České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Ruslan Jumurov**

Studijní program: Otevřená informatika
Obor: Umělá inteligence

Název tématu: **Meta-optimalizace solveru systému generálního a hlavního klíče**

Pokyny pro vypracování:

Použijte metody strojového učení pro zlepšení výkonu solveru systémů generálního a hlavního klíče.

Seznam odborné literatury:

- Anna Lawer: Calculation of Lock Systems. Master thesis, Royal Institute of Technology, 2004
- Edmund K Burke et al.: Hyper-heuristics – a survey of the state of the art. Journal of the Operational Research Society, 2013
- I. H. Witten: Data Mining – Practical Machine Learning Tools and Techniques

Vedoucí: MSc. Radomír, Černoch

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Ing. Filip Železný, Ph.D.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 25. 3. 2015

master's thesis

Meta-optimization of a master key system solver

Ruslan Jumurov

May 11, 2015

Supervisor: Radomír Černoch

Czech Technical University in Prague
Faculty of Electrical Engineering, Open Informatics

Acknowledgement

I would like to thank my supervisor, Bc. Radomír Černoch, MSc. for his leading and helping me with this master thesis and my family for their support.

Declaration

I declare that I worked out the presented thesis independently and I quoted all used sources of information in accord with Methodical instructions about ethical principles for writing academic thesis.

Abstract

Many practical problems are NP-Hard (or NP-Complete), for which best algorithms may guarantee only exponential worth-case time complexity, if P != NP. Typically, these algorithms can not do better, but several types of heuristics can be invented into algorithms to improve their performance, without any guarantee for better worth-case complexity.

Lock-chart solving is a NP-Hard problem. It is specified by a lock system – a set of keys and locks installed in one or several buildings, for each key customer decides which doors it should be able to open. Lock-Chart problem is formulated by these requirements. For lock-chart solving many different combinatorial optimization algorithms and techniques can be used. One of which is backtracking algorithm. This algorithm can use different pruning technique and heuristics. Heuristics are typically chosen randomly on each backtrack, with some participation of *random shakes* technique.

This master thesis presents approach of meta-heuristic optimization for Lock-chart problem Solver based on Machine Learning techniques. Main efforts of this thesis are: patterns and dependencies between heuristics and lock-chart problems were discovered, Decision Tree based heuristic selection system was created and included into the current backtracking algorithm, performance improvement was measured by two experiments.

Proposed Decision tree based heuristic selection approach outperformed the random heuristic selection approach in 92 problem instances out of 113.

Keywords

Meta-optimization; Lock-chart; meta-heuristic; Machine Learning

Contents

Abbreviations

List of abbreviations

abbrv.	explanation
ML	Machine Learning
CSP	Constraint satisfaction problem
TP	True Positive
FP	False Positive
TN	True Negative
FN	False Negative
DT	Decision tree
NP	Nondeterministic polynomial time
LSEP	Lock System Extension Problem
SAT	Boolean Satisfiability Problem
NFL	“No free lunch” Theorem
NN	Nearest neighbor
API	Application programming interface

1 Introduction

Nowadays many practical computational problems are NP-Hard (or NP-Complete), for solving of which best algorithms may guarantee only exponential worth-case complexity. Typically, these algorithms can not do better, but several types of heuristics were invented into algorithms, which can help to produce solution quicker, than approach without heuristic, without any guarantee for better worth-case complexity.

A lock system is a set of keys and locks installed in one or several buildings. For each key customer decides which doors it should be able to open. Formally, each key has its subset of locks, which can be opened by this key. Such requirements can be represented by matrix, in which keys appointed to rows and locks are to columns. This matrix is called Lock-chart system, for such configuration Lock-Chart problem can be formulated. Lock-chart is a NP-Hard problem, for its solution many different combinatorial optimization algorithms and techniques can be used. One of which is backtracking algorithm. This algorithm uses pruning technique and heuristics. Heuristics are typically chosen randomly on each backtrack, with some participation of *random shakes* technique. With lock-chart problems a lot of data, related to heuristic selection process, can be generated and collected; Machine Learning techniques can be applied to that data for tuning heuristic selection process.

This Master's thesis was initiated with the intent to find some patterns in *successful* heuristics and lock-chart characteristics and to optimize choice process using Machine learning techniques. Inspired by ideas and methods of meta- and hyper- heuristic optimization techniques of combinatorial problem solving, one of this project's goal is to optimize heuristic selection for Lock-chart problem described above.

1.1 Project Goal

Goals of this project are:

1. The discovering of patterns and correspondences between heuristics and lock-chart problems characteristics by Machine Learning methods
2. Application of the gained knowledge for backtracking algorithm improvement
3. Estimation of performance improvement

1.2 Project Outline

The second Chapter describes details of lock-chart problem, its complexity. The Third Chapter presents current Lock-chart problem solver and describes heuristics, which are used by the solver. Next Chapter considers heuristic selection process as, well-known in Machine learning, classification problem. The next two Chapters are describe Machine learning method selection and implementation. The Seventh Chapter presents results of the experiments, executed during this work. Chapter 8 gives a brief review of related and used materials. Finally, Chapter 9 concludes the work and makes possible future work outline.

2 Lock-Chart problem

In this chapter detailed lock-chart problem formulation is given. Starting from description of Lock system, mathematical representations of the key and the lock, then hardness analysis of the problem is shown.

2.1 Lock-chart problem description

In this section basic description of Lock system, mathematical representations of the key and the lock are given.

2.1.1 Lock-chart matrix

A lock system consists of a two different sets - set of locks and set of keys. Each key open a certain subset of locks in the lock system. One way to specify a lock system is to complete a lock chart. Keys are represented by rows and locks are by columns. 1 in $i-th$ row and $j-th$ column indicates that key i must be able to open lock j. 0 means that key i does not open lock j.

In the literature on the problem locks are also defined as *cylinders*, in this thesis these two definitions are used interchangeably.

As an example of lock-chart table, see Tab. 2.1. Keys that open several locks are

Key/Lock	1	2	3	4
MK1	1	1	1	1
MK2	1	1	0	0
MK3	0	0	1	1
PK4	1	0	0	0
PK5	0	0	1	0
PK6	1	0	0	0

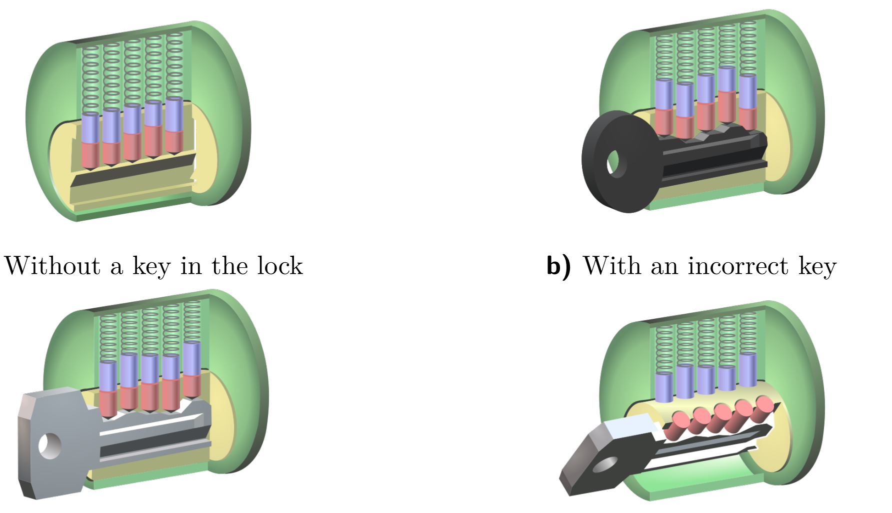
Table 2.1 Example of Lock-chart

referred to as master keys (MK1, MK2, MK3 in the table). If a key opens only one door it is called an individual or *proper* key (PK4, PK5, PK6).

A lock, which can be opened by more than 2 keys is called *central.* A structured lock system is a system that can be represented by a tree. Each node in the tree represents a key. Each key open all the doors that are opened by the keys in its subtree. The key in the root of this tree opens all the doors in the lock system and is referred to as the grand master key. All other lock systems are referred to as unstructured lock systems [1].

2.1.2 Mathematical representation of key and lock

The pin tumbler lock (or Yale lock, after lock manufacturers Yale) is a lock mechanism that uses pins of varying lengths to prevent the lock from opening without the correct key [2]. As an example of functioning lock mechanism, see Fig. 2.1.



a) Without a key in the lock **b)** With an incorrect key

c) With the correct key **d)** Unlocked with the correct key

Figure 2.1 Functioning of Pin key-lock system [2]

When inserting the key in the cylinder a number of pins will be pushed up against the key cuts. The pins are segmented in one or several places. If the key cuts have the appropriate height then the pins will stop in a position where they are segmented on the same level: the level of the shear line which is on the edge of the cylinder core. One will thus be able to turn the cylinder core and the door will open. If the key cuts have the wrong heights then one or more several pins will not be segmented at the shear line and will block any attempt to turn the cylinder core. The different depths of the key cuts can be represented by numbers. A sequence of numbers can therefore represent the key cuts [1].

Here,

$$< 1, 3, 4, 3, 7 > \tag{2.1}$$

i-th number in such sequence define that there is cut of depth x in position i. Lock is specified by an ordered sequence of sets $S_1 \ldots S_n$. Set S_i indicates which key cuts are accepted in the *i-th* position. There are $|S_1| \times |S_2| \times \cdots \times |S_n|$ potential keys that open a given lock.

2.2 Hardness of Lock-chart problem

In context of Lock-chart problem another very similar problem can be considered, which is called the Lock System Extension Problem.

Definition 2.1 *The Lock System Extension Problem (LSEP) is the following problem. Given the cylinder technology with n pins and only two different allowed cut depths for each pin-position. Given a set of cylinders C and a search space that corresponds to the entire cylinder technology. Is there a key in the search space that does not open any of the cylinders? [1]*

This problem is in field of interest of Lock-chart problem, because one can face it when *extension* of lock-chart is required. By extension we mean adding new keys. While adding new key k customer defines subset L_o from a set of locks L, set $L_n = L/L_o$ represents locks, which can not be opened by k. L_n set only set that figures in the LSEP problem. *LSEP is NP-complete.* There are exist two reductions for NP-Completeness prove [1]:

- Reduction of the SAT problem to to the Lock Calculation Problem
- Reduction to the Induced Sub-graph Isomorphism Problem

3 Backtracking algorithm

In this chapter backtracking algorithm and its possible usage for Lock-chart problem solving is described.

3.1 Lock-Chart problem as CSP

Lock-chart problem can be represented as Constraint Satisfaction Problem.

Definition 3.1 *Constraint satisfaction problem (or CSP) is defined by a set of variables, $X_1, X_2, \ldots, X_n$, and a set of constraints, $C_1, C_2, \ldots, C_m$. Each variable X_i has a nonempty domain D_i of possible values. Each constraint C_i involves some subset of the variables and specifies the allowable combinations of values for that subset. A state of the problem is defined by an assignment of values to some or all of the variables, $\{X_i = v_i, X_j = v_j, ...\}$. An assignment that does not violate any constraints is called a consistent or legal assignment. A complete assignment is one in which every variable is mentioned, and a solution to a CSP is a complete assignment that satisfies all the constraints. Some CSPs also require a solution that maximizes an objective function [3]*

In a CSP formulation of Lock-Chart problem, X set is the set of keys, and constraints are in Lock-key matrix. Treating problem as CSP gives some benefits, first of all, CSP problems are well-known research area, with developed set of tools and techniques, secondly, CSP has a good representation as a graph with:

- **Initial state** - the empty assignment of the variable set
- **Successor function** - function, which assigns new value to an unassigned variable, provided, that such assignment would not conflict with previous assignment
- **Goal test** - the assignment is complete and does no violate any constrains

As stated here [4], CSP can be formulated as search problem. Using such formulation any graph-search algorithm (Breadth-first search, Depth-first search, etc.) can be used. For solving search problems backtracking algorithm is also well-known option.

3.2 Naïve Backtracking algorithm

Backtracking algorithm is a modified version of the brute force search approach, which searches for a solution to a problem among all available options in systematic way. Solution in that case is represented by a vector V, each entry of which represents possible state of a variable. Often, it is Depth-first search based technique. In each step new applicable entry to V is added, if in current such entry does not exist, algorithm backtracks. If all entries of vector are filled, algorithm reports a solution. If it backtracked to initial state, it reports failure, or impossibility to solve CSP. In the backtracking algorithm for Lock-Chart v_i entry of vector V represents $i-th$ key configuration. Naïve algorithm for Lock-chart is presented in Algorithm 3.1.

Algorithm 3.1 Naïve backtracking algorithm

```
function BACKTRACK($L$)
    $V = \emptyset$
    $i = 1$
    while unassigned keys left AND i > 0 do
        while unassigned keys left and next(V, L) returns valid key codes do
            V[i] = next($V, L$)
            increment i
        end while
        if there are unassigned keys left then
            V[i] = null
            decrease i
        end if
        if V fully assigned then
            return V
        end if
    end while
    return impossible to solve L
end function
function NEXT($V, L$)
    return possible key, $k$
end function
```

In naïve approach function $next(V, L)$ is mentioned. This function takes current partial solution Vector V and Lock chart L and returns next key, which satisfy lock-chart constrains or informs impossibility of that. Search space for this approach grows rapidly, for selection of l keys out of n possible key configurations we would have:

$$\frac{n!}{(n-l)!} \tag{3.1}$$

number of possibilities. For example, if we have 5 keys out of 500 configurations, number of posibilities would be more than $3 * 10^{13}$. With factorial growth of possibilities this approach suits only for very small problems.

3.3 Backtracking algorithm refinement

As we have seen in previous subsection, naïve backtracking algorithm is applicable for small Lock-chart problems only. Some improvements for it are required. It can be achieved by adding pruning technique and heuristics for next key selection.

In the current realization of the Lock-chart solver backtracking algorithm with heuristics and pruning technique is implemented. Pruning is done, when violation of one or more of the constrains has occurred. Examples of such constrains can be:

- Key - two adjacent key pins have inappropriate length difference
- Lock - 3 or more lock parameters are equal
- Key-key - 2 keys have equal parameter more than in 2 positions
- Key-lock - Individual key opens subset of locks, described by a lock-chart

and some others. Specific parameters for each constraint are defined by a particular lock-chart problem, for example maximal difference between two adjacent pins in a key, number of equal positions for keys, etc.

Additional improvement of the backtracking algorithm is done by heuristics. For Heuristics for key-code generation, see Tab. 3.1

Name	Description
UseCapacityHeuristic	If enabled, algorithm tries to,make master and proper keys different as soon as possible
numToBePenalized	Defines number of positions, in which proper key should differ from master
UseEKeySelection	How should keys be sorted by number of opening keys, i.e. in increasing or decreasing order
UseDiagonal	More restricted, than numToBePenalized, define exact difference between master and own keys, cuts any other configurations
OwnAndMasterBorder	Defines *n-th* position, algorithm tries to encode master keys on the left side from position, and proper keys on the right side.

Table 3.1 Key-code generation heuristic

Using pruning technique and heuristics for code, we can refine *next*() function, see algorithm 3.2

For large Lock-chart problems pruning checking is high-cost operation, especially, Key-key and Key-lock constraint checking. For large problems it is reasonable for each key assignment check only *key* and *lock* constraints, while next two make on some periodical basis.

Algorithm 3.2 Next function with pruning and key-code generation heuristics

```
H - heuristic set, C - constraint parameter set
function NEXT($V, L, H, C$ )
    for all k in possible key codes according to H do
        add k to V
        if V satisfy C then
            return k
        end if
    end for
    return impossible to extend V
end function
```

4 Heuristic selection and metaheuristic optimization

In this chapter heuristic selection process is described, possible application of Machine Learning for metaheuristic optimization is discussed.

4.1 Heuristic selection

Heuristic selection for backtracking algorithm represents separate and well known research area. At the dawn of the backtracking algorithm development some random approaches were used, controlled randomization is perhaps the oldest strategy for seeking to overcome local optimality in combinatorial optimization. Often, it might take two different forms: the first is the well-known “random restart”, second - as “random shakeup”. “Random restart” approach injects a randomizing element into the generation of an initial starting point to which a heuristic is subsequently applied, “random shakeup” is the procedure which, instead of restarting, periodically generates a randomized series of moves that leads the heuristic from its customary path into a region it would not otherwise reach [5].

Current implementation of backtracking algorithm for Lock-chart problem uses very similar to mentioned previously random approaches for heuristic selection, see Algorithm 4.1

Algorithm 4.1 Solve with restarts algorithm

```
function SOLVEWITHRESTARTS($L$)
    initModel
    $restart = 0$
    $H = \emptyset$
    while restart < maxNumberOfRestarts do
        fillHeuristicParameters(H)
        if solveWithBacktracking(L, H) then
            return solution
        else
            increment restart
            continue
        end if
    end while
    return impossible to solve L
end function
function FILLHEURISTICPARAMETERS($H$)
    initialize H randomly
    return H
end function
```

More advanced techniques include meta- and hyper- heuristic optimization.

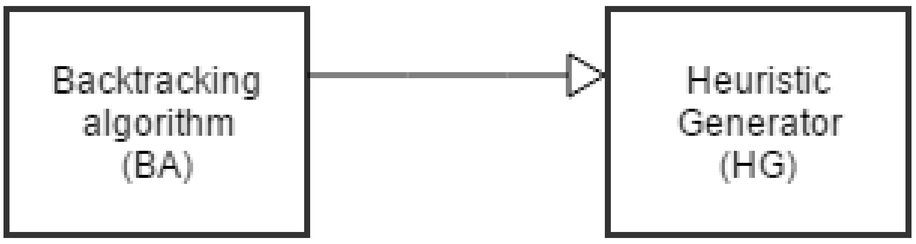


Figure 4.1 Heuristic generator with backtracking algorithm

Definition 4.1 *Metaheuristics are typically high-level strategies which guide an underlying, more problem specific heuristic, to increase their performance. The main goal of metaheuristic approach is to avoid the disadvantages random search. Many of the methods can be interpreted as introducing a bias such that high quality solutions are produced quickly. This bias can be of various forms and can be cast as descent bias (based on the objective function), memory bias (based on previously made decisions) or experience bias (based on prior performance). The main difference to pure random search is that in metaheuristic algorithms randomness is not used blindly but in an intelligent, biased form [6].*

Another method called hyper-heuristic approach is widely used. Hyperheuristics represent a class of methods that are non-problem specific. The decision for a heuristic selection is based on problem independent measures, such as the change in the quality of a solution when the selected heuristic is used [7]. The main limit of this project is the fact that the implemented search algorithm can not be changed. The only thing that can be influenced from outside - it is a heuristics selection process, which in current realization is random, see Heuristic generator in Fig. 4.1. That is, the project implements, in a certain sense, some of the approach of both methods. From Metaheuristic optimization we would have be a *"high-level strategy"*, which *"introducing a bias"* to a heuristic selection process, while from hyper-heuristic we would have a heuristic selection process, based on a measure of the change in the quality of a solution when the selected heuristic is used for specific problem.

4.2 Application of Machine Learning for heuristic selection

Due to specifics of backtracking algorithm with heuristics, it is possible to create some Machine Learning classifier, probabilistic model, etc., or, more informally, a *tool*, which for given instance of a problem can make a heuristic selection more instance specific and biased than simple random approach. Such a "tool" can help to avoid disadvantages of random search, concretely, add some bias toward heuristic selection process, while it would not change backtracking algorithm itself. Meanwhile, replace Heuristic generator in Fig. 4.1

Interesting might seen applying another "portion" of Metaheuristic optimization to a previous "tool", informally, Metaheuristic optimization for Metaheuristic optimization tool. But in that case we have a restriction in the "No free lunch theorem" (NFL), which states, that "if an algorithm performs well on a certain class of problems then it

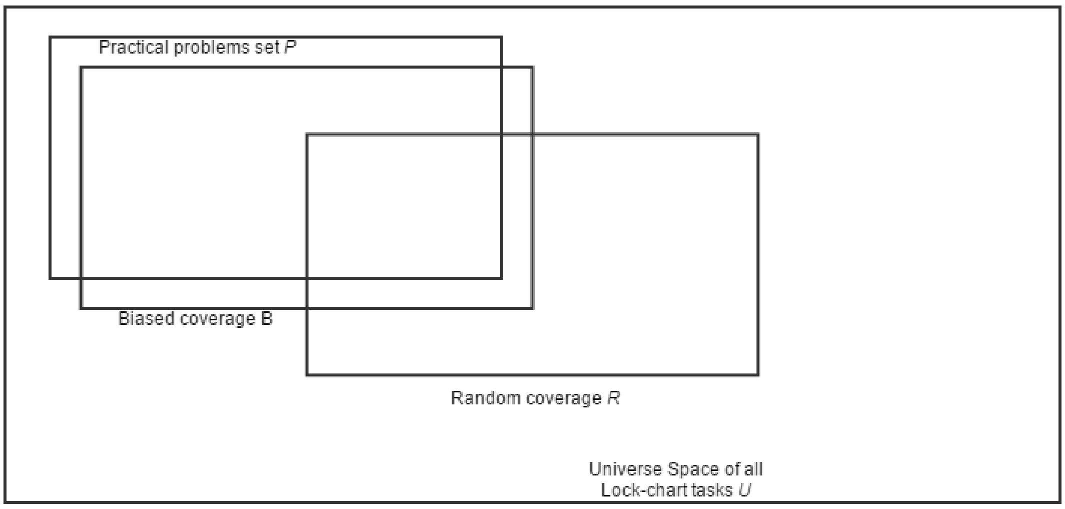


Figure 4.2 Representation of Lock-chart problem search space

necessarily pays for that with degraded performance on the set of all remaining problems" [8]. NFL theorem restricts a possibility of creation "tool" for heuristic selection, usage of which in backtracking algorithm would outperform backtracking algorithm with random heuristic selection on the whole search space. Under this restriction we only can try to outperform random selection on a certain area of the Universe Search Space. We might make an assumption that Lock-chart problems, which are commonly required to be solved in production, cover only a part of the Universe space. Our goal is to move coverage of backtracking algorithm with ML heuristic selector as close as it possible to coverage of production problems, for details see Fig. 4.2. Set P represents production problems coverage under Universal Space U, Random backtracking algorithm coverage denoted as R. Our goal is to move Backtracking algorithm by changing heuristic selection process toward set P.

5 Method selection

In this chapter justification of selected method is provided. Firstly, short description of Machine learning classification concept is given, then its possible application for heuristic selection tool is discussed and, finally, short explanation of Decision tree technique is done.

5.1 Classification problem in Machine Learning

In Machine Learning classification is the one of the most common and well-known problem. Informally, classification is a problem of assigning a new observation to one category (class) from a set of categories, based on a training set of data, containing observations, whose membership is known. It is need to be mentioned, that such approach lies in the field of Supervised learning.

Supervised learning is the machine learning task of inferring a function from labeled training data. The training data consist of a set of training examples. In supervised learning, each example is a pair consisting of an input object (typically a vector) and a desired output value (also called the supervisory signal). A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples. An optimal scenario will allow for the algorithm to correctly determine the class labels for unseen instances [9].

An algorithm that implements classification commonly called as classifier. This term also sometimes refer to a function, implemented by a classification algorithm, that maps input data to a category (class). A classification algorithm **L** is first trained on set $T_{train} : \{\mathbf{X_i}, c_i\}_{i=1}^{m}$ of size m, where X is an object, represented by its features, i.e. vector in n-dimensional feature space. c_i takes one of a fixed number of values, known as class. After that, created model (or hypothesis) is tested on a *test* set $T_{test} : \{\mathbf{X_i}, c_i\}_{i=1}^{m}$. For testing object features without class label are given to **L**. The goal in classification is to produce a hypothesis h that "best", according to some error function, predicts C_{train}. The classifier training and training work-flow is illustrated in Fig. 5.1. Performance of a classifier depends on the nature of the data, on which it trained and classified. Training

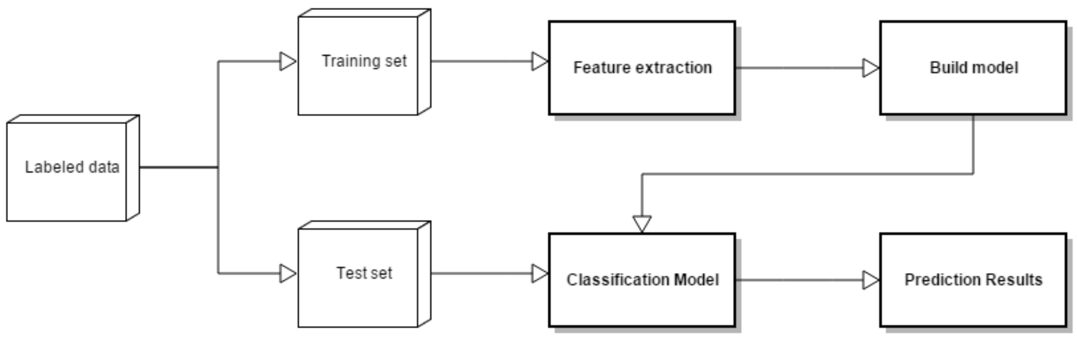


Figure 5.1 Work-flow of classifier training and testing

data should consist of independently and identically distributed (i.i.d.) examples. There is no single classifier that works best on all given problems (due to NFL theorem).

5.2 Heuristic selection as classification problem

As was stated in previous chapter, some refinement of heuristic selection process is required to achieve better results in Lock-chart problem solving by backtracking algorithm. As it was mentioned, the only part, which can be changed, is a heuristic selection process. This process can be represented as classification problem by combining problem's features P with heuristic set H to vector X. This vector X represents an object for classification, model should predict success or failure of backtracking algorithm on problem P with heuristic set H. Such classification procedure may help in heuristic selection process, for details see Algorithm 5.1.

Algorithm 5.1 Heuristic selection algorithm

```
function SELECTHEURISTICS($L, classifier$)
    $P = getTaskFeatures(L)$
    $Heuristics = \emptyset$
    for all h in all possible heuristic combinations do
        $X = \{L, h\}$
        if classifier.classify(X) == 1 then
            add h to Heuristics
        end if
    end for
    return Heuristics
end function
```

It is worth noting, that size of set of all heuristics growths rapidly with each heuristics' domain size.

$$|H| = |h_1| \times |h_2| \times \cdots \times |h_n| \tag{5.1}$$

For that reason, classification model should be fast enough to perform heuristic selection quickly. Heuristic selection process might be handled by a classifier algorithm. Such algorithm should have:

- relatively simple implementation process
- fast classification time, to handle possibly large amount of objects to classify
- self-descriptive process of classification to make dependencies between tasks and heuristics analysis process possible.

5.3 Nearest neighbor

The k-Nearest Neighbors algorithm (or k-NN for short) is a non-parametric method used for classification and regression.[1] In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression [10]. In k-NN classification, the output is a class membership, an object is classified by a majority vote of its k neighbors, with the object being assigned to the class most common among its k nearest neighbors, when k = 1, then the object is simply assigned to the class of that single nearest neighbor.

Algorithm works as follow:

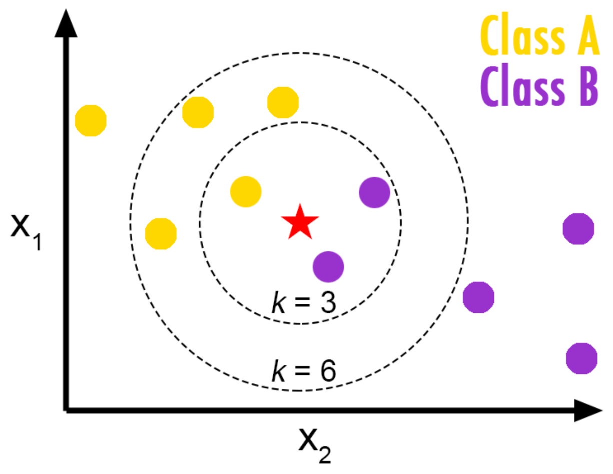


Figure 5.2 K-nn classification example with different k

- The training examples are vectors in a multidimensional feature space. The training phase of the algorithm consists of storing the feature vectors and class labels of the training samples and normalization of features, if it is required [11].
- In the classification phase, k is a predefined constant, and object for classification is classified by assigning the label which is most frequent among the k training samples nearest to that object. Example of k-nn is shown in Fig. 5.2.

A commonly used distance metric for continuous variables is Euclidean distance. For discrete variables, such as for text classification, another metric can be used, such as the overlap metric (or Hamming distance).

A major drawback of the basic "majority voting" classification occurs when the class distribution is skewed. That is, examples of a more frequent class tend to dominate the prediction of the new example, because they tend to be common among the k nearest neighbors due to their large number. This drawback is main restriction toward using this algorithm as heuristic selection tool, because class distribution is uneven.

5.4 Decision trees

A decision tree is a tree-like structure in which each internal node represents a "test" on an attribute, each edge represents the outcome of the test and each leaf node represents a class label (decision taken after computing all attributes). The paths from root to leaf represents classification rules [12]. Simple example of a decision tree illustrated in Fig. 5.3. This decision tree can be used to classify week day by weather, concretely, is it Saturday or not. A decision tree is well-known algorithm for classification. Exist many techniques and algorithms for tree induction, which are based on a concept of *information gain*. One of the most popular among them - is C4.5, developed by Quinlan [14]. One of the main C4.5 advantage - support of numeric attributes. General algorithm for decision tree induction is here [15] [16]

Decision tree has all required properties for heuristic selection process:

- Tree induction process is relatively simple. C4.5 Decision tree induction time complexity is has a time complexity of $O(m \cdot n^2)$, where m is the size of the training data and n is the number of attributes [17].
- Time complexity of classification is $O(k)$, where k is tree-depth, which is very fast for heuristic selection process.

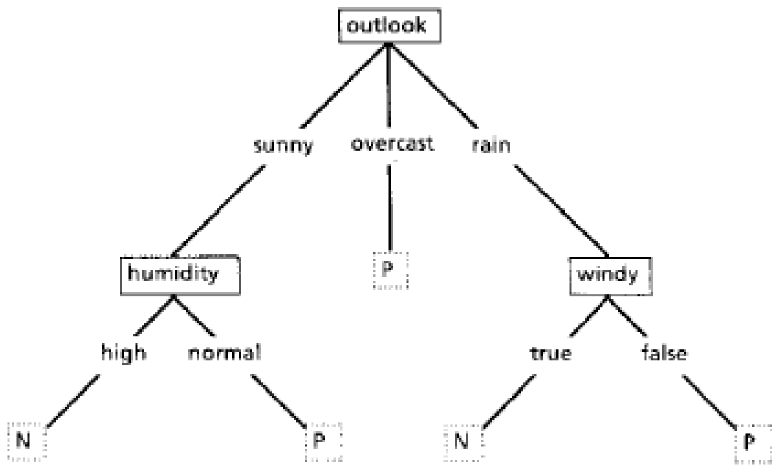


Figure 5.3 Example of a simple decision tree [13]

Algorithm 5.2 c4.5

```
function FormTree($D$)
    computeClassFrequency(D)
    if oneClass of few cases then
        Return a leaf;
    end if
    Create a decision node N
    for all Attribute A in D do
        computeGain(A)
    end for
    N.test = A with best gain
    if N.test continuous then
        find Threshold
    end if
    for all D' in the splitting of D by N.test do
        if D' is empty then
            Child of N is a leaf
        else
            Child of N = FormTree(D')
        end if
    end for
    return N
end function
```

- Due to information gain concept, decision tree is easily interpretative. Using result decision tree model, dependencies between object features and classes. In a previous example, it is always Saturday, when the "outlook" is overcast.

Decision tree classification algorithm was chosen, because it has all required properties.

6 Implementation

In this chapter detailed explanation of implementation process is given. Starting from data mining process, then creation of a decision tree explained, finishing with inclusion of decision tree heuristic selector to the backtracking algorithm for lock-chart solver.

6.1 Data mining process

First of all, we need to decide what features would represent a lock-chart. Properties of a task are:

- number of keys
- number of master keys
- number of locks
- number of central locks
- average over locks from number of accesses of master keys
- average over keys from number of access to non-central locks.

Feature vector for Tab. 2.1 would be [6, 3, 4, 3, 2, 0.33].

Second part of object for classification would be a heuristic vector, from heuristics, which are presented here Tab. 3.1. Each heuristic has its own domain:

- $UseEKeySelection \in \{0, 1\}$
- $numToBePenalized \in \{5, 6\}$
- $UseEKeySelection \in \{0, 1\}$
- $UseDiagonal \in \{0, 1\}$
- $OwnAndMasterBorder \in \{2, 3\}$

Heuristics started with “Use” (UseEKeySelection, UseDiagonal, UseCapasity) are binary – “yes” or “no”. Last two also have domain of size two. Computed by formula 5.1, number of combinations is 32. For one lock-chart problem instance 320 tries were performed (10 for each heuristic). Final object (vector) contains feature vector, heuristic vector and binary value, which indicates was task solved in that try or not.

Data mining process for one Lock-chart problem is shown here, Alg. 6.1

For data mining 20 hardest Lock-chart problems were chosen, each of which represents some particular complexity for lock-chart solving algorithm. Due to non-deterministic steps in backtracking algorithm this process was repeated 10 times for each task. By this way data was collected. This data contained some identical entries, which belonged to different classes. The reason was in non-deterministic nature of backtracking algorithm. During 10 tries with the same heuristic set, some tryouts ended successfully and some not. We had decided to aggregate such entries to one and, if at least three of tries finished successfully, entry added to final data set classified as positive, negative otherwise. Informally, such positive entry indicates, that has relatively high probability of success. Also, some additional information was collected (name of the task, run time, etc). Example of data object:

$$[01 : Y_0001_T, 3510, 404, 1, 403, 0.0, 1, 1.99, 5, 0.0, 5, 1, 1, 2, 1] \quad (6.1)$$

First two entries are name and run time in ms, next part is task features vector with heuristics, finalizing with the result of a try.

Algorithm 6.1 Data mining process

```
function MINEDATA($L$)
    $Result = \emptyset$
    $P = getTaskFeatures(L)$
    $Heuristics = getAllPossibleCombinations$
    for all h in all possible heuristic combinations do
        $X = \{L, h\}$
        if solver.solved($L$) then
            add 1 to X
        else
            add 0 to X
        end if
        add X to Result
    end for
    return Result
end function
```

6.2 Decision tree creation

Final data set was collected and filtered. Size of it is 1030 entries

For decision tree creation WEKA data mining and Machine learning tool was used. The WEKA project aims to provide a comprehensive collection of machine learning algorithms and data preprocessing tools [18]. C4.5 (J4.8 implementation for Java) decision tree was created with WEKA, for result, see Fig. 6.1. WEKA also has comprehensive Java API [19], it can be used in current implementation of the solver, which is written in Java as well. Summary of decision tree generation is listed in Tab. 6.1 During tree creation 10-Fold cross-validation was performed. Confusion matrix is presented here Tab. 6.2.

Characteristic	Value/Percent
Correctly Classified Instances	898/87.1845
Incorrectly Classified Instances	132/12.8155
Mean absolute error	0.153
Root mean squared error	0.3121
Coverage of cases	-/97.767

Table 6.1 Summary of Decision tree generation

a	b	classified as
533	76	a = 0
56	365	b = 1

Table 6.2 Confusion matrix

Result tree has interesting properties regarding to lock-chart problem properties. There are some leafs, path to which from root does not contain any “heuristic” nodes, which means, that it is impossible to affect the outcome through changes in heuristics.

During data mining process, some of the tasks were not solved by any combination of heuristics, properties of such tasks resulted in leaf with 0 result, without any heuristic.

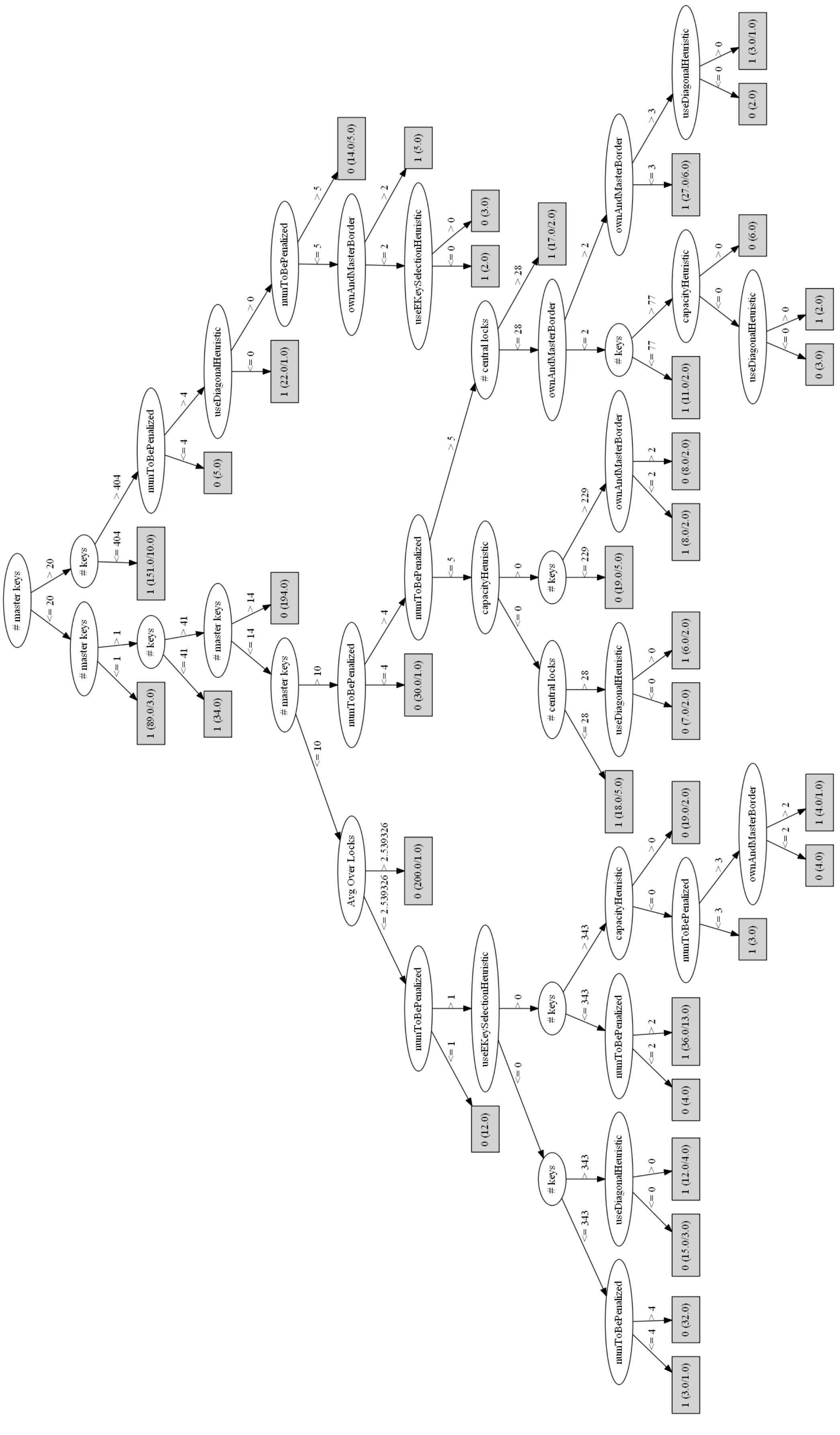


Figure 6.1 Decision tree

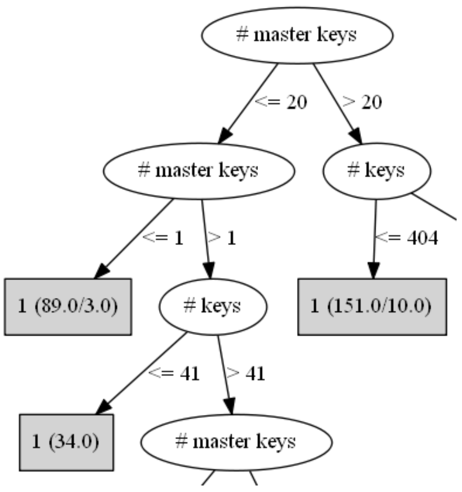


Figure 6.2 Part of the Decision tree with root

Part of the tree, illustrated in Fig. 6.2, has interesting properties. It shows how some characteristics of a task can affect on a result. For example, if task has a small number of master keys (smaller than 20) or just one, than such task can be solved by any heuristic combination, as well as if it has number of master keys in a range 1 and 20 and total number of keys lower that 44. Other leaves have heuristics on their path from root, which means, that result can be controlled by right heuristic selection.

6.3 Decision tree as heuristic selector

In previous section Decision tree creation process was shown, in this section usage of the tree as heuristic selector is discussed and final backtracking algorithm with decision tree as heuristic selector is presented.

Possible usage of classification algorithm as heuristic selection support tool was discussed in the section 5.2. In that case backtracking algorithm gives features of a task to heuristic selector, see Algorithm 5.1, which tries to classify the task with all possible heuristic combinations and return all "positive" heuristic sets. The classification results may or may not match task's actual status. Algorithm should handle each type of outcome properly or tries to reduce possible losses.

- True positive: solvable task is classified as solvable. Algorithm handles it properly.
- False positive: unsolvable task is classified as solvable. Algorithm tries to solve the task with proposed heuristics sets, without successful result.
- True negative: Unsolvable task classified as unsolvable. Algorithm handles this case properly.
- False negative: Solvable task classified as unsolvable. Algorithm fails to produce solution for solvable task.

Main drawback of proposed algorithm – improper handling of False negative cases. It can be handled by inclusion previously mentioned random heuristic generation process in case, when classifier based heuristic selector is failed to produce any heuristic combination. It produces another problem with True negative cases, in that case algorithm tries to solve unsolvable task with random heuristic set. This can be handled by reducing number of restarts, if heuristic selector produces empty candidate set. Final version of solving algorithm is presented in Algorithm 6.2.

Algorithm 6.2 Final Solve with restarts algorithm

```
function SOLVEWITHRESTARTS($L$)
    initModel
    $restart = 0$
    $H = \emptyset$
    $P = extractTaskFeatures(L)$
    $C = initClassifier()$
    $Cand\_H = selectHeuristics(P, C)$
    if size($Cand\_H$) $== 0$ then
        reduce maxNumberOfRestarts
    end if
    while restart < maxNumberOfRestarts do
        fillHeuristicParameters($H, Cand\_H$)
        if solveWithBacktracking($L, H$) then
            return solution
        else
            increment restart
            continue
        end if
    end while
    return impossible to solve L
end function
function FILLHEURISTICPARAMETERS($H, Cand\_H$)
    if size($Cand\_H$) $> 0$ then
        return pop($Cand\_H$)
    else
        return randomly initialized H
    end if
end function
```

7 Experiments and Results

In this section description of implemented experiments and their results are given. Starting from description of chosen methods for experiment,then continuing with evaluation of created model with training data set and finishing with testing on a large data set.

7.1 Experiment setup

In the previous section decision tree and its usage as heuristic selection tool was shown. The main method for testing is the comparison of time and number of tries required to solve tasks by backtracking algorithm with random heuristic selection and with decision tree selection was chosen. The reason to choose such evaluation criteria is, because main goal of the project is to improve run time of the algorithm.

Firstly, algorithm was tested on the training set of 20 hardest tasks. Each task was solved by two versions of algorithm separately. After that, large set of about 160 tasks was collected and used for experiments, with which the same procedure was done. The main reason to provide the experiments with two different sets of problems is to identify possible under-fitting and over-fitting in decision tree.

First problem set (training set) is used for under-fit checking, i.e. if algorithm works poorly on problems, with which training set was generated, reason of this might be in under-fitting. In order to identify possible over-fitting second problem set was collected. Tests performed on this problem set can help to identify it. Information about type I and type II errors (false– negatives and positives) also might be helpful for both cases. This might be useful because some of the tasks were classified as unsolvable and were solved by the same algorithm, i.e. by backtracking algorithm with random heuristic selection. As it was mentioned in Chapter 6, for proper handling of false positive cases, algorithm, if no heuristics were selected by decision tree selector, makes some tries with randomly initialized heuristics. If such tries ended successfully or separated run of algorithm with random selection solved this problem, such task may be called as false negative, i.e. incorrectly identified as unsolvable. For that purpose for both experiments confusion matrices are provided. Also sensitivity, specificity and precision measurements are provided.

$$TPR = TP/P = TP/(TP + FN) \tag{7.1}$$

$$SPC = TN/N = TN/(TN + FP) \tag{7.2}$$

$$PPV = TP/(TP + FP) \tag{7.3}$$

Sensitivity, or true positive rate (TPR) is computed by formula 7.1, specificity (SPC) - formula 7.2, and for precision, or positive predictive value (PPV) formula 7.3 can be used.

To handle non-determinism 5 tries for each heuristic set were performed. Due to high computing power requirements, The National Grid Infrastructure, MetaCentrum,

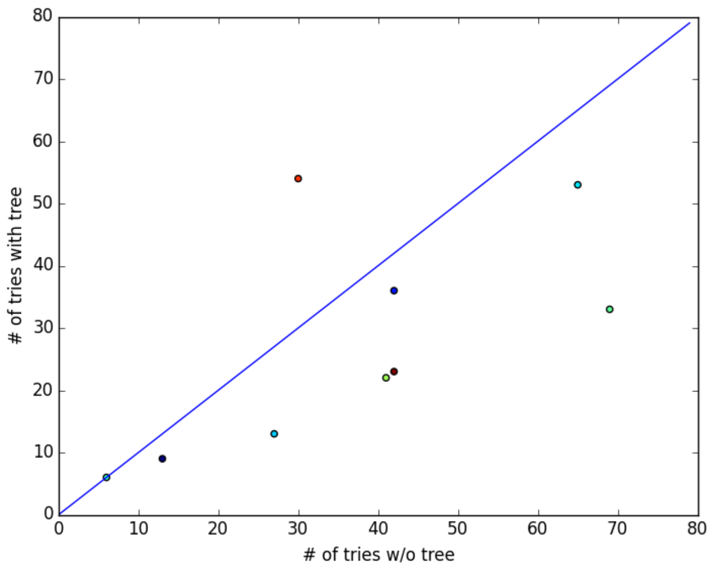


Figure 7.1 Number of tries, required to solve task by different version of algorithm, experiment #1

was used. MetaCentrum project, an activity of the CESNET association, operates and manages distributed computing infrastructure consisting of computing and storage resources owned by CESNET as well as those of co-operative academic centers within the Czech Republic. MetaCentrum is responsible for building the National Grid and its integration to related international activities, especially in the European Union. It is actively involved in many international Grid projects such as EGI, InSpire, EUAsia-GRID, CHAIN [20].

Each computational node has following configuration:

- Memory: 12 GB
- OS: linux, x64
- CPU Cores: 16

Solving system was implemented on Java, Decision tree support tool was created with WEKA API.

7.2 Experiments

Two experiments are presented in this section. The first experiment evaluates the performance on a training set. This experiment is done with possibility to identify under-fitting The second one shows the behavior of the algorithm on unseen problem set with relatively large number of tasks. This experiment can help to find possible over-fitting during tree creation. Both experiments are used to identify is better approach for Lock-Chart problem solving.

7.2.1 Evaluation on the training set

As it was mentioned in a previous chapter, for decision tree training 20 most hardest tasks were used. First step of evaluation process was done by testing implemented algorithm on this problem set. The results of the of the experiments on 20 tasks are shown in Fig. 7.1 and Fig. 7.2.

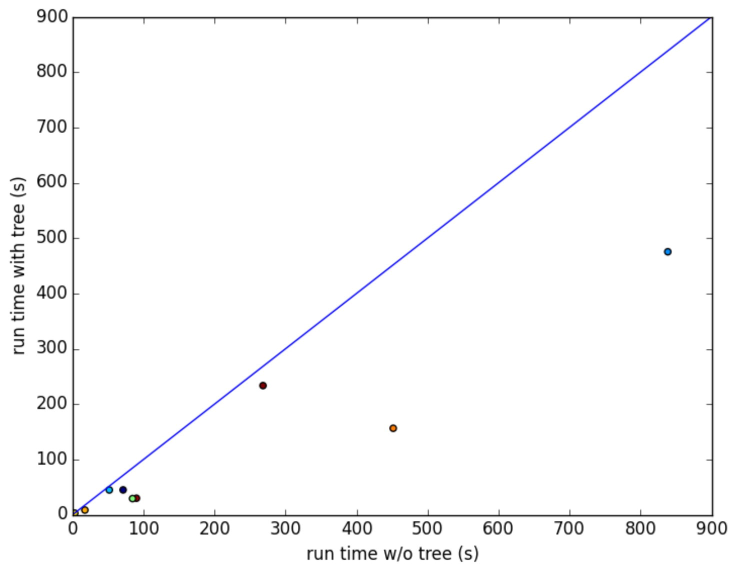


Figure 7.2 Time, required to solve task by different version of algorithm, experiment #1

First scatter plot represents number of tries, required to solve a task, x-axis denotes number of tries required by backtracking algorithm with random heuristic selection, y-axis - with Decision tree as selector. Tasks (points), lie in right-bottom corner of the square, represent outperformed by decision tree selector, opposite is correct for left-top corner. Points, lie strictly on the diagonal, represent tasks which were solved by the same number of tries. This plot shows, that solving only one task took less number of tries with random heuristic selection approach.

Second plot represents the same relationship, but for time, required to solve tasks. Here, backtracking algorithm with decision tree outperformed random in all cases.

In this experiment 20 tasks were used, but only part of them is presented on the plots. This is because some of the tasks were classified as unsolvable and were solved by the same algorithm, i.e. by backtracking algorithm with random heuristic selection. Plotting of such tasks is unreasonable, because they can not help to measure performance of the decision tree heuristic selector.

a	b	classified as
6	0	a = 0
5	9	b = 1

Table 7.1 Confusion matrix for experiment #1

By data from Tab. 7.1 it is possible to calculate TPR, SPC and PPV.

- $\text{TPR} = 9/(9+6) = 9/15 = 0.6$
- $\text{SPC} = 6/(6+0) = 6/6 = 1$
- $\text{PPV} = 9/(9+0) = 9/9 = 1$

As we can see, evaluated on training data, perfect specificity and precision, while recall is only 60%. From the data obtained it can be concluded that under-fitting is possible, because has only 15 out of 20 correctly identified instances. To prove or refute, second experiment is required. It is well-known, that high specificity with low sensitivity often leads to many false negatives (FN) with few number of false positives (FP). By this we

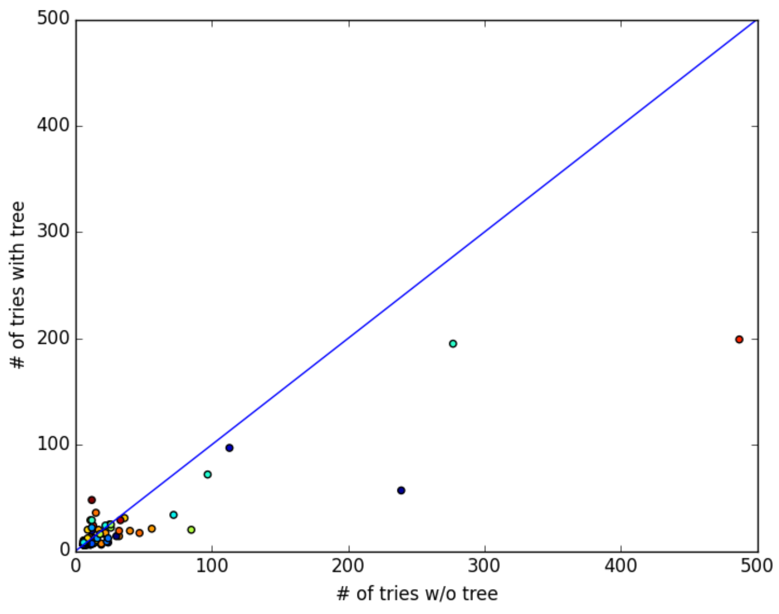


Figure 7.3 Number of tries, required to solve task by different version of algorithm, experiment #2

may do a prediction for evaluation on a test set. It should have more FN than FP.

In the first experiment decision tree based algorithm outperformed/showed the same performance as random on 8 tasks, out 9 predicted. Algorithm has shown good performance on training data, correctly classified 15 out of 20 problems.

7.2.2 Evaluation on the test set

Second experiment was done on almost random problem set, consisted of 160 problems. The results of the of the experiment are shown in Fig. 7.2 and Fig. 7.3.

First scatter plot represents number of tries, required to solve a task, x-axis denotes number of tries required by backtracking algorithm with random heuristic selection, y-axis – with Decision tree as selector. Tasks (points), lie in right-bottom part of the square, represent outperformed by decision tree selector, opposite is correct for left-top corner.

These plots show, that random heuristic approach in many cases required more tries (or time) than decision tree based. We can see that with a large number of tries required a solution with a tree gives a better result, because much more points with number of tries more than 50 are concentrated on right-bottom side, which means, that random approach required more tries, see Fig. 7.5. There we can see, that for solving all tasks Decision tree approach outperformed random. Different situation with time cut on 50 seconds, see Fig. 7.6. There we have 3 on 5 split.

In the lower left corner there is an accumulation of points, which is not possible to see in full details. shows number of tries required for solving by both algorithms, bounded by maximum number of 50 tries, see Fig. 7.7. This can help to identify accumulation of points in left-bottom corner. These points represent tasks, for solving of which small amount of tries and time was required.

Fig. 7.8 represents cut by 50 seconds of total run time maximum. In left-bottom corner there is an accumulation of the points, where random approach has better per-

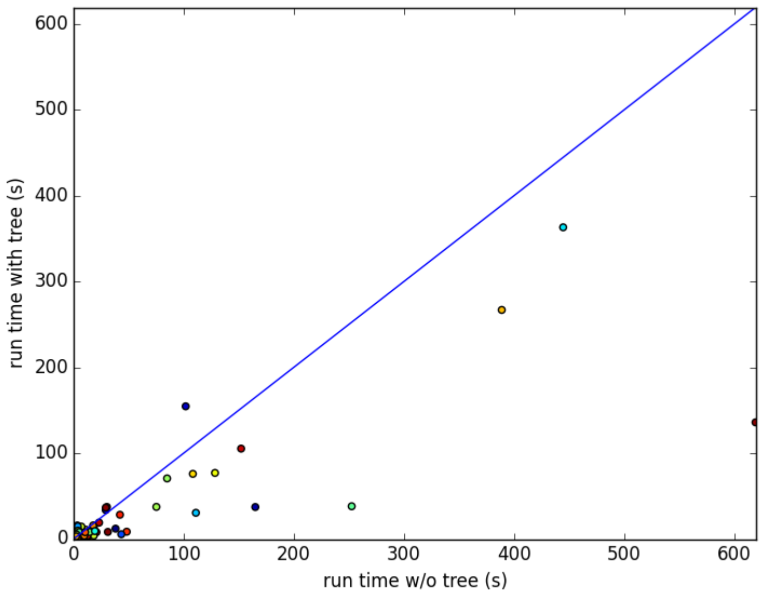


Figure 7.4 Time, required to solve task by different version of algorithm, experiment #2

formance, but further we can see dominance of decision tree based.

Tab. 7.2 is confusion matrix for experiment #2. With it, we can confirm or refute the prediction, made on the based on confusion matrix of previous experiment. There are 12 FP and 17 FN, which confirms predicted behavior.

In the second experiment decision tree based algorithm outperformed/showed the same performance as random on 92 tasks, from 113 predicted. Assumption that on both FN and TN tasks algorithm has the same performance due to identical random approach was done. Assumption, that classifier under-fit the data is not proven, as we can see, it shows good performance on the test set – 131 correctly classified instance out of 160.

a	b	classified as
18	17	a = 0
12	113	b = 1

Table 7.2 Confusion matrix for experiment #2

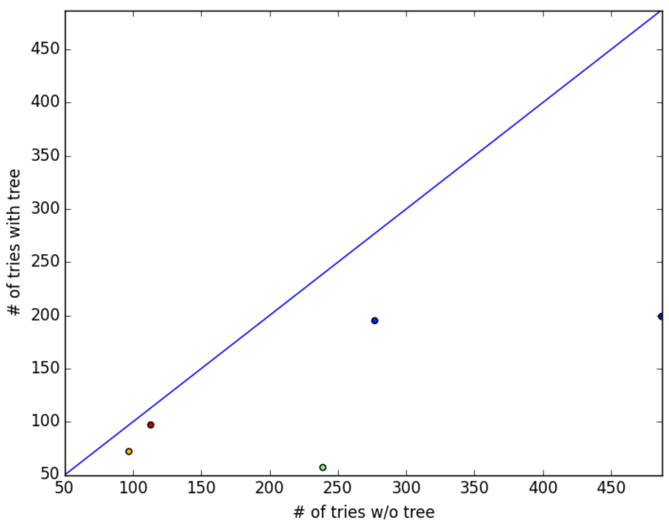


Figure 7.5 Number of tries, required to solve task by different version of algorithm with 50 minimum tries, experiment #2

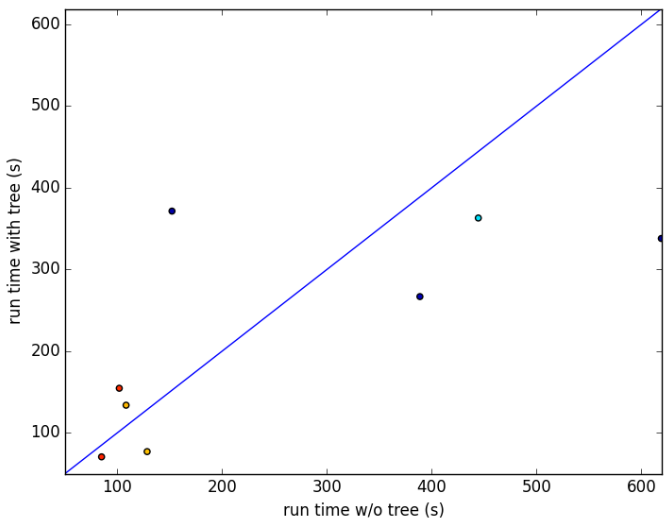


Figure 7.6 Time, required to solve task by different version of algorithm, 50 s minimum, experiment #2

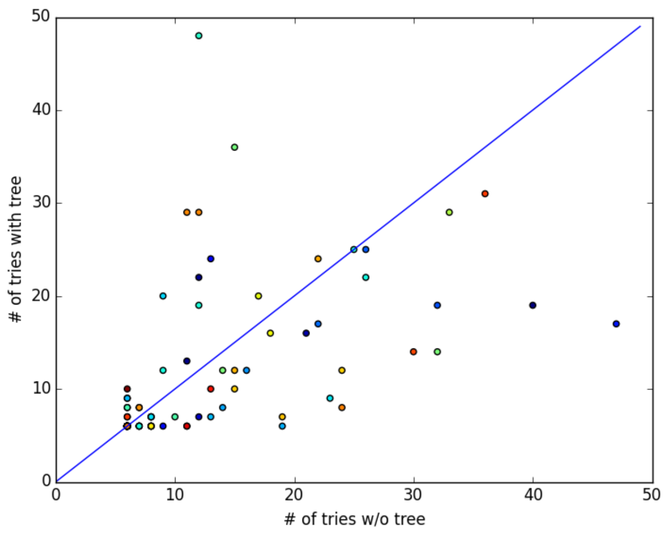


Figure 7.7 Number of tries, required to solve task by different version of algorithm with 50 maximum tries, experiment #2

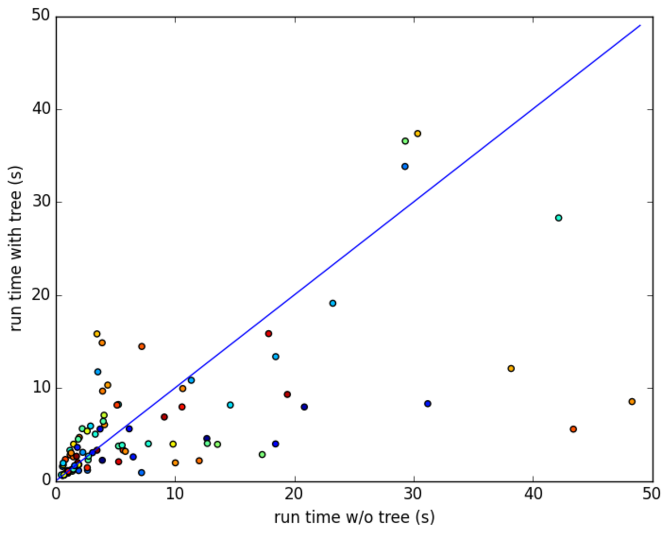


Figure 7.8 Time, required to solve task by different version of algorithm, 50 s maximum, experiment #2

8 Related works and used materials

In this chapter two different clusters of related works and used materials, which were used for this thesis, are presented.

First cluster represents works, related to Lock-chart problem and its possible solution techniques. It includes works about Lock-chart problems, Constraint satisfaction problem, backtracking algorithm.

Second one consists of works, related to meta- and hyper- heuristic optimization and Decision tree algorithm.

8.1 Lock-chart problem materials

There are not a lot of works in the literature, related to Lock-chart problem. One of the main source of information about this problem, which was used for the thesis, is Master thesis by Anna Lawer, "Calculation of Lock Systems" [1]. In this work basic definitions for *key*, *lock*, *lock-chart* concepts are given. Meanwhile, complexity analysis, which was mentioned in Chapter 2, is given. One of the main contribution of this work is prove of *Lock System Extension Problem* as NP-Complete problem. This was done by reduction of the SAT problem to to the Lock Calculation Problem and reduction of the Lock-calculation to the Induced Sub-graph Isomorphism Problem.

Possible solution by backtracking algorithm was presented and implemented, this algorithm uses some pruning techniques, without any heuristics, such approach tested on different problems with maximum number of 72 keys and 60 locks.

As a guide for Constraint satisfaction problem classical "Artificial Intelligence: A Modern Approach" by Stuart Russell and Peter Norvig was used. This book provides comprehensive description of CSP and possible ways of its solution, including backtracking algorithm.

8.2 Meta- and hyper- heuristic optimization materials

A lot of works, related to the meta- and hyper- heuristic optimization problems have been published in recent years. But at the same time, the definitions are very general and unspecified. Started from classical Glower's "Future paths for integer programming and links to artificial intelligence" [5], where one of the first mentions about meta-heuristics is given. This article gives comprehensive review of random heuristic selection techniques and presents idea of meta-heuristic optimization for simulated annealing, tabu search and hill-climbing search techniques. Article by Thomas Stützle "Local search algorithms for combinatorial problems: analysis" [6], provides more concrete definitions for meta- and hyper-heuristic techniques.

As a guide for Decision tree algoritm, mainly, two articles were used. First, is "C4.5: programs for machine learning" by John Ross Quinlan [14], author of C4.5 algorithm. From this article the algorithm is originated. Second one, "Efficient C4.5" by Salvatore Ruggieri, provides broad description of the algorithm, its possible usage and implementation details and tricks.

9 Conclusions and Feature work

This master thesis presents approach for meta-heuristic optimization for Lock-chart problem Solver based on Machine Learning techniques. First of all, lock-chart problem was defined and solution technique by backtracking algorithm was presented. After that, possible usage of heuristic selection “tool” was discussed and implemented. Two different experiments show significant improvement of backtracking algorithm performance with decision tree heuristic selection approach.

During working on this project large data set was collected. After different Machine Learning Techniques tryouts were performed, Decision tree, as a support tool for heuristics selection process, was chosen, by the reason of relatively simpleness, good performance and clearness of classification process.

Backtracking algorithm with proposed and implemented Decision tree support tool outperformed naïve backtracking with random approach in 92 problems out of 113.

Project goals were fulfilled:

1. Patterns and dependencies between heuristics and lock-chart problems were discovered.
2. Decision tree was created and was included in current algorithm
3. Performance improvement was estimated by two experiments

Future work involves different directions and improvements:

- Usage of different machine learning algorithms (k-NN, probabilistic model, random forest, etc.)
- Involving more tasks for training data set generation
- Usage of discovered dependencies between heuristic and task parameters for algorithm improvement
- Usage of discovered dependencies between heuristic and task parameters for another heuristic generation

The method of usage Machine Learning based “tool” has shown its perspective for Lock-chart problem optimization. In this case different ML algorithms may be tried to find optimal one.

For training data set generation 20 hardest tasks were used. It is reasonable to include additional hard task from training set.

Bibliography

[1] Anna Lawer. “Calculation of Lock Systems”. Master. Royal Institute of Technology, 2004.

[2] Wikipedia: The Free Encyclopedia. *Pin tumbler lock*. URL: `http://en.wikipedia.org/wiki/Pin_tumbler_lock` (visited on 29/04/2015).

[3] Peter Russell Norvig. “Artificial intelligence: a modern approach (International Edition)”. In: (2002).

[4] Alan K Poole David Mackworth. *Artificial Intelligence: foundations of computational agents*. Cambridge University Press, 2010.

[5] Fred Glover. “Future paths for integer programming and links to artificial intelligence”. In: *Computers & operations research* 13.5 (1986), pp. 533–549.

[6] Thomas G Stützle. *Local search algorithms for combinatorial problems: analysis, improvements, and new applications*. Vol. 220. Infix Sankt Augustin, Germany, 1999.

[7] Ender Özcan, Burak Bilgin, and Emin Erkan Korkmaz. “A comprehensive analysis of hyper-heuristics”. In: *Intelligent Data Analysis* 12.1 (2008), pp. 3–23.

[8] William Wolpert Macready. “No free lunch theorems for optimization”. In: *Evolutionary Computation, IEEE Transactions on* 1.1 (1997), pp. 67–82.

[9] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT press, 2012.

[10] Naomi S Altman. “An introduction to kernel and nearest-neighbor nonparametric regression”. In: *The American Statistician* 46.3 (1992), pp. 175–185.

[11] Luai Al Shalabi and Zyad Shaaban. “Normalization as a preprocessing engine for data mining and the approach of preference matrix”. In: *Dependability of Computer Systems, 2006. DepCos-RELCOMEX'06. International Conference on.* IEEE. 2006, pp. 207–214.

[12] Wikipedia: The Free Encyclopedia. *Decision tree*. URL: `http://en.wikipedia.org/wiki/Decision_tree` (visited on 29/04/2015).

[13] J. Ross Quinlan. “Induction of decision trees”. In: *Machine learning* 1.1 (1986), pp. 81–106.

[14] J Ross Quinlan. *C4.5: programs for machine learning*. Elsevier, 2014.

[15] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. *Supervised machine learning: A review of classification techniques*. 2007.

[16] Salvatore Ruggieri. “Efficient C4. 5 [classification algorithm]”. In: *Knowledge and Data Engineering, IEEE Transactions on* 14.2 (2002), pp. 438–444.

[17] Jiang Su and Harry Zhang. “A fast decision tree learning algorithm”. In: *Proceedings of the National Conference on Artificial Intelligence*. Vol. 21. 1. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999. 2006, p. 500.

[18] Mark Hall et al. “The WEKA data mining software: an update”. In: *ACM SIGKDD explorations newsletter* 11.1 (2009), pp. 10–18.

[19] F Eibe, H Mark, and T Len. *Weka api*. 2010.

[20] Metacentrum. *Project MetaCentrum*. URL: `https://www.metacentrum.cz/en/about/meta/index.html` (visited on 29/04/2015).