Improving the performance of the puzzle "Escape from Zurg" by applying pruning methods

Francisco Noya
fnoya2@illinois.edu

Abstract

Reducing the search space is critical for search programs that have to deal with spaces that tend to grow exponentially. Functional programming can produce elegant algorithms that can benefit from features of modern functional languages such as lazy data structures. Using the "Escape from Zurg" puzzle, I converted the original exhaustive search algorithm into a more efficient version by means of two pruning methods based on a lazy data structure called *improving sequence*.

After applying these techniques the running time for finding the optimal solution was slashed by over 20 times and the amount of memory used was reduced by 95%.

I. Overview

The Haskell implementation presented in the original "Escape from Zurg" paper ([2]) constructs an exhaustive search space with all possible configurations of the elements of the problem. This is done by extending the search tree with a function that generates every alternative from each intermediate node. Each node (or state) is paired with the list of moves that leads to it. The state itself is represented by a tuple containing the position of the flashlight (L or R) and the elements (toys) that are still on the left side of the bridge (where Zurg is).

The construction of the search space takes advantage of Haskell list comprehension and lazy evaluation features to terminate without an explicit terminating condition. This space is subsequently filtered with the isSolution predicate to select those paths that lead to the goal within the given constraint ($duration \leq 60$).

This approach of solving the problem is clear and concise but it is also very inefficient. For example, with the original implementation and setup 108 paths are generated and evaluated but only two of them satisfy the duration constraint. To circumvent this problem, I decided to include a technique that would early prune those paths that will not lead to the goal because their partial durations are already over the threshold.

The objective of search problems of this kind is to find the least cost to reach the goal from an initial state. A common characteristic of these problems is that the cost function grows monotonically along the depth of the search tree. This property can be use for pruning.

Improving sequences are lazy data structures that consist of a monotonical sequence of approximation values that are gradually improved on the basis of some ordering relation as they approach the final value ([3]). These approximating values are usually generated during the computations but rarely used. For example, to obtain the length of an array, the function uses an accumulator that gradually approaches the final result. If the intent of the programmer was to evaluate the expression 1 <length [1..100] the length will inefficiently iterate one hundred times before returning the value 100. The idea behind *improving sequences* is that if an intermediate value of some expression has sufficient information to yield the result of an outer expression, further computations that generate unnecessary values are pruned.

I applied this idea to the "Escape from Zurg" problem to prune the branches of the search tree whose intermediate durations were already over a threshold. In fact, I adapted the algorithm to find the paths of minimum duration or cost that solve the problem. The implementation was based on Morimoto et al. ([4]) and is summarized in the following section.

II. Implementation

Improving sequences are defined as the following sum data type in Haskell:

```
data IS a = a :? IS a | E deriving (Eq, Show)
```

The sequence consists of gradually improving values. The sequence could be infinite but, if finite, E represents the terminal symbol that indicates that the current value cannot be further improved. For example, 1 :? 2 :? 3 :? E denotes a sequence in which 3 is the most improved value.

For example, a redefined length function that makes use of an improving sequence will have the following definition.

```
length :: [a] -> a -> IS a
length [] n = n :? E
length (x:xs) n = n :? IS.length xs (n+1)
```

Special binary relations are also needed to make the IS type useful.

```
(.<) :: Ord a => a -> IS a -> Bool
n .< E = False
n .< (x :? xs) = (n < x) || (n .< xs)
```

The advantages in efficiency of IS are obvious when measuring the time and memory needed to compare the length of a large list against 1.

```
*Zurg> s=[1..10000000000]

*Zurg> 1 < Prelude.length s
True
(162.96 secs, 720,000,052,040 bytes)

*Zurg> 1 .< IS.length s 0
True
(0.01 secs, 53,272 bytes)
```

Before applying IS to the "Escape from Zurg" puzzle I defined a new search function whose purpose is to find the shortest time or minimum duration to achieve the goal. Without using IS this function has the following definition.

```
search :: ([m],s) -> Int
search (ms,state)
         | isGoal (ms,state) = cost (ms,state)
         | otherwise = minimum [search (moves : ms,ss) | (moves,ss) <- trans state ]
```

The cost function is just the cumulative duration of the moves required to attain the given state. The isGoal function is just the result of comparing the given state with the desired final state (R,[]).

To use IS in this problem, we first need to define a new minimum function based on IS.

```
minimum :: Ord a => [IS a] -> IS a
minimum = foldl1 minB

minB :: Ord a => IS a -> IS a -> IS a
minB E ys = E
minB xs E = E
minB xxs@(x :? xs) yys@(y :? ys)
     | x == y = x :? minB xs ys
     | x < y = x :? minB xs yys
     | otherwise = y :? minB xxs ys
```

The new searchIS function based on IS has the following form.

```
searchIS :: ([m],s) -> IS Int
searchIS (ms,state)
         | isGoal (ms,state)= cost (ms,state) :? E
         | otherwise = cost (ms,state) :? IS.minimum [searchIS (moves : ms, ss) |
             (moves,ss) <- trans state]
```

When evaluating the minimum cost between two search tree branches, the computations are pruned when the end of the IS with the minimal value is reached. States further along a branch that has an intermediate cost already higher than the final value of another path are not even explored saving time and resources. This turns out to be an implementation of Dijkstra's single-source shortest-path search algorithm ([1]).

However, in this implementation computation are only terminated when the end of an IS is reached. However, it could be possible that the intermediate value of the branch is already higher than the current global minimum. In this case, efficiency would be gained if the exploration of that branch could be terminated early. By introducing an upper bound to the search function and rewriting the minimum function as a continuation, we could achieve this objective.

```
searchD :: ([m],s) -> IS Int
searchD (ms, state) = searchD' (ms,state) (maxBound :? E)
  where
    searchD' (ms,state) u
             | isGoal (ms,state) = cost (ms,state) :? E
             | otherwise = cost (ms,state) :? IS.minimumD u [searchD' (moves : ms, ss) |
                 (moves,ss) <- trans state]

minimumD :: Ord a => IS a -> [IS a -> IS a] -> IS a
```

```
minimumD = foldl dfbb

dfbb :: Ord a => IS a -> (IS a -> IS a) -> IS a
dfbb u f = finalize (minB u (f u))

finalize :: Ord a => IS a -> IS a
finalize (x :? E) = x :? E
finalize (x :? xs) = finalize xs
```

The finalize function just returns the most improved value of the IS. The global minimum is kept as a short IS of the form x :? E where x is the minimum and as such is fed into minB.

This is an implementation of a Depth-First Branch-and-Bound search algorithm.

III. Tests

The three search algorithms (search, searchIS, and searchD) were applied to solve the original puzzle with 4 toys as well as more difficult versions with 5 to 8 toys. The following table summarizes the definitions for each toy: name and cost (the time that it takes to cross the bridge).

TABLE I
DEFINITION OF TOYS USED IN THE EXPERIMENTS

Toy name	Time to cross
Buzz	5
Woody	10
Rex	20
Hamm	25
T1	11
T2	13
T3	14
T4	15

The time and memory consumed on each run for each algorithm are presented in the following table.

TABLE II
TIME AND MEMORY CONSUMED ON EACH RUN

No. of toys	search	searchIS	searchD
4	0.02s 0.6 Mb	0.01s 0.5 Mb	0.01s 0.5 Mb
5	0.16s 24 Mb	0.08s 12 Mb	0.05s 12 Mb
6	7.8s 2 Gb	1.6s 0.4 Gb	1.3s 0.4 Gb
7	1011s 288 Gb	115s 22 Gb	60s 23 Gb
8	ND	ND	5874s 1.582 Gb

ND: Not done.

IV. Listing

Diabetes mellitus comprises various medical conditions that are associated with hyperglycemia and it can be caused by an abnormal secretion of insulin in the body system ([**?**]). The number of diabetic patients worldwide has soared in the last 20 years ([**?**]). This increment has led to various studies on the efficient management of this condition. Diabetes has contributed to a 75% increase in mortality rate: a diabetic adult is at higher risk of death from other diseases compared to a non-diabetic adult in the US ([**?**]). Therefore, an efficient and timely interventional clinical decision support system that would aid in preventing death or predicting mortality would lead to a major milestone in diabetes management research. Anand et al., ([**?**]) stated that having diabetes could affect the treatment of ICU patients as unique factors have to be taken into consideration, for example, glucose levels. Diabetic patients account for more than 45% of ICU stays and consume more resources compared to patients suffering from other chronic diseases ([**?**]).

Our research study aims to build a multi-network deep learning model to predict the mortality of ICU diabetic patients using the clinical notes, and features from the Apache II scoring system plus features such as glycosylated hemoglobin (HbA1c), serum creatinine, and glucose levels, that can improve the predictability for diabetic patients ([**?**]).

V. Previous Work

Research in diabetes is broad and different studies have engaged machine learning algorithms to improve the treatment and management of this global metabolic disorder. We streamlined past works to those that interest us most, which were done in the past five years and targeted a prediction task. Anand et al., ([**?**]) employed binomial logistic regression to predict the risk mortality of ICU diabetic patients. Their research was based on the certainty that these variables (HbA1c, mean glucose during stay, serum creatine, diagnoses upon admission, and type of admission) were the major features needed for mortality prediction. Their research used the data from the MIMIC-III database and their model achieved ROC AUC value of 0.787.

Ye, Yao, Shen, Janarthanam & Luo ([**?**]) employed different machine learning algorithms and knowledge-guided feature extraction to predict mortality in critically ill diabetic patients. Knowledge-guided CNN using CUI (UMLS concept unique identifiers) plus word embedding and CNN using word embedding were applied to clinical notes to predict mortality in diabetic ICU patients. They also ran different machine learning models such as Logistic Regression, Random Forest, XGBoost, Gradient Boosting, Deep Learning ANN, and Majority Voting with Majority Voting model taking the lead with an AUC of 0.87. These machine learning algorithms were used with structured EHR data to predict mortality risk in ICU diabetic patients. CNN with word embedding performed best overall with ROC AUC of 0.97.

Yang, Kuang & Xia ([**?**]) proposed a multimodal deep learning neural network, which uses time series data and clinical notes to predict mortality of ICU patients, and obtained an ROC AUC of 0.861. Che, Purushotham, Cho, Sontag, & Liu ([**?**]) built a model GRU_D based on Gated Recurrent Unit (GRU), which could handle missing data, for mortality prediction using the time series features including input events, output events, lab events, and prescription events from the *MIMIC III* dataset. Choi, Bahadori, Kulas, Schuetz, Stewart and Sun ([**?**]) explored the use of multi RNNs with multi attention at different levels of granularity. We are going to apply a similar mechanism on each of the different networks, notes and events.

VI. Approach and implementation

A. Problem formulation

The EHR information of a patient that is admitted into ICU can be represented as a timeseries of multiclass, multivariate observations, in our case, we will pay attention to two classes, chart/lab events and clinical notes. Considering v_i a multi-hot vector of the values for the different types of events observed in the date i, and n_i an aggregate notes embedding for all the notes registered in the date i. Being T_p the number of dates with either events or notes registered for the patient p. At each date i, we can represent the patient p with the tuple $[v_i^{(p)}, n_i^{(p)}]$ where $i = 1 \dots T_p$.

The goal of our network is to predict the labels $y(p) \in \{0, 1\}$ where 0 means the patient is predicted to be alive 48 hrs after T_p, and 1 is predicted to be deceased.

B. Data Source

For our project, we used the data from MIMIC-III database, because this is a very complete source with a lot of good real-life records of patients who have been admitted to ICU.

C. Cohort Selection

First, we defined our cohort as patients who have a diabetic diagnosis, that is, ICD-9 codes containing the word "diabetes" exempting codes 3572, V771, V180, V1221. We considered all the data of each patient from the moment they were first admitted, all the way to 48 hrs before last discharge. Some patients had multiple admissions, so we chose the observation window to be the earliest admission time to 48 hrs before the last discharge time. Any patient who did not have any data 48 hrs before final discharge was removed from the analysis. We used the presence of "deathtime" in the Admissions table to create a mortality flag: zero for alive and one for dead. The summary statistics of the cohort is depicted in the Table III.

TABLE III
Cohort statistics

Number of Cohort Patients	9822
Maximum Length of Stay	116 days
Minimum Length of Stay	<1 day

The statistics of the Clinical Notes are shown in Table IV.

The statistics of the Clinical / Lab events we used for the Events Network are shown in Table V.

TABLE IV
CLINICAL NOTES STATISTICS

Min # of notes per patient	0
Max # of notes per patient	505
Avg. # of notes per patient	13

TABLE V
STATISTICS OF SELECTED FEATURES FROM CHART EVENTS AND LAB EVENTS.

	Characteristics				
	Count	Avg	Std Dev	Min	Max
APACHE II Features					
Diastolic blood pressure	7038135	56	220	-40	114108
Fraction inspired oxygen	3489	1.26	0.91	0	10
Glascow coma scale eye opening	1521073	3.28	1.06	1	4
Glascow coma scale motor response	565486	5.29	1.39	1	6
Glascow coma scale total	945427	11.43	3.74	3	15
Glascow coma scale verbal response	565912	3.09	1.89	1	5
Glucose	1570663	140	1143	-124	999999
Heart Rate	7941588	102	3548	-88	9999999
Height	12015	168.75	15.23	0	445
Mean blood pressure	6363937	78	100	-135	120130
Oxygen saturation	173361	88	22	0.8	6363
Respiratory rate	7439791	19	863	-1	2355555
Systolic blood pressure	6386370	119	125	-69	141146
Temperature	129290	37	1	0	43
Weight	1628937	84.63	24.61	0	300
pH	530657	7.38	0.09	0	7.99
Diabetes related Features					
HbA1C	16619	6.76	1.66	0.45	22
Blood Glucose	749156	131	66	0	3565
Serum Creatinine	797231	1.56	1.89	0	808

D. Data Processing

The *MIMIC-III* database is available for access via AWS without need to download. We created a cohort master table (diabetes_patient_cohort), which we later used to create the source of information for each of our networks, notes and events. Using SQL queries, we created tables such as:

- diabetic_patients_notes_agg contains the notes of a patient aggregated by distinct date by concatenating the notes for that date if there are more than one.
- diabetic_patients_events contains the actual values for the different laboratory tests, vital signs and other values from ICU charts that we selected as features. The data is aggregated by patient and date. If multiple readings were obtained for the same feature of a patient on the same date, the daily average of it was used.

We split the diabetic_patient_cohort into train and evaluation sets using a split ratio of 80% to 20%. We called this our "unbalanced" cohort. We observed that the data has an inherent class imbalance. To resolve this issue, we oversampled the minority class in the training set to create a "balanced" cohort. All these actions were done using the sklearn library, class resample and stored on S3 BUCKET to later load onto AWS ATHENA tables: (train_cohort and test_cohort).

E. Clinical Notes Embeddings

We formed a notes table for the diabetic patients in the ICU by aggregating the notes for a given date. Then, we preprocessed the notes and trained our own word embedding model using the Word2Vec class from the gensim.models package. Once we had our embedding model, we processed each *i-th* distinct date aggregated notes for every patient p to obtain $n_i^{(p)}$ by calculating an embedding vector for each day notes. We achieved this by obtaining the embedding vector for each word and then by adding the embeddings for all the words of the notes of the date *i*.

F. Notes Network

The Notes Network architecture consists of taking the clinical notes embedding described on the previous section and then feed it into two parallel RNN (GRUs) layers, to calculate on each its attention. On one of them we calculate the attention for each distinct date (α), and on the second one, we calculate the attention of each embedding component (β), followed by a FCL layer, and a sigmoid activation function, to output the embedding vectors for this network.

TABLE VI
MODEL METRICS.

Notes Network	Events Network	Dataset	Time	LR	Epochs	Prec	Recall	F1	ROC AUC
RNN	RNN	Unbalanced	286.65	0.001	10	0.6269	0.367	0.463	0.871
RNN	RNN	Balanced	566.94	0.00001	10	0.1780	0.948	0.299	0.765
RNN	α,β Attn	Unbalanced	318.50	0.001	10	0.6902	0.516	0.591	0.879
RNN	α,β Attn	Balanced	629.93	0.00001	10	0.3912	0.838	0.533	0.896
RNN	α,β Attn	Unbalanced	279.23	0.01	10	0.7280	0.292	0.417	0.845
RNN	α,β Attn	Balanced	610.52	0.0001	10	0.8470	0.876	0.861	0.974
α Attn	α,β Attn	Unbalanced	400.16	0.001	10	0.7380	0.397	0.516	0.879
α Attn	α,β Attn	Balanced	741.47	0.0001	10	0.8770	0.884	0.880	0.976
α,β Attn	α,β Attn	Unbalanced	431.99	0.001	10	0.6608	0.521	0.582	0.908
α,β Attn	α,β Attn	Unbalanced	434.23	0.0001	10	0.6114	0.493	0.546	0.893
α,β Attn	α,β Attn	Balanced	852.77	0.0001	10	0.8790	0.868	0.873	0.977

G. Events Network

The multi-hot vectors $v_i^{(p)}$ were used as inputs for the Events Network. The architecture of this network comprises another pair of parallel RNNs (GRUs) with alpha (α) and beta (β) attention, where α means the attention for each distinct event date, and β means the attention for the different features. The network finally has a FCL and a sigmoid activation function to produce the embeddings vector of the events of each patient.

H. Notes-Events Network

The final Model takes the output of the two previous Networks (embedding vectors output of Notes and Events network were of size 128 each), concatenates them and applies two additional FCL layers with a dropout in the middle and a final sigmoid activation function to generate the final prediction.

I. Normalization of attention weights

Before extracting interpretable information from the attention weights of the events codes, the raw weights must be normalized to make them comparable with each other. The normalization was done with the following transformation:

$$\bar{\beta}_i = \frac{\sum_b \bar{e_i}^{(b)} |\beta_i^{(b)}|}{B} \tag{1}$$

where β_i is the normalized weight for event code i, $\bar{e_i}^{(b)}$ is the average of the values of event code i in the batch b, $\bar{\beta_i}^{(b)}$ is the average of attention weights for event code i in the batch b, and B is the number of batches. For this calculation the batch size was set to 1,000 patients.

J. Experimental Setup

The model was implemented in Pytorch 1.7.1 and trained initially in AWS Sagemaker environment with a memory of 8GBs. Eventually we moved the training to a more powerful machine with the same Pytorch version, but with 32GBs of RAM and an NVIDIA GeForce GTX 1060 with 6GB GPU. This enabled us to train the model and do fine tuning with much faster training times compared to only using the CPU environment.

VII. Experimental Results

A. Model assessment

Table VI shows the results of our model as we progressed in the construction and experimentation (always 10 epochs).

B. Ranking of risk factors

One of the advantages of attention mechanisms is the ability to produce interpretable models. In our case, we can take advantage of the attention weights to identify which of the clinical and paraclinical observations from chart and lab events has a bigger impact on the outcome of the patient.

This is in line with the accepted view that high plasma glucose levels, high body mass index (BMI) and high blood pressure are risk factors commonly found together ([**?**]).

To confirm that the most relevant features carry the information with most predictive power, we trained the model with just the eight features that received the most attention: blood glucose (lab event), glucose (chart event), weight, systolic blood pressure, temperature, heart rate, mean blood pressure, and oxygen saturation. After evaluation, these "essential" features' model gave a precision of 0.622, recall of 0.390, f1 score of 0.873 and ROC AUC of 0.867. The most dramatic difference between the full and the essential features model was on the recall (0.390 vs. 0.521) 21% lower while the separation power (AUC ROC) was only 3% lower (0.867 vs. 0.893) confirming that most of the information relevant for model performance was retained in the smaller features subset.

TABLE VII
CONTRIBUTIONS OF EACH NETWORK MODULE ON MODEL METRICS.

Notes Network	Events Network	Dataset	Prec	Recall	F1	ROC AUC
—	α,β Attn	Unbalanced	0.6033	0.4644	0.525	0.850609
α,β Attn	—	Unbalanced	0.625	0.0199	0.039	0.78265
—	α,β Attn	Balanced	0.7319	0.8632	0.792	0.960047
α,β Attn	—	Balanced	0.7026	0.8077	0.751	0.937759

C. Contribution of each network module to the predictive power of the model.

Next, we wanted to determine how much of the performance of the model comes from the clinical notes and how much comes from the sequences of chart and lab events.

To determine this, we trained each network by itself and evaluated against our test sets, the results of the evaluations are shown in Table VII.

These results show that the clinical notes' main contribution is on improving the general performance of the full model, but with a bigger improvement on the recall of the model or reducing the number of false negatives. On the other hand, the main contribution of the lab and charts events is of course improving the overall performance; however, it is the precision that seems to be the one that increases the most, or, namely, reducing the number of false positives.

Overall, the ability of the model to distinguish between classes, measured by AUC ROC, was reduced by 1.7% when the clinical notes were ignored, and by 4% when the lab and chart features were ignored.

VIII. DISCUSSION

We analyzed the clinical notes of diabetic patients including the lab and other clinical features collected from the MIMICIII database and fed the data into parallel RNNs to predict mortality. We had originally attempted to use CNN on top of the Clinical Notes data, but after testing this, we switched to an RNN model as this makes easier the processing of the notes, and leads to a much faster training time.

Our full network yielded good results compared to previous works, achieving a maximum AUC ROC of 0.977. We originally had observed a trade-off between the reported recall and precision when using the original training set, versus when using a balanced training set, but once we updated our models to include α and β attention on both the Events and Notes network, the balanced cohort clearly yielded much better results, validating the idea of using a re-sampling algorithm to avoid our model from giving more weight to the majority class.

By ablating either one of the parallel RNN networks we were able to quantify the contribution that each module had in the outcome. With this approach we were able to determine that the events module captured most of the predictive power of the model (Table VII). However, the notes module helped in improving its precision or reducing the number of false positives. The clinical notes capture the medical insight that is not reflected in the objective chart and lab data. One can speculate that this insight discriminates those patients that have good chances of surviving despite their poor looking chart and lab events.

The addition of attention mechanisms analogous to RETAIN ([**?**]) produced a significant improvement over the naïve RNNs (Table VI). The ROC AUC improved from 0.765 to 0.896 after the addition of the two-level attention mechanism in the events module, and to 0.977 after implementing attention also in the notes' module. The magnitude of these improvements was striking, being much higher than those reported in the original publication ([**?**]). Although, attention mechanisms have shown similar degrees of improvement on text classification applications ([**?**]).

Using this attention-based model we were able to identify those features from the APACHE II score that are more relevant for predicting the outcome of the patient (Figure 3). Retraining the model with just eight of these features was able to produce an ROC AUC just 11% lower than the one obtained with the full feature set. The essential features set comprised vital signs and measurements such as blood glucose, weight, blood pressure, temperature, heart rate, and oxygen saturation. The association of some of these entities with a poor health in diabetes patients is well documented in the clinical literature (Church et al., 2004). The fact that our deep learning model was able to draw conclusions like large population studies is remarkable. The power of deep learning algorithms to extract valuable clinical information that is hidden in EHR data cannot be overstated. These insights raise the question on the application of APACHE II score to rank diabetic patients in ICU. Getting correct APACHE II scores require adherence to strict guidelines and regular training of medical staff ([**?**]). A reduction on the number of dimensions that need to be considered to calculate the score without losing its predictive capability could be seen as an improvement in diabetic patient care.

IX. CONCLUSION/OPTIMIZATION

We developed a multi-network multi-attention deep learning model to predict the mortality of diabetic ICU patients 48 hrs before discharge. The evaluation of the model reported an AUC ROC that is higher than previous similar models. The model is a step further of previous works by engaging multiple networks and using both clinical notes and clinical features as source to parallel RNNs as well as implementing multi-attention on each of the parallel networks in the prediction task. This model

is an improvement in the right direction, which could be integrated in the clinical setting to manage the extensive resources that are being expended on diabetic patients and increase the efficiency of clinicians in the ICU.

Further optimization can be explored in the future to improve the scores and validate the results found in other datasets of similar composition. Additionally, further work is required to ensure the resiliency of the model by implementing k-fold cross validation with resampling as well as testing the trained model against inputs that have noise, that is, inputs that have certain random additional components. Another area of exploration is to analyze whether making additional pre-processing of the lab/chart events features, like normalization of the values would improve the performance or resiliency of the model. In addition, the continuous numerical values from the chart and lab events can be converted to categorical values (e.g. below normal range, normal range, above normal range). Categorical input values could be represented in one-hot encodings and it might be easier for the model to learn their patterns of distribution.

An interesting perspective that might be worth furthering the project on, would be to consider different observation window for the note, instead of the 48 hrs we used for the project, a window of 24 hrs or lesser time frame could be considered.

X. Code Repository

- Github
- Google Drive: data processing files included, *.p, *.npy.

References

[1] Dijkstra, E. W.. (1959). “A note on two problems in connexion with graphs”. *Numerische Mathematik*, 1(1), 269–271. https://doi.org/10.1007/bf01386390

[2] Erwing, M. (2004). “Escape from Zurg: An Exercise in Logic Programming”. *Journal of Functional Programming*, 14(3), 253-261.

[3] Iwasaki, H., Morimoto, T., & Takano, Y. (2011). “Pruning with improving sequences in lazy functional programs”. *Higher-order and Symbolic Computation* (formerly LISP and Symbolic Computation), 24(4), 281–309. https://doi.org/10.1007/s10990-012-9086-3

[4] Morimoto, T., Takano, Y., & Iwasaki, H. (2006). “Instantly Turning a Naive Exhaustive Search into Three Efficient Searches with Pruning”. *In* Lecture Notes in Computer Science (pp. 65–79). https://doi.org/10.1007/978-3-540-69611-7_4