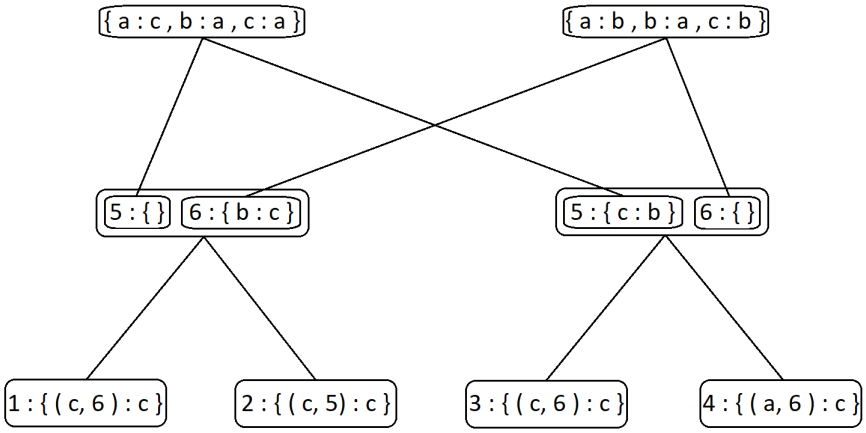
Започваме, като разделим Ψ на четири вектора, кодиращи функционалните стойности при фиксиран първи аргумент - $v_1 = (c, a, a, b, c, c), v_2 = (c, a, c, b, c, b), v_3 = (c, a, b, b, a, c)$ и $v4 = (c, a, b, c, a, b)$. Групираме векторите в два клъстера - $C'_1 = \{v_1, v_2\}$ и $C'_2 = \{v_3, v_4\}$, чиито центрове са съответно $v_5 = (c, a, a, b, c, b)$ и $v_6 = (c, a, b, b, a, b)$.

На втората стъпка разбиваме центровете v_5 и v_6 на по 2 нови вектора, кодиращи преходите при фиксирани първи и трети аргумент на Ψ. Получаваме $u_1 = (c, a, a), u_2 = (b, c, b), u_3 = (c, a, b)$ и $u_4 = (b, a, b)$. Клъстеризираме новия списък от 4 вектора и получаваме два клъстера - $C''_1 = \{u_1, u_3\}$ и $C''_2 = \{u_2, u_4\}$, които имат центрове съответно $v_5 = (c, a, a)$ и $v_6 = (b, a, b)$. На фигура 1 е илюстрирана йерархичната организация на горните вектори.

Накрая конструираме частични функции, като дефинираме стойности, които отразяват разликата между вектор и неговия родител. В съответствие на v_1, v_2, v_3 и v_4 имаме $\hat{v_1} = \{((c, 6), c)\}, \hat{v_2} = \{((c, 5), c)\}, \hat{v_3} = \{((c, 6), c)\}$ и $\hat{v_4} = \{((a, 6), c)\}$. За $u_1, \ldots, u_6$ получаваме съответно $\hat{u_1} = \{\}, \hat{u_2} = \{(b, c)\}, \hat{u_3} = \{(c, b)\}, \hat{u_4} = \{\}, \hat{u_5} = \{(a, c), (b, a), (c, a)\}$ и $\hat{u_6} = \{(a, b), (b, a), (c, b)\}$. Частичните функции $\hat{v_1}, \ldots, \hat{v_4}$ и $\hat{u_1}, \ldots, \hat{u_6}$ заедно с родителските връзки между тях, изобразени на фигура 2, представят крайния вид на компресираната функция Ψ.



Фигура 1: Йерархична организация на векторите от примера.



Фигура 2: Йерархично представяне на компресирана Ψ-функция.

4 Формално описание на алгоритъма за компресиране на Ψ-функция на бимашина

4.1 Хаминг разстояние между два вектора

Нека u и v са N-мерни вектори от U^N, където U е произволно множество. Първо дефинираме спомагателна булева функция Neq, индикираща

неравенство между два елемента на U.

$$Neq : U \times U \to \{0,1\},$$
$$Neq(c_1, c_2) = \begin{cases} 0, \text{ ако } c_1 = c_2 \\ 1, \text{ иначе} \end{cases} .$$

Нека с $(u)_i$ бележим i-тия компонент на вектор $u \in U^N$, за $i \in \{1 \dots N\}$. Тогава с $H_d(u, v)$ бележим разстоянието на Хаминг между u и v.

$$H_d : U^N \times U^N \to \mathbb{N},$$
$$H_d(u, v) = \sum_{i=1}^{N} Neq\big((u)_i, (v)_i\big).$$

4.2 Хаминг център на списък от вектори

Ако A е списък от k елемента на U, с $(A)_i$ бележим i-тия му елемент, а с $Mf(A)$ бележим произволен, най-често срещан елемент в списъка (т.е. някой от елементите с най-голям брой срещания в A). Нека L е списък от n на брой N-мерни вектора, принадлежащи на U^N - $L = [u_1, u_2, \dots, u_n]$. Използваме $Pr_i(L)$ да означаваме проекцията на L по координата $i \in \{1 \dots N\}$. Тоест $Pr_i(L)$ е списък от i-тите компоненти на векторите от L.

$$Eq : U \times U \to \{0,1\}, \quad Eq(c_1, c_2) = \begin{cases} 1, \text{ ако } c_1 = c_2 \\ 0, \text{ иначе} \end{cases} ,$$
$$Cnt : U^* \times U \to \mathbb{N}, \quad Cnt(A, c) = \sum_{i=1}^{k} Eq\big((A)_i, c\big),$$
$$Mf : U^* \to U, \quad Mf(A) = \operatorname*{argmax}_{c \in A} Cnt(A, c),$$
$$Pr_i : (U^N)^* \to U^* \quad Pr_i(L) = [(u_1)_i, (u_2)_i, \dots, (u_n)_i].$$

Тогава векторът $H_c(L)$ наричаме Хаминг център на списъка от вектори L.

$$H_c : (U^N)^* \to U^N,$$
$$H_c(L) = \langle Mf\big(Pr_1(L)\big), Mf\big(Pr_2(L)\big), \dots, Mf\big(Pr_N(L)\big)\rangle.$$

Свойство 4.2.1. *Нека имаме списък от вектори $L = [u_1, u_2, \dots, u_n]$. Нека v е произволен вектор от U^N, тогава сумата $\sum_{i=1}^{n} H_d(v, u_i)$ се*

минимизира при стойност на v, равна на Хаминг центъра на L. Т.е. за $c = H_c(L)$ и произволно $v \in U^N$ имаме:

$$\sum_{i=1}^{n} H_d(c, u_i) \leq \sum_{i=1}^{n} H_d(v, u_i).$$

Доказателство. Нека v е произволен вектор от U^N. Тогава:

$$\begin{aligned}
\sum_{i=1}^{n} H_d(v, u_i) &= \sum_{i=1}^{n}\sum_{j=1}^{N} Neq((v)_j, (u_i)_j) = \sum_{j=1}^{N}\sum_{i=1}^{n} Neq((v)_j, (u_i)_j) \\
&\geq \sum_{j=1}^{N}\sum_{i=1}^{n} Neq(Mf((u_1)_j, (u_2)_j, \ldots, (u_n)_j), (u_i)_j) \\
&= \sum_{j=1}^{N}\sum_{i=1}^{n} Neq(Mf(Pr_j([u_1, u_2, \ldots, u_n]), (u_i)_j) \\
&= \sum_{j=1}^{N}\sum_{i=1}^{n} Neq(Mf(Pr_j(L)), (u_i)_j) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{N} Neq(Mf(Pr_j(L)), (u_i)_j) \\
&= \sum_{i=1}^{n} H_d(\langle Mf(Pr_1(L)), Mf(Pr_2(L)), \ldots, Mf(Pr_N(L))\rangle, u_i) \\
&= \sum_{i=1}^{n} H_d(H_c(L), u_i) = \sum_{i=1}^{n} H_d(c, u_i).
\end{aligned}$$

□

В горното доказателство използваме тривиалното твърдение, че при даден списък от елементи $A = [c_1, c_2, \ldots, c_n] \in U^*$ и произволно $t \in U$ е в сила:

$$\sum_{i=1}^{n} Neq(Mf(A), c_i) \leq \sum_{i=1}^{n} Neq(t, c_i).$$

Което ще рече: броят на различните между произволен елемент t и елементите от списъка A е по-голям или равен на броя на различните между най-често срещания елемент $Mf(A)$ и елементите в A.

4.3 K-means клъстеризиране в Хаминг пространство

Нека е даден списък L от n на брой вектори от множеството U^N. Също така е дадено $K \in \mathbb{N}, K > 0$. По даден списък $C = [c_1, c_2, \ldots, c_K]$ от K на брой вектори (центрове), принадлежащи на U^N, дефинираме $Cl(u, C)$ да е най-близкия по Хаминг разстояние до u център от C.

$$Cl : U^N \times (U^N)^* \to U^N,$$
$$Cl(u, C) = \operatorname*{argmin}_{c \in C} H_d(u, c).$$

Клъстеризиране, определено от C, наричаме множеството от K списъка (или клъстера) $S_1, S_2, \ldots, S_K$, за които $S_i = [u \mid u \in L \,\&\, Cl(u, C) = c_i]$. Целта на задачата е да намерим списък C от K на брой вектори в L и съответно клъстеризиране $S_1, S_2, \ldots, S_K$, за които постигаме минимална стойност на сумата:

$$\Phi(C) = \sum_{i=1}^{K} \sum_{u \in S_i} H_d(c_i, u).$$

Методът за клъстеризиране започва с избирането на първоначалния списък C_I от K центъра. Нека за C_I сме взели някои K на брой различни вектори.

Втората част от метода за клъстеризиране работи итеративно, тръгвайки от началния списък от центрове C_I. На всяка стъпка подобряваме сумата $\Phi(C^{(i)})$, като заменяме всеки център от текущия списък с Хаминг центъра на породения от него клъстер. Ще дадем индуктивна конструкция за получаване на крайния списък от центрове C.

Стъпка 1: Започваме с построения в предната част списък C_I. Т.е. имаме:

$$C^{(0)} = C_I.$$

Стъпка $i + 1$**:** Нека след i-тата стъпка имаме $C^{(i)}$. За всеки център от $C^{(i)}$ сме пресметнали съответния клъстер, определен от него - $S_j^{(i)}$. На текущата стъпка конструираме нов списък от Хаминг центровете на съответните клъстери:

$$S_j^{(i)} = [u \mid u \in L \,\&\, Cl(u, C^{(i)}) = c_j^{(i)}], \quad 1 \leq j \leq K,$$
$$C^{(i+1)} = [H_c(S_j^{(i)}) \mid 1 \leq j \leq K].$$

Индукцията приключва, когато сумата $\Phi(C^{(i)})$ спре да намалява.

4.4 Коректност на алгоритъма за клъстеризиране

Тук формално ще покажем, че горният алгоритъм приключва своето изпълнение след краен брой стъпки.

Доказателство. За целта ще докажем, че стойността на сумата $\Phi(C)$ намалява след всяка итерация на индукцията с изключение на последната.
Нека след приключването на стъпка i сме конструирали множество от центрове $C^{(i)} = [c_1^{(i)}, c_2^{(i)}, \ldots, c_K^{(i)}]$. Клъстерите, определени от $C^{(i)}$, са $S_1^{(i)}$, $S_2^{(i)}$, $\ldots$, $S_K^{(i)}$. Сумата от Хаминг разстоянията между векторите и съответните им центрове е $\Phi(C^{(i)}) = \sum_{j=1}^{K} \sum_{u \in S_j^{(i)}} H_d(c_j^{(i)}, u)$. Нека на стъпка $i{+}1$ идентифицираме Хаминг центъра $c_j^{(i+1)}$ на всеки от получените клъстери $S_j^{(i)}$, за $j \in \{1, \ldots, K\}$. Използвайки свойството 4.2.1, доказано по-горе, получаваме $\sum_{u \in S_j^{(i)}} H_d(c_j^{(i+1)}, u_i) \leq \sum_{u \in S_j^{(i)}} H_d(c_j^{(i)}, u_i)$. Т.е. за всеки клъстер $S_j^{(i)}$ неговият Хаминг центъра $c_j^{(i+1)}$ подобрява (или не променя) сумата от разстоянията в сравнение с вектора $c_j^{(i)}$, породил клъстера.
Нека дефинираме помощна функция $Cli^{(i)}(u)$, която връща индекса на клъстера, в който се намира u след приключването на стъпка i.

$$Cli^{(i)} : U^N \to \{1, \ldots, K\},$$
$$Cli^{(i)} = \{(u, c) \mid u \in L, c = IndexOf_{C^{(i)}}(Cl(u, C^{(i)})\}$$

След като сме намерили новия списък от центрове $c_j^{(i+1)}$, отново разпределяме всеки от векторите в L към най-близкия му център. От дефиницията на функцията Cl следва, че за всяко $u \in L$ е вярно $H_d(c_{Cli^{(i)}(u)}^{(i+1)}, u) \geq H_d(Cl(u, C^{(i+1)}), u)$. Което ще рече - разстоянието между u и Хаминг центъра на стария му клъстер(, принадлежащ на $C^{(j+1)}$) е по-голямо или равно на разстоянието от u до най-близкия до него вектор от $C^{(j+1)}$.

Тогава имаме:

$$\begin{aligned}\Phi(C^{(i)}) &= \sum_{j=1}^{K}\sum_{u\in S_j^{(i)}} H_d(c_j^{(i)},u) \geq \sum_{j=1}^{K}\sum_{u\in S_j^{(i)}} H_d(c_j^{(i+1)},u)\\ &= \sum_{u\in L} H_d(c^{(i+1)}_{Cli^{(i)}(u)},u)\\ &\geq \sum_{u\in L} H_d(Cl(u,C^{(i+1)}),u)\\ &= \sum_{j=1}^{K}\sum_{u\in S_j^{(i+1)}} H_d(c_j^{(i+1)},u) = \Phi(C^{(i+1)}).\end{aligned}$$

Ясно е, че $\Phi(C)$ може да заема единствено стойности от естествените числа, т.к. е крайна сума от Хаминг разстояния. Следователно е ограничена отдолу от нулата. Тъй като след всяка итерация на горния алгоритъм $\Phi(C)$ единствено намалява или остава непроменена, следва, че той ще приключи изпълнението си след краен брой стъпки. □

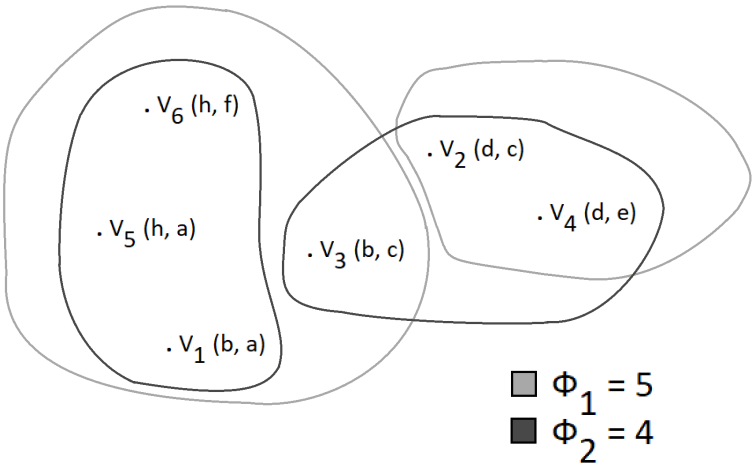
Нека отбележим, че гореизложеният алгоритъм не предоставя асимптотична граница за броя на итерациите при изпълнението си. Също така не може да се гарантира, че намереното решение определя локален минимум на функцията $\Phi(C)$. Може да покажем пример, илюстриращ как различни стратегии за решаване на равенствата при асоциирането на даден вектор с най-близък до него център биха довели до различен краен резултат.

Нека имаме $\Sigma = \{a,b,c,d,e,f,h\}$ и $L \in \Sigma^2, L = [v_1 = (b,a), v_2 = (d,c), v_3 = (b,c), v_4 = (d,e), v_5 = (h,a), v_6 = (h,f)]$. Искаме разделяне на 2 клъстера ($K = 2$). Нека $C^{(1)} = [v_1, v_2]$ е началният списък от центрове. Векторите v_3 и v_6 са на равно Хаминг разстояние от текущите центрове v_1 и v_2. В следствие имаме различни възможности за формиране на първоначалните клъстери. Ако групираме v_3 и v_6 в клъстер с v_1, получаваме $C_1'^{(1)} = [v_1, v_3, v_5, v_6]$ и $C_2'^{(1)} = [v_2, v_4]$. Векторите v_1 и v_2 са Хаминг центрове на съответните клъстери ($C^{(2)} = [v_1, v_2]$) и алгоритъмът приключва след текущата итерация, давайки резултат $\Phi_1 = H_d(v_1,v_1) + H_d(v_1,v_3) + H_d(v_1,v_5) + H_d(v_1,v_6) + H_d(v_2,v_2) + H_d(v_2,v_4) = 5$.

От друга страна, може да образуваме клъстери $C_1''^{(1)} = [v_1, v_5, v_6]$ и $C_2''^{(1)} = [v_2, v_3, v_4]$. Така след края на първата итерация получаваме нов списък от центрове $C''^{(2)} = [v_5, v_2]$, определен от съответните Хаминг центровете. Следващата итерация не променя клъстерите и тогава спираме

изпълнение. Крайният резултат е $\Phi_2 = H_d(v_5, v_5) + H_d(v_5, v_1) + H_d(v_5, v_6) + H_d(v_2, v_2) + H_d(v_2, v_3) + H_d(v_2, v_4) = 4$.

Диаграмата на фигура 3 съдържа визуално представяне на двете групирания в горния пример.



Фигура 3: Различни групирания на векторите от списъка L.

Примерът показва, че съществуват конфигурации на равенство, при които няма очевидна стратегия на избор, която би довела до най-добрия локален резултат. Понякога е достатъчно практично алгоритъмът за клъстеризиране да бъде изпълнен многократно с различни списъци от начални центрове и накрая да бъде избран най-подходящият резултат.

4.5 Метод за компресиране на бимашина

Нека е дадена бимашина $B =< A_L, A_R, \Psi >$ с ляв автомат $A_L =< \Sigma, Q_L, s_L, Q_L, \delta_L >$, десен автомат $A_R =< \Sigma, Q_R, s_R, Q_R, \delta_R >$ и функция на изходите $\Psi : Q_L \times \Sigma \times Q_R \to \Sigma^*$. Целта ни е да получим компресирано представяне на δ_L, δ_R и Ψ.

Нека означим $|Q_L| = L$, $|\Sigma| = S$ и $|Q_R| = R$. Също така нека индексираме елементите на Q_L, Σ и Q_R и нека разглеждаме тези множества като списъци от елементи. Имаме $Q_L = [l_1, l_2, \ldots, l_L]$, $\Sigma = [\sigma_1, \sigma_2, \ldots, \sigma_S]$ и $Q_R = [r_1, r_2, \ldots, r_R]$. Ако имаме списък L, съдържащ елемент u, използваме функцията $IndexOf_L(u)$ да означим индекса на u в L. Така например $IndexOf_\Sigma(\sigma_i) = i$, за $1 \leq i \leq S$.

4.5.1 Компресиране на δ-функция

Нека започнем с метода за компресия на функцията на преходите δ_L на левия краен детерминиран автомат от B. Ще представим δ_L като списък от $|Q_L|$ на брой $|\Sigma|$-мерни вектори $\vec{\delta_i}$. За $1 \leq i \leq |Q_L|$ имаме:

$$\vec{\delta_i} = \langle \delta_L(l_i, \sigma_1), \delta_L(l_i, \sigma_2), \ldots, \delta_L(l_i, \sigma_S) \rangle.$$

Сега следва да приложим горния метод за K-means клъстеризиране на векторите $\vec{\delta_i}$, като за стойност на K избираме $\sqrt{L}$. В резултат получаваме списък C от K на брой центъра на клъстери:

$$C = Cluster_{kmeans}(\sqrt{L}, [\vec{\delta_i} \mid i \in \{1 \ldots L\}]).$$

Като следваща стъпка ще конструираме частични функции η'_i, представящи компонентите с различни стойности между всеки вектор $\vec{\delta_i}$ и Хаминг центъра на съответния му клъстер $Cl(\vec{\delta_i}, C)$:

$$\eta'_i : \Sigma \to Q_L,$$
$$\eta'_i = \{(\sigma, (\vec{\delta_i})_j) \mid \sigma \in \Sigma, j = IndexOf_\Sigma(\sigma), p = Cl(\vec{\delta_i}, C) \,\&\, (p)_j \neq (\vec{\delta_i})_j\}.$$

Ще дефинираме и функция T, която да представя родителската връзка от вектор $\vec{\delta_i}$ към центъра на клъстера му:

$$T : \mathbb{N} \to \mathbb{N},$$
$$T = \{(i, j) \mid 1 \leq i \leq L, p = Cl(\vec{\delta_i}, C), j = IndexOf_C(p)\}.$$

От дефинициите на T, C и функциите η' се вижда директно равенството за получаване на стойностите на δ_L:

$$\delta_L(q, \sigma) = \begin{cases} \eta'_i(\sigma), \text{ ако } !\eta'_i(\sigma), \text{ където } i = IndexOf_{Q_L}(q) \\ (p)_j, \text{ иначе, където } j = IndexOf_\Sigma(\sigma), k = T(i), p = (C)_k \end{cases}.$$

Нека построим още един етаж на текущата йерархична структура. За целта нека означим с r Хаминг центъра на първия списък от центрове на клъстери C - $r = H_c(C)$. Отново правим функционално представяне на векторите от C. Като за $i \in \{1 \ldots K\}$ имаме:

$$\eta''_i : \Sigma \to Q_L,$$
$$\eta''_i = \{(\sigma, (v)_j) \mid \sigma \in \Sigma, j = IndexOf_\Sigma(\sigma), v = (C)_i \,\&\, (v)_j \neq (r)_j\}.$$

Сега сме готови да дефинираме компресираната форма на δ_L - δ_L^C, използвайки r, T, η' и η'':

$$\delta_L^C : Q_L \times \Sigma \to Q_L,$$
$$\delta_L^C(q,\sigma) = \begin{cases} \eta'_i(\sigma), \text{ ако } !\eta'_i(\sigma), \text{ където } i = IndexOf_{Q_L}(q) \\ \eta''_j(\sigma), \text{ ако } !\eta''_j(\sigma), \text{ където } j = T(i) \\ (r)_k, \text{ иначе, където } k = IndexOf_{\Sigma}(\sigma) \end{cases}.$$

От горните дефиниции директно следва, че $\delta_L(q,\sigma) = \delta_L^C(q,\sigma)$, за $q \in Q_L$ и $\sigma \in \Sigma$.

За цялостното компресиране на дадената бимашина B прилагаме същата процедура върху функцията на преходите δ_R на десния автомат A_R и получаваме съответната компресирана функция δ_R^C.

4.5.2 Компресиране на Ψ-функция

За начало нека представим Ψ като функция Ψ^I с образ в естествените числа и списък W (речник) от всички функционални стойности на Ψ:

$$W = [w \mid w \in Range(\Psi)],$$
$$\Psi^I : Q_L \times \Sigma \times Q_R \to \mathbb{N}, \quad \Psi^I(q_l,\sigma,q_r) = IndexOf_W\big(\Psi(q_l,\sigma,q_r)\big).$$

Ще представим Ψ^I като списък от L на брой вектора $\vec{\Psi}_1, \vec{\Psi}_2, \ldots, \vec{\Psi}_L$ от множеството $\mathbb{N}^{RS}$. Като за $1 \le i \le L$ имаме:

$$\begin{aligned} \vec{\Psi}_i = \langle &\Psi^I(l_i,\sigma_1,r_1), \Psi^I(l_i,\sigma_2,r_1), \ldots, \Psi^I(l_i,\sigma_S,r_1), \ldots \\ &\Psi^I(l_i,\sigma_2,r_2), \Psi^I(l_i,\sigma_2,r_2), \ldots, \Psi^I(l_i,\sigma_S,r_2), \ldots \\ &\ldots \\ &\Psi^I(l_i,\sigma_1,r_R), \Psi^I(l_i,\sigma_2,r_R), \ldots, \Psi^I(l_i,\sigma_S,r_R)\rangle. \end{aligned}$$

Избираме $K' = \sqrt{L}$ и прилагаме горната процедура за клъстеризиране върху векторите $\vec{\Psi}_i$. Като резултат получаваме списък от центрове C':

$$C' = Cluster_{kmeans}(K', [\vec{\Psi}_i \mid 1 \le i \le L]).$$

Сега следва да конструираме частични функции φ'_i, съответстващи на всеки вектор $\vec{\Psi}_i$. Тези функции ще представят компонентите на $\vec{\Psi}_i$, които се различават по стойност от съответните компоненти на центъра на неговия клъстер $Cl(\vec{\Psi}_i, C')$. Т.е. за $1 \le i \le L$ имаме:

$$\varphi'_i = \{((\sigma,r),(\vec{\Psi}_i)_j) \mid \sigma \in \Sigma, r \in Q_R, j = (IndexOf_{Q_R}(r)-1)R + IndexOf_{\Sigma}(\sigma), \\ p = Cl(\vec{\Psi}_i, C') \,\&\, (p)_j \neq (\vec{\Psi}_i)_j\}.$$

Също така конструираме и функция P', представяща родителска връзка между векторите $\vec{\Psi}_i$ и центровете на съответните им клъстери:

$$P' : \mathbb{N} \to \mathbb{N},$$
$$P' = \{(i,j) \mid 1 \leq i \leq L, p = Cl(\vec{\Psi}_i, C'), j = IndexOf_{C'}(p)\}.$$

От дефиницията на гореспоменатите C', P' и $\varphi'_1, \varphi'_2, \ldots, \varphi'_L$ се вижда директно, че е в сила равенството:

$$\Psi^I(l,\sigma,r) = \begin{cases} \varphi'_i(\sigma,r), \text{ ако } !\varphi'_i(\sigma,r), \text{ където } i = IndexOf_{Q_L}(l) \\ (p)_j, \text{ иначе, където } j = (IndexOf_{Q_R}(r)-1)R + IndexOf_{\Sigma}(\sigma), \\ \qquad\qquad\qquad\qquad k = P'(i), p = (C')_k \end{cases}.$$

Така получаваме алтернативно (компресирано) представяне на функцията Ψ^I.

С цел постигане на по-добра компресия ще приложим метода за клъстеризиране на векторите от списъка C'. Тъй като броят им е по-малък, ще разделим всеки от тях на R нови вектора, представящи стойностите на Ψ^I при фиксирани l и r. Такова разделяне ни позволява по-добра грануларност за идентифициране на еднакви функционални стойности. Получаваме нов списък C^S от $K'R$ на брой S-мерни вектори:

$$C^S = [\langle (c)_i, (c)_{i+1}, \ldots, (c)_{i+S-1} \rangle \mid c \in C', 0 \leq j \leq R-1, i = Sj+1].$$

Сега избираме $K'' = \sqrt{|C^S|} = \sqrt{K'R}$ и прилагаме метода за клъстеризиране върху векторите, принадлежащи на C^S. Отново получаваме списък от центрове на клъстери $C'' = Cluster_{kmeans}(K'', C^S)$.

Аналогично с предната итерация конструираме функции φ'', отразяващи разликите в компонентите на векторите от C^S и центровете на съответните им клъстери. За $1 \leq i \leq K'S$ имаме:

$$\varphi''_i = \{(\sigma, (v)_j) \mid \sigma \in \Sigma, j = IndexOf_{\Sigma}(\sigma), v = (C^S)_i, p = Cl(v, C'') \,\&\, (p)_j \neq (v)_j\}.$$

Отново дефинираме функция P'', определяща родителската връзка между векторите от C^S и центровете на клъстерите им в C'':

$$P'' : \mathbb{N} \to \mathbb{N},$$
$$P'' = \{(i,j) \mid 1 \leq i \leq K'R, p = Cl\big((C^S)_i, C''\big), j = IndexOf_{C''}(p)\}$$

Вече сме готови да представим компресираната функция Ψ^I във финалния ѝ вид - Ψ^C:

$$\Psi^C : Q_L \times \Sigma \times Q_R \to \mathbb{N},$$

$$\Psi^C(l,\sigma,r) = \begin{cases} \varphi'_i(\sigma,r), \text{ ако } !\varphi'_i(\sigma,r), \text{ където } i = IndexOf_{Q_L}(l) \\ \varphi''_j(\sigma), \text{ ако } !\varphi''_j(\sigma), \text{ където } j = P'(i) + \\ \qquad\qquad IndexOf_{Q_R}(r) - 1 \\ (p)_s, \text{ иначе, където } s = IndexOf_{\Sigma}(\sigma), \\ \qquad\qquad m = P''(j), p = (C'')_m \end{cases}.$$

Директно се вижда, че $\Psi^I(l,\sigma,r) = \Psi^C(l,\sigma,r)$.

За получаване на крайните стойности на Ψ ползваме равенството:

$$\Psi(l,\sigma,r) = (W)_m, \quad m = \Psi^C(l,\sigma,r).$$

4.5.3 Техника за представяне на стойности на Ψ, запазващи идентитета

Много от практическите проблеми, решавани с използването на бимашини, се свеждат до намирането на определени думи от даден текст и заместването им с други. Останалите думи в текста остават непроменени. При бимашина, реализираща тотална функция от такъв вид, много от стойностите на нейната Ψ-функцията на изходите просто запазват стойността на втория си входен аргумент. Т.е. за много тройки $(q_l,\sigma,q_r) \in Q_L \times \Sigma \times Q_R$ имаме $\Psi(q_l,\sigma,q_r) = \sigma$.

Проведените експерименти показват, че в такива ситуации използването на проста техника показва добри резултати при компресирането на Ψ. В подготвителната фаза на гореописания метод, когато конструираме индексното представяне на функцията на изходите Ψ^I, избираме някое естествено число, което не е валиден индекс в списъка W, например 0. Сега нека дефинираме разширен вариант на индексната функция на изходите - Ψ^{IE}. Функционална стойност 0 ще означава, че в оригиналния си вид Ψ запазва стойността на втория си аргумент.

$$\Psi^{IE} : Q_L \times \Sigma \times Q_R \to \mathbb{N},$$

$$\Psi^{IE}(q_l,\sigma,q_r) = \begin{cases} 0, \text{ ако } \Psi(q_l,\sigma,q_r) = \sigma \\ IndexOf_W\big(\Psi(q_l,\sigma,q_r)\big), \text{ иначе} \end{cases}.$$

Прилагайки методът за компресиране върху Ψ^{IE}, получаваме нова компресирана функция Ψ^{CE}. Като за получаването на оригиналните

стойности на Ψ имаме предвид, че с нула сме кодирали идентитет на резултата с втория входен аргумент.

$$\Psi(q_l, \sigma, q_r) = \begin{cases} \sigma, \text{ ако } \Psi^{CE}(q_l, \sigma, q_r) = 0 \\ (W)_m, \text{ иначе, където } m = \Psi^{CE}(q_l, \sigma, q_r) \end{cases} .$$

В следствие на тази оптимизация често достигаме до ситуация, при която векторите от последното ниво C'' имат една стойност, повтаряща се за много компоненти. В този случай може да намалим ползваната памет за всеки вектор, като го представим чрез частична функция. Тя ще бъде дефинирана за индексите на компонентите с различна стойност от най-често срещаната, а самата най-срещана може да кодираме на индекс 0. Така за всеки вектор от C'' дефинираме съответна функция φ'''. Т.е. за $1 \leq i \leq |C''|$:

$$\begin{aligned} \varphi'''_i =& \{(j, (p)_j) \mid j \in \{1 \dots S\}, p = (C'')_i, m = Mf(p) \,\&\, (p)_j \neq m\} \cup \\ & \{(0, Mf((C'')_i))\}. \end{aligned}$$

Тогава ако $p = (C'')_i$, за да получим стойността на j-тия му компонент, използваме равенството:

$$(p)_j = \begin{cases} \varphi'''_i(j), \text{ ако } !\varphi'''_i(j) \\ \varphi'''_i(0), \text{ иначе} \end{cases} .$$

5 Емпирични резултати при прилагане на метода за компресия

В процеса на разработване на гореизложения метод бяха проведени експерименти, измерващи постигнатата компресия. За входни данни бяха използвани две бимашини с практическо приложение.

Първата е бимашина, чиято изходна функция представя така наречения стемер на Портър. За референция нека посочим [7]. Бимашината работи върху произволен английски текст и като резултат извежда съответен текст, в който всички думи са преведени в канонична форма, следвайки правилата за заместване, дефинирани от Портър.

Втората бимашина реализира лексикографски анализатор (лексер) за езикът C(M) - програмен език, представящ удобни абстрактни примитиви за работа с различни формални математически конструкции. Описание на синтаксиса и приложенията може да се намери в [2]. Изходната функция на бимашината приема текст, представляващ програма

на C(M), и извежда текста след прибавени анотации, разделящи го на поредица от токени (идентификатори, ключови думи, литерали, оператори и т.н.) Лексикално анотираният текст е необходим за последващото парсиране и компилиране до крайната изпълнима програма.

При провеждане на експериментите първоначално бе конструиран функционален преобразувател, представящ съответната функция на изхода. Бимашините бяха построени от преобразувателите, следвайки конструкцията описана в [2]. Като за имплементацията също бе използван езикът C(M).

За определяне на броя центрове при всяко едно клъстеризиране, участващо в метода за компресия, бяха изпробвани множество стойности близки до $\sqrt{N}$(, където с N бележим броя на входните вектори,) и бяха избрани тези, постигащи най-добри резултати.

Следващите таблици показват паметта, необходима за представяне на оригиналните бимашини и компресираните им алтернативи. Колоните съдържат съответно размера на азбуката на бимашината, броя състояния в левия автомат, броя състояния в десния автомат, броя на преходите в левия автомат, броя преходи в десния автомат, броя функционални стойности, представящи функцията на изходите, и размерът в килобайти на паметта, необходима за цялата бимашина.

Таблица 1 съдържа резултатите за стемера на Портът, където с BP означаваме началната бимашина, а с BP_C - компресираната версия на BP. Резултатите за C(M) лексера са представени в таблица 2. Тук с BL бележим входната бимашина. С BL_C компресираната ѝ версия.

Бимашина	$\|\Sigma\|$	$\|Q_L\|$	$\|Q_R\|$	$\|\delta_L\|$	$\|\delta_R\|$	$\|\Psi\|$	Размер в KB
BP	30	637	140	19 110	4 200	2 675 400	166 992
BP_C	30	637	140	2 903	512	26 321	788

Таблица 1: Представя резултатите от прилагането на метода за компресия върху бимашина, реализираща Портър стемер.

Бимашина	$\|\Sigma\|$	$\|Q_L\|$	$\|Q_R\|$	$\|\delta_L\|$	$\|\delta_R\|$	$\|\Psi\|$	Размер в KB
BL	166	222	370	36 852	61 420	13 635 240	1 026 800
BL_C	166	222	370	2 429	3 023	53 954	1 436

Таблица 2: Представя резултатите от прилагането на метода за компресия върху бимашина, реализираща C(M) лексер.

Както се вижда, методът за компресия довежда до намаляване на заеманата памет в размер от три порядъка.

6 Примерна имплементация

Тук ще представим примерна имплементация на гореизложения метод за компресиране, реализирана на езика C(M). Описание на езика, както и имплементации на основните алгоритми за строене на крайни автомати, преобразуватели и бимашини, могат да бъдат намерени в глава 9 от учебника [2].

Започваме с дефинициите на основните типове от данни, които ще използваме. За представяне на състояние $\mathcal{STATE}$ и символ $\mathcal{SYMBOL}$ използваме множеството на естествените числа. Дефинираме азбука $\mathcal{ALPHABET}$ като множество от символи, а дума $\mathcal{WORD}$ - като последователност от символи. Продължаваме с дефиниции на някои помощни функции, ползвани от основната процедура за клъстеризиране. Функцията H_d пресмята разстоянието на Хаминг между два вектора, представени тук като списъци от състояния/цели числа. Следва функция $mostFrequent$, намираща стойността на първия най-често срещан елемент във вектор, а функцията $squareRoot$ пресмята закръглена стойност на корен квадратен от цяло число. H_C получава за вход списък от вектори и намира техния Хаминг център, а функцията $assign$ работи върху списък от вектори и списък от центрове. За резултат връща по един списък за всеки център от C_l, съдържащ индексите на векторите от V_l, образуващи неговия клъстер.

$\mathcal{STATE}$ **is** $\mathbb{N}$;
$\mathcal{SYMBOL}$ **is** $\mathbb{N}$;
$\mathcal{ALPHABET}$ **is** $2^{\mathcal{SYMBOL}}$;
$\mathcal{WORD}$ **is** $\mathcal{SYMBOL}^*$;
$\mathrm{H_d} : \mathcal{STATE}^* \times \mathcal{STATE}^* \to \mathbb{N}$;
$\mathrm{H_d}(v_1, v_2) := \sum_{i \in \langle 1,\ldots,|v_1| \rangle} \begin{cases} 0 & \textbf{if } (v_1)_i = (v_2)_i \\ 1 & \textbf{otherwise} \end{cases}$;
mostFrequent : $\mathcal{STATE}^* \to \mathcal{STATE}$;
mostFrequent$(I) := (I)_m$, **where**
  $counts := \langle \sum_{i \in I} \begin{cases} 1 & \textbf{if } v = i \\ 0 & \textbf{otherwise} \end{cases} \mid v \in I \rangle$;
  $m := \#_{counts}(\max(counts))$;
  ;
squareRoot : $\mathbb{N} \to \mathbb{N}$;
squareRoot$(n) := m$, **where**
  $l \in \mathbb{R}$;
  $l := n$;
  $m := |\mathrm{lrint}(\mathrm{sqrt}(l))|$;

```
    ;
H_C : (𝒮𝒯𝒜𝒯ℰ*)* → 𝒮𝒯𝒜𝒯ℰ*;
H_C(V_l) := ⟨mostFrequent(⟨(v)_i | v ∈ V_l⟩) | i ∈ {1, ..., |(V_l)_1|}⟩;
assign : (𝒮𝒯𝒜𝒯ℰ*)* × (𝒮𝒯𝒜𝒯ℰ*)* → (ℕ*)*;
assign(V_l, C_l) := C, where
    C' := ⟨(i, #_d(min(d))) | i ∈ {1, ..., |V_l|}, d = ⟨H_d((V_l)_i, c) | c ∈ C_l⟩⟩;
    C := ⟨⟨i | (i, k) ∈ C' & k = j⟩ | j ∈ {1, ..., |C_l|}⟩;
    ;
```

Следва имплементация на метода за клъстеризиране на вектори. Функцията *initialSelect* избира началните K центъра от даденото множество V_l. Тя работи индуктивно, като за база конструира списък от първия вектор на V_l. На следващите итерации намира най-близкия център от вече избраните за всеки вектор от V_l. След това разширява списъка от центрове с елемента, който е най-отдалечен от най-близкия си център. Същинското клъстеризиране се реализира от функцията *kmeans*. Тя получава начален списък от вектори V_l и брой клъстери K и връща списък от центрове C и списъци A, определящи елементите във всеки клъстер. Също работи итеративно, като от начало започва със списък от центрове, пресметнат от *initialSelect*. На всяка итерация се пресмята разпределението на векторите към съответния им център. Итерациите след началната се стремят да избират нов център за всеки от формираните клъстери. Индукцията приключва, когато функцията на потенциала Ф спре да намалява.

```
initialSelect : (𝒮𝒯𝒜𝒯ℰ*)* × ℕ → (𝒮𝒯𝒜𝒯ℰ*)*;
initialSelect(V_l, K) := C, where
    C := induction
        step 0 :
            C^(0) := ⟨(V_l)_1⟩;
        step n + 1 :
            D := ⟨ min_{c∈C^(n)} H_d(v, c) | v ∈ V_l⟩;
            i := #_D(max(D));
            C^(n+1) := C^(n) · ⟨(V_l)_i⟩;
        until |C^(n)| = K
        ;
    ;
kmeans : (𝒮𝒯𝒜𝒯ℰ*)* × ℕ → (𝒮𝒯𝒜𝒯ℰ*)* × (ℕ*)*;
kmeans(V_l, K) := (C, A), where
    C ∈ (𝒮𝒯𝒜𝒯ℰ*)*;
    A ∈ (ℕ*)*;
```

41 $\Phi \in \mathbb{N}$;
42 $stop \in \mathbb{B}$;
43 $(C, A, \Phi, stop) :=$ **induction**
44 **step** 0 :
45 $C^{(0)} := \text{initialSelect}(V_l, K)$;
46 $A^{(0)} := \text{assign}(V_l, C^{(0)})$;
47 $\Phi^{(0)} := \sum_{j \in \{1,\ldots,\left|A^{(0)}\right|\}} \sum_{i \in (A^{(0)})_j} \mathrm{H_d}((V_l)_i, (C^{(0)})_j)$;
48 $stop^{(0)} := false$;
49 **step** $n+1$:
50 $C^{(n+1)} := \langle \begin{cases} \mathrm{H_C}(\langle (V_l)_i \mid i \in (A^{(n)})_j \rangle) & \textbf{if } (A^{(n)})_j \neq \varepsilon \\ (C^{(n)})_j & \textbf{otherwise} \end{cases} \mid j \in \{1, \ldots, \left|A^{(n)}\right|\} \rangle$;
51 $A^{(n+1)} := \text{assign}(V_l, C^{(n+1)})$;
52 $\Phi' := \sum_{j \in \{1,\ldots,\left|A^{(n+1)}\right|\}} \sum_{i \in (A^{(n+1)})_j} \mathrm{H_d}((V_l)_i, (C^{(n+1)})_j)$;
53 $stop^{(n+1)} := \Phi' = \Phi^{(n)}$;
54 $\Phi^{(n+1)} := \Phi'$;
55 **until** $stop^{(n)}$
56 ;
57 ;

Следващият сегмент съдържа дефиниция на процедурата за компресиране на функция на преходите на краен детерминиран автомат. В началото дефинираме типа, представящ самата функцията на преходите - $\mathcal{DTRANSITIONS}$, както и типа за представяне на целия детерминиран автомат - $\mathcal{DFSA}$. След това дефинираме нов тип данни за компресираното ѝ представяне - $\mathcal{CDTRANSITIONS}$. Той съдържа списък от частични функции, представящи върховете на дървото - $\mathcal{DTNODE}$, и функция, кодираща родителските връзки в дървото - $\mathcal{CTREE}$.

Функцията $compress_\delta$ реализира същинското компресиране. На входа получава краен детерминиран автомат от тип $\mathcal{DFSA}$. В началото трансформираме функцията на преходите на входния автомат в списък от вектори V_l. Извикваме процедурата за клъстеризиране върху този списък и получаваме списък от центрове C и разделяне на векторите в клъстери A. CL съпоставя всеки вектор към центъра на клъстера му. Следващата индукция конструира първото ниво от върхове в структурата, представено като списък от функции η'. В тях кодираме разликите в стойностите между всеки вектор и неговия център. За връх на дървото вземаме Хаминг центъра на векторите от C. Функциите от списъка η'' кодират разликите между тези вектори и върха. Накрая конкатенираме всички частични функции в списъка $Nodes$ и конструираме функцията

$Tree$ според съответните родителски връзки.

Функцията $calculate_\delta$ извлича стойността на преход по дадена компресирана репрезентация и двойка от състояние и символ.

```
58  DTRANSITIONS is STATE × SYMBOL → STATE;
59  DFSA is ALPHABET × 2^STATE × STATE × 2^STATE × DTRANSITIONS;
60  DTNODE is SYMBOL → STATE;
61  CTREE is ℕ → ℕ;
62  CDTRANSITIONS is DTNODE* × CTREE;
63  compress_δ : DFSA → CDTRANSITIONS;
64  compress_δ(Σ, Q, s, F, Δ) := (Nodes, Tree), where
65      Σ_l := Σ;
66      Q_l := ⟨1, ..., |Q|⟩;
67      V_l := ⟨⟨Δ(q, σ) | σ ∈ Σ_l⟩ | q ∈ Q_l⟩;
68      C ∈ (STATE*)*;
69      A ∈ (ℕ*)*;
70      (C, A) := kmeans(V_l, squareRoot(|V_l|));
71      CL : ℕ → ℕ;
72      CL := {(q, i) | i ∈ {1, ..., |A|}, q ∈ (A)_i};
73      η' ∈ DTNODE*;
74      η' := ⟨{((Σ_l)_s, (u)_s) | s ∈ {1, ..., |Σ_l|}, u = (V_l)_n, p = (C)_CL(n) & (u)_s ≠ (p)_s} | n ∈
        {1, ..., |V_l|}⟩;
75      Root := H_C(C);
76      η'' ∈ DTNODE*;
77      η'' := ⟨{((Σ_l)_s, ((C)_i)_s) | s ∈ {1, ..., |(C)_i|} & ((C)_i)_s ≠ (Root)_s} | i ∈ {1, ..., |C|}⟩;
78      r ∈ DTNODE;
79      r := {((Σ_l)_s, (Root)_s) | s ∈ {1, ..., |Σ_l|}};
80      Nodes := η' · η'' · ⟨r⟩;
81      Tree := {(i, |V_l| + CL(i)) | i ∈ {1, ..., |V_l|}};
82      ;
83  calculate_δ : CDTRANSITIONS × STATE × SYMBOL → STATE;
84  calculate_δ((NL, T), q, σ) := η(σ), where
85      p := T(q);
86      gp := |NL| ;
87      η := { (NL)_q   if !((NL)_q)(σ)
             { (NL)_p   if !((NL)_p)(σ)   ;
             { (NL)_gp  otherwise
88      ;
```

В тази секция е имплементацията на метода, компресиращ функцията на изходите на бимашина. Функцията е представена с типа $BMOUTPUT$.

За цялата бимашина използваме $\mathcal{BM}$. Типа от данни $\mathcal{CBMOUTPUT}$ дефинира компресирания вид на функцията. Съдържа списък от частични функции за върхове от първо ниво - $CL1NODE$, списък от функции за върхове от второ ниво - $CL2NODE$, две функции, представящи йерархичните връзки, и списък с всички думи от образа на входната функция.

Компресията на Ψ е дефинирана във функцията $compress_\Psi$. Списъка W съдържа всички думи от образа на Ψ, а V_l е списък от вектори, представящи Ψ със стойности - индекси в W. След извършване на първоначалното клъстеризиране получаваме списък от центрове C'. Първата индукция конструира функции φ', които кодират разликите между векторите от V_l и центровете C'. На следващата стъпка разбиваме векторите от C' по стойностите за фиксиран символ и получаваме списък $V2_l$. Клъстеризираме и него и получаваме нов списък от центрове C''. Функциите φ'' отразяват разликите в стойностите между векторите от $V2_l$ и съответните им центрове от C''. Функциите φ''' кодират стойностите на върховете от последното ниво, като изпускат повторение на най-често срещаната стойност. Накрая конструираме списъци от върхове от първо и второ ниво - $L1Nodes$ и $L2Nodes$, и функции за родителските връзки - $L1Parents$ и $L2Parents$.

Функцията $calculate_\Psi$ декодира стойностите на Ψ-функция в компресиран вид.

89 $\mathcal{BMOUTPUT}$ **is** $\mathcal{STATE} \times \mathcal{SYMBOL} \times \mathcal{STATE} \to \mathcal{WORD}$;
90 $\mathcal{BM}$ **is** $\mathcal{DFSA} \times \mathcal{DFSA} \times \mathcal{BMOUTPUT}$;
91 $\mathcal{RANGEOUTPUT}$ **is** $\mathcal{WORD}^*$;
92 $CL2NODE$ **is** $\mathcal{SYMBOL} \to \mathbb{N}$;
93 $CL1NODE$ **is** $\mathcal{SYMBOL} \times \mathcal{STATE} \to \mathbb{N}$;
94 $\mathcal{CBMOUTPUT}$ **is** $CL1NODE^* \times CL2NODE^* \times \mathcal{CTREE} \times \mathcal{CTREE} \times \mathcal{RANGEOUTPUT}$;
95 $\text{compress}_\Psi : \mathcal{BM} \to \mathcal{CBMOUTPUT}$;
96 $\text{compress}_\Psi(L, R, \Psi) := (L1Nodes, L2Nodes, L1Parents, L2Parents, W)$, **where**
97 $(\Sigma, Q_L, s_L, F_L, \Delta_L) := L$;
98 $(\Sigma', Q_R, s_R, F_R, \Delta_R) := R$;
99 $\Sigma_l := \Sigma$;
100 $QL_l := \langle q \mid q \in \{1, \ldots, |Q_L|\}\rangle$;
101 $QR_l := \langle q \mid q \in \{1, \ldots, |Q_R|\}\rangle$;
102 $W := \text{Proj}_4(\Psi)$;
103 $V_l \in (\mathbb{N}^*)^*$;
104 $V_l := \langle\langle \begin{cases} 0 & \textbf{if } \Psi(q,\sigma,r) = \langle\sigma\rangle \\ \#_W(\Psi(q,\sigma,r)) & \textbf{otherwise} \end{cases} \mid r \in QR_l, \sigma \in \Sigma_l\rangle \mid q \in QL_l\rangle$;
105 $C' \in (\mathcal{STATE}^*)^*$;

106 $A' \in (\mathbb{N}^*)^*$;
107 $(C', A') := \text{kmeans}(V_l, \text{squareRoot}(|V_l|))$;
108 $\text{CL}' : \mathbb{N} \to \mathbb{N}$;
109 $CL' := \{(q, i) \mid i \in \{1, \ldots, |A'|\}, q \in (A')_i\}$;
110 $\varphi' \in CL1NODE^*$;
111 $\varphi' := \langle\{(((\Sigma_l)_s, (QR_l)_r), (u)_{(r-1)\times|\Sigma_l|+s}) \mid r \in \{1, \ldots, |QR_l|\}, s \in \{1, \ldots, |\Sigma_l|\}, u = (V_l)_n, p = (C')_{\text{CL}'(n)} \;\&\; (u)_{(r-1)\times|\Sigma_l|+s} \neq (p)_{(r-1)\times|\Sigma_l|+s}\} \mid n \in \{1, \ldots, |V_l|\}\rangle$;
112 $P' \in \mathcal{CTRE\mathcal{E}}$;
113 $P' := \{(i, \text{CL}'(i)) \mid i \in \{1, \ldots, |V_l|\}\}$;
114 $V2_l \in (\mathbb{N}^*)^*$;
115 $V2_l := \bigodot_{c \in C'} \langle(c)_{(i-1)\times|\Sigma_l|+1,\ldots,i\times|\Sigma_l|} \mid i \in \{1, \ldots, |QR_l|\}\rangle$;
116 $C'' \in (\mathcal{STATE}^*)^*$;
117 $A'' \in (\mathbb{N}^*)^*$;
118 $(C'', A'') := \text{kmeans}(V2_l, \text{squareRoot}(|V2_l|))$;
119 $\text{CL}'' : \mathbb{N} \to \mathbb{N}$;
120 $CL'' := \{(q, i) \mid i \in \{1, \ldots, |A''|\}, q \in (A'')_i\}$;
121 $\varphi'' \in CL2NODE^*$;
122 $\varphi'' := \langle\{((\Sigma_l)_s, (u)_s) \mid s \in \{1, \ldots, |\Sigma_l|\}, u = (V2_l)_n, p = (C'')_{\text{CL}''(n)} \;\&\; (u)_s \neq (p)_s\} \mid n \in \{1, \ldots, |V2_l|\}\rangle$;
123 $MFs := \langle\text{mostFrequent}((C'')_i) \mid i \in \{1, \ldots, |C''|\}\rangle$;
124 $\varphi''' \in CL2NODE^*$;
125 $\varphi''' := \langle\{((\Sigma_l)_s, ((C'')_i)_s) \mid s \in \{1, \ldots, |(C'')_i|\} \;\&\; ((C'')_i)_s \neq (MFs)_i\} \cup \{(0, (MFs)_i)\} \mid i \in \{1, \ldots, |C''|\}\rangle$;
126 $L1Nodes := \varphi'$;
127 $L1Parents := P'$;
128 $L2Nodes := \varphi'' \cdot \varphi'''$;
129 $L2Parents := \{(i, |V2_l| + \text{CL}''(i)) \mid i \in \{1, \ldots, |V2_l|\}\}$;
130 ;
131 $\text{calculate}_\Psi : \mathbb{N} \times \mathcal{CBMOUTPUT} \times \mathcal{STATE} \times \mathcal{SYMBOL} \times \mathcal{STATE} \to \mathcal{WORD}$;
132 $\text{calculate}_\Psi(QR_s, (L1N, L2N, P1, P2, W), q, \sigma, r) := \begin{cases} \langle\sigma\rangle & \textbf{if } w = 0 \\ (W)_w & \textbf{otherwise} \end{cases}$, **where**
133 $p := (\text{P1}(q) - 1) \times QR_s + r$;
134 $gp := \text{P2}(p)$;
135 $w := \begin{cases} ((L1N)_q)(\sigma, r) & \textbf{if } !((L1N)_q)(\sigma, r) \\ ((L2N)_p)(\sigma) & \textbf{if } !((L2N)_p)(\sigma) \\ ((L2N)_{gp})(\sigma) & \textbf{if } !((L2N)_{gp})(\sigma) \\ ((L2N)_{gp})(0) & \textbf{otherwise} \end{cases}$;
136 ;

7 Заключение

В настоящата работа бе изложен метод за компресирано представяне на бимашина. По-конкретно показахме компресирано представяне на функция на преходите на краен детерминиран автомат и на функция на изходите на бимашина.

Същината на идеята е да бъдат идентифицирани връзки между елементите от първообраза на компресираната функцията, определящи еднакви функционални стойности. За целта представихме функцията като съвкупност от вектори и ги групирахме в клъстери, избирайки център на всеки клъстер, така че да минимизираме сумата от Хаминг разстоянията между векторите и съответните им центрове. Това позволява кодирането на множество повтарящи се стойности на единствено място.

Освен за функция на преходите и функция на изходите същата основна идея може да бъде приложена и за компресирано представяне на други функции, при които броя на елементите в образа е поне линейно помалък от броя на елементите в първообраза. Разбира се, всяка функция има различна структура на зависимостите между стойностите си. Това предполага, че крайният резултат от компресията до голяма степен ще зависи от начина, по който функцията бъде представена като множество от вектори, и от общия брой равни компоненти между тях.

Накрая представихме примерна имплементация на алгоритъма за компресия и показахме резултатите от неговото изпълнение върху използвани в практиката бимашини, демонстрирайки намаляване на използваната памет в размер от три порядъка.

Литература

[1] Stoyan Mihov, Klaus U. Schulz: *Efficient dictionary-based Text rewriting using subsequential transducers*, Natural Language Engineering, Volume 13, Issue 04, pp 353-381, December 2007.

[2] Stoyan Mihov and Klaus U. Schulz: *Finite-State Techniques, Automata, Transducers and Bimachines*, Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 2019.

[3] Stefan Gerdjikov, Stoyan Mihov, Klaus U. Schulz: *Space-Efficient Bimachine Construction Based on the Equalizer Accumulation Principle*, Theoretical Computer Science, February 2018.

[4] R.E. Tarjan, A.C.C. Yao: *Storing a Sparse Table*, Communications of the ACM, 22(11), 606-611, 1979.

[5] Dan Gusfield: *Algorithms on Strings, Trees, and Sequences*, Computer Science and Computational Biology, Cambridge University Press, 1997.

[6] Shai Shalev-Shwartz, Shai Ben-David: *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, 2014.

[7] Martin Porter: *An algorithm for suffix stripping*, Program, 14 no. 3, pp 130-137, July 1980.
`https://tartarus.org/martin/PorterStemmer/`