Appendix A
Dagger Software System

Dagger software system is being developed in KAUST for research and educational purposes. The system implements several algorithms dealing with decision trees, systems of decision rules, and more general class of problems than can be solved by dynamic programming. Dagger has modular structure and can be extended further.

This appendix describes capabilities and architecture of Dagger. Section A.1 describes features of the system, lists third-party libraries used, and states requirements to environment for running Dagger. Section A.2 overviews key design decisions.

A.1 Introduction

Dagger is capable of performing several tasks that can be divided into two classes: general tasks related to dynamic programming and more specific tasks related to building decision trees and systems of decision rules. The first group of tasks operates with abstract objects (subproblem, uncertainty measure, cost function, etc.) that are defined differently for each domain. It contains the following operations:

- building of DAG up to specified uncertainty threshold $\alpha$;
- computing minimum cost of a solution relative to a given cost function;
- optimizing DAG such that it describes optimal solutions only;
- finding a set of Pareto-optimal points for a pair of criteria (each criterion can be either uncertainty measure or cost function);
- counting the number of optimal solutions.

The second group contains the following operations:

- building one of decision trees (system of decision rules) described by DAG;
- greedy algorithms for building decision trees and systems of decision rules.

\footnote{The name origins from DAG, directed acyclic graph.}
Table A.1  List of main libraries

<table>
<thead>
<tr>
<th>Library name</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_set</td>
<td>Generating, loading, storing and other transformations of data sets</td>
</tr>
<tr>
<td>partition</td>
<td>Representation of decision table and uncertainty measures for decision table</td>
</tr>
<tr>
<td>model</td>
<td>Common operations with classifiers: pruning, error estimation, etc.</td>
</tr>
<tr>
<td>tree</td>
<td>Implementation of decision tree object. Greedy algorithms for decision tree construction</td>
</tr>
<tr>
<td>rule</td>
<td>Implementation of system of decision rules object. Greedy algorithms for construction of a system of decision rules</td>
</tr>
<tr>
<td>dag</td>
<td>Implementation of dynamic programming problem solver. Support functions for parallel computing</td>
</tr>
<tr>
<td>dag_table</td>
<td>Implementation of objects common to decision trees and systems of decision rules used by dynamic programming algorithm</td>
</tr>
<tr>
<td>dag_tree</td>
<td>Implementation of objects specific to decision trees used by dynamic programming algorithm</td>
</tr>
<tr>
<td>dag_rule</td>
<td>Implementation of objects specific to systems of decision rules used by dynamic programming algorithm</td>
</tr>
</tbody>
</table>

Fig. A.1  Graph of library dependencies

In addition to the key operations listed above, Dagger performs several auxiliary tasks such as loading and preparing decision tables, generating decision tables for Boolean functions and random decision tables, calculating prediction error for models, and others.

Dagger consists of a set of applications and dynamic libraries. Each dynamic library in the system covers certain functional area. Table A.1 contains the list of main libraries and short description of their functions. Library dependencies form an acyclic graph shown on Fig. A.1.

Dagger uses several third-party libraries:

- GetPot [1] for parsing command line arguments and configuration files,
- PThreads [4] for thread management,

The front end of Dagger is a set of console applications taking parameters from command line or configuration files and reporting result to standard output and log files. Decision table can be loaded from a comma-separated values (CSV) file of a
special format or ARFF [7] file. The objects that require graphical representation (DAG, decision tree) are saved in GraphML [5] format and can be displayed by third-party applications (for example, yEd [3]). Dagger is capable of drawing plots of functions in LaTeX using PGFPlots package [2].

Most operations allow for parallel execution in a large-scale multiprocess environment. The system uses hybrid parallelization. Computational load is distributed between several processes, and the number of processes can vary in a wide range. Each process contains two threads that provides nonblocking interprocess communication.

The most part of code is platform-independent, however, some thread synchronization primitives and means for console output are Windows-specific.

A.2 Architecture

Building the graph of subproblems differs from a typical problem solved by dynamic programming. First, for some domains, the graph has an irregular structure. For example, consider a graph of subtables of a decision table. The number of children in each node depends on values of conditional attributes for each row and assignment of decisions. Graph structure remains unknown until the graph is constructed. The second peculiarity is absence of data locality. If we consider parents of a particular node, layer of the parents may vary greatly (by layer of a node we mean the minimum length of a path from this node to the root). In any division of the graph into a few parts of equal size, the number of edges crossing the part boundaries is large.

The above-mentioned characteristics influence implementation of the algorithms for a multiprocessor environment. Absence of prior knowledge about graph structure makes it impossible to do explicit partitioning of the graph and assign each part to own process. Absence of data locality implies high amount of interprocess communications, so traditional star-like communication scheme is ineffective as communication with the main process become a bottleneck.

Dagger makes implicit graph partitioning in the following way. A hash function is defined for subproblems and each process owns a sub-range of its values. Thus to assign a subproblem to a process, value of the hash function is calculated and then the problem passed to the process that owns corresponding range of hash values. It is important the hash value to be the same for a subproblem regardless of the order of decompositions of the initial problem. For example, description of a subtable by set of constraints of the form “attribute = value” does not work as different sets of constraints can describe the same subtable. Instead, a subtable in Dagger is described by a vector of row indices relative to the initial decision table. This property allows for effective identification of duplicate subproblems. If the same subproblem appears multiple times, all related requests will be addressed to the same process based on the calculated hash value. The owner process stores solutions for all problems and reuses it when receives the second and further requests for the same subproblem. As
a result, each process keeps a part of graph nodes and performs operations on it that makes algorithm scalable and applicable to larger problems.

Computations are organized as a number of worker processes and the master process that synchronizes work of others and interact with the front end modules. Most of the time worker processes communicate with each other directly in order to reduce synchronization overhead and avoid overloading the master process. Communication scheme can be represented as a virtual network whose structure is described by the graph of subproblems. In this network, each node can communicate with its parent and child nodes. Worker processes communicate with each other asynchronously using queue of incoming and outgoing messages.

The whole task is described by working scenario that is a sequence of jobs such as graph building, computing the minimum cost, optimization, etc. Working scenario is provided by application that calls methods of the object representing the master process. Upon calling a particular method, the master process announces to worker processes start of a particular job and then waits for job termination. There are barrier synchronization points between jobs so the next job is started only when all processes are done with the previous job.

For example, consider implementation of the operation that calculates the minimum cost of a solution after the graph of subproblems has been constructed. On start of the operation the master process broadcasts to all worker processes the name of the cost function that will be used. Upon receiving this message, each worker process iterates through own list of graph nodes. For each terminal node, the cost of solution is calculated and passed to all parent nodes. Each intermediate node accumulates downstream messages until messages from all children are received. Then the minimum cost of solution for the node is calculated and passed to all parents. Following these rules, messages are propagated in the graph from the terminal nodes towards the root. The operation is completed when the root node receives messages from all children and calculates minimum cost of solution of the original problem.

References
The aim of this book is to extend the framework of dynamic programming and to create two new dynamic programming approaches: multi-stage optimization and bi-criteria optimization. Both approaches are applicable in the cases when there are two or more criteria of optimization on the considered set of objects. The first approach allows us to optimize objects sequentially relative to a number of criteria, to describe the set of optimal objects, and to count its cardinality. The second approach gives us the possibility to construct the set of Pareto optimal points and to study relationships between the considered criteria.

We concentrate on the study of four directions related to decision trees, decision rules and rule systems, element partition trees, and classic combinatorial optimization problems. For each of these directions, we created algorithms for multi-stage optimization, and for the first three directions – for bi-criteria optimization. We proved the correctness of these algorithms and analyzed their time complexity. We described classes of decision tables for which algorithms for decision trees, rules and rule systems have polynomial time complexity depending on the number of conditional attributes in the input tables. The algorithms for element partition trees are polynomial depending on the size of input meshes, and the algorithms for four classic combinatorial optimization problems studied in this book are polynomial depending on the input length.

For each of the considered four directions, we described different applications of the proposed techniques related to studying problem complexity, data mining, knowledge representation, machine learning, PDE solver optimization, and combinatorial optimization.

Future study will be devoted to the extension of the proposed methods to the decision tables with many-valued decisions and to inhibitory trees and rules. Decision tables with many-valued decisions are, sometimes, more adequate models of problems than usual decision tables. For some decision tables, inhibitory trees and rules describe more information contained in the table than decision trees and rules.
Index

A

Algorithm

\( A_{10} \) – construction of \((U, \alpha, \beta)\)-greedy set of decision rules for decision table, 138

\( A_{11} \) – construction of POPs for decision rules, cost versus cost, 152

\( A_{12} \) – construction of POPs for decision rule systems, cost versus cost, 157

\( A_{13} \) – construction of POPs for decision rules, cost versus uncertainty, 176

\( A_{14} \) – construction of POPs for decision rule systems, cost versus uncertainty, 180

\( A_{15} \) – construction of DAG \( \Delta(M) \), 208

\( A_{16} \) – counting the number of element partition trees, 209

\( A_{17} \) – procedure of element partition tree optimization, 211

\( A_{18} \) – construction of POPs for element partition trees, 226

\( A_{1} \) – construction of normal representation for the set of POPs, 16

\( A_{2} \) – fusion of sets of POPs, 18

\( A_{3} \) – construction of DAG \( \Delta_U, \alpha(T) \), 27

\( A_{4} \) – counting the number of decision trees, 40

\( A_{5} \) – procedure of decision tree optimization, 51

\( A_{6} \) – construction of POPs for decision trees, cost versus cost, 86

\( A_{7} \) – construction of POPs for decision trees, cost versus uncertainty, 106

\( A_{8} \) – procedure of decision rule optimization, 133

\( A_{9} \) – counting the number of decision rules, 136

Average minimum distance, 95

Average relative difference, 95, 162

B

Basic numerical operation, 24

Binary search trees, 246

cost function, 246

average weighted depth, 247

increasing, 247

strictly increasing, 247

weighted depth, 247

multi-stage optimization, 251

procedure of optimization, 248

Boolean function, 55

conjunction, 58

decision table representation, 55

exhaustive, 65

linear, 58, 66

majority, 58

monotone, 67

partial, 55

total, 55

unate, 66

C

CART-like decision trees, 116

algorithm \( A_{\text{DAG}} \) for directed acyclic graph construction, 116

algorithm \( A_{\text{POPs}} \) for construction of Pareto optimal points, 117

binary split, 115

best, 116
Corner point, 98
Bresenham circle of radius three, 98
Cost function for decision rules, 126
coverage $c$, 126
length $l$, 126
miscoverage $mc$, 127
modified coverage $c_M$, 127
relative coverage $rc$, 127
relative miscoverage $rmc$, 127
strictly increasing, 126
Cost function for decision trees, 44
average depth $h_{avg}$, 45
bounded, 45
depth $h$, 45
increasing, 44
integral, 44
number of nodes $L$, 45
number of nonterminal nodes $L_n$, 45
number of terminal nodes $L_t$, 45
strictly increasing, 44
total path length $tpl$, 45
Cost function for element partition trees, 199
ASAP mode, 194, 200
KB mode, 194, 200
increasing, 200
memory memory$p_{mode}$, 201
parallel time $time^p_{node}$, 201
polynomial order $p$, 200
sequential time $time^{1,p,q}_{node}$, 201
strictly increasing, 200
time of one memory transfer operation relative to the cost of performing an arithmetic operation $q$, 200

D
Dagger software system, 269
Decision rule
$(U, \alpha)$-decision rule for decision table, 126
$(U, \alpha)$-decision rule for decision table and row, 126
$U$-decision rule for decision table, 126
$U$-decision rule for decision table and row, 126
for decision table, 126
for decision table and row, 126
over decision table, 125
totally optimal, 136, 143
uncertainty, 126
Decision table, 23
complete, 143
conditional attribute, 23
decision attribute, 23
degenerate, 24
diagnostic, 24
empty, 24
most common decision, 24
over information system, 28
separable subtable, 24
size, 24
subtable, 24

Decision tree, 35
$(U, \alpha)$-decision tree for decision table, 38
$U^{max}$-decision tree for decision table, 42
$U^{sum}$-decision tree for decision table, 42
for decision table, 37
irredundant, 42, 43
nondeterministic, 148
optimal, 50
over decision table, 36
pruning of subtree, 42
strictly optimal, 50
totally optimal, 54
Directed acyclic graph for CART-like decision trees, 116
Directed acyclic graph for decision table, 27
bundle of edges, 27
labeled proper subgraph, 136
proper subgraph, 28
terminal node, 28
Directed acyclic graph for mesh, 207
internal node, 208
proper subgraph, 208
terminal node, 208

E
Element partition tree, 198
optimal, 210
strictly optimal, 210
totally optimal, 215

F
Function
increasing, 17, 43
strictly increasing, 20, 43
Function of $k$-valued logic, 145

G
Greedy algorithm $A_I$ for decision tree construction, 92
impurity function, 92
type, 92
Greedy heuristic $A_{\text{greedy}}$ for decision rule construction, 161
heuristic point, 162

I
Iteration-free circuit, 67
constant fault, 67
monotone basis, 67
problem of diagnosis, 67

K
$k$-valued information system, 28
$r$-restricted, 29
attribute, 28
decision table, 28
linear, 30
clone, 30
restricted, 29
system of equations, 29
irreducible, 29

L
Local misclassification error rate, 112
LU factorization, 192
$ASAP$ mode, 194
$KB$ mode, 194
Doolittle algorithm, 192
fully assembled row, 193
Schur complement, 193

M
Majority problem, 77
adaptive algorithm, 77
modified, 78
non-adaptive (oblivious) algorithm, 77
Matrix chain multiplication, 237
cost function, 238
increasing, 238
strictly increasing, 238
multi-stage optimization, 242
procedure of optimization, 239
Mesh, 195
angle-singular $A_k$, 216
boundary side, 195
corner, 195
dividing line, 196
edge-singular $E_k$, 216
mesh line, 196
boundary line, 196
common border side, 197
distinct point, 197
division, 197
maximal, 196
mesh point, 196
point-singular $P_k$, 216
size, 198
submesh, 196
border side, 197
unitary, 197
Misclassification error rate, 111
Multi-pruning for decision trees, 113
Multiset, 16

P
Pareto optimal point (POP), 15
normal representation of the set of POPs, 16
Partial order on the plain, 15
antichain, 15
comparable points, 15
Problem of sorting, 75

R
Reduct, 81
cost function, 238
increasing, 238
strictly increasing, 238
multi-stage optimization, 242
procedure of optimization, 239
Matrix chain multiplication, 237

S
Sequence alignment, 254
cost function, 254
multi-stage optimization, 257
procedure of optimization, 255
System of decision rules for decision table, 128
cost function, 128
uncertainty, 128
<table>
<thead>
<tr>
<th>Term</th>
<th>Page(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>U</strong></td>
<td>138</td>
</tr>
<tr>
<td>$(U, \alpha, \beta)$-greedy set of decision rules for decision table</td>
<td></td>
</tr>
<tr>
<td>Uncertainty measure</td>
<td>24</td>
</tr>
<tr>
<td>entropy $\text{ent}$</td>
<td>24</td>
</tr>
<tr>
<td>function $R$</td>
<td>24</td>
</tr>
<tr>
<td>Gini index $\text{gini}$</td>
<td>24</td>
</tr>
<tr>
<td>increasing</td>
<td>25</td>
</tr>
<tr>
<td>misclassification error $\text{me}$</td>
<td>24</td>
</tr>
<tr>
<td>relative misclassification error $\text{rme}$</td>
<td>24</td>
</tr>
<tr>
<td>strictly increasing</td>
<td>25</td>
</tr>
<tr>
<td>sum-increasing</td>
<td>26</td>
</tr>
</tbody>
</table>