To appear in IEEE Expert
Spec. Issue on Feature Transformation and Subset Selection
(H. Liu, M. Motoda, eds)

Feature transformation by function decomposition

Blaž Zupan, Marko Bohanec, Janez Demšar, Ivan Bratko

While not explicitly intended for feature transformation, some methods for switching circuit design implicitly deal with this problem. Given a tabulated Boolean function, these methods construct a circuit that implements that function. In 1950s and 1960s, Ashenhurst [1] and Curtis [2] proposed a *function decomposition* method that develops a switching circuit by constructing a nested hierarchy of tabulated Boolean functions. Both the hierarchy and the functions themselves are discovered by the decomposition method and are not given in advance. This is especially important from the viewpoint of feature construction, since the outputs of such functions can be regarded as new features not present in the original problem description.

The basic principle of function decomposition is the following. Let a tabulated function $y = F(X)$ use a set of input features $X = x_1, \ldots, x_n$. The goal is to decompose this function into $y = G(A, H(B))$, where A and B are subsets of features in X such that $A \cup B = X$. G and H are tabulated functions that are determined by the decomposition and are not predefined. Their joint complexity (determined by some complexity measure) should be lower than the complexity of F. Such a decomposition also discovers a new feature $c = H(B)$. Since the decomposition can be applied recursively on H and G, the result in general is a hierarchy of features. For each feature in the hierarchy, there is a corresponding tabulated function (such as $H(B)$) that determines the dependency of that feature on its immediate descendants in the hierarchy.

Ashenhurst-Curtis decomposition was intended for switching circuit design of completely specified Boolean functions. Recently, Wan and Perkowski [3] extended the decomposition to handle incompletely specified Boolean functions. In the framework of feature extraction, Ross et al. [4] used a set of simple Boolean functions to show the decomposition's capability to discover and extract useful features. This article presents an approach to feature transformation that is based on the extension of the function decomposition method by Zupan et al. [5]. This method allows the decomposition to deal with functions that involve nominal (i.e., not necessarily binary) features, and is implemented in a system called HINT(Hierarchy INduction Tool).

1 Single-step function decomposition

The core of the decomposition algorithm is a *single-step decomposition* which decomposes a tabulated function $y = F(X)$ into two possibly less complex tabulated functions G and H, so that $y = G(A, c)$ and $c = H(B)$. The resulting functions G and H have to be consistent with F. For the purpose of decomposition, a set of features X is partitioned to two disjoint subsets A and B, referred to as a *free* and *bound set*, respectively. For a given feature partition, single-step decomposition discovers a new feature c and a tabular representation of H and G.

For example, consider a function $y = F(x_1, x_2, x_3)$ as given in Table 1. The input features x_1, x_2, and the output feature y can take the values `lo`, `med`, `hi`; input feature x_3 can take the values `lo`, `hi`.

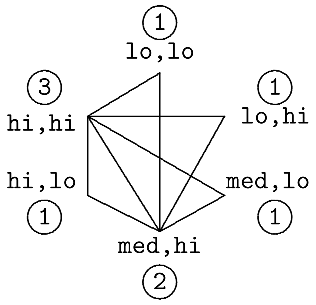
x_1	x_2	x_3	y
lo	lo	lo	lo
lo	lo	hi	lo
lo	med	lo	lo
lo	med	hi	med
lo	hi	lo	lo
lo	hi	hi	hi
med	med	lo	med
med	hi	lo	med
med	hi	hi	hi
hi	lo	lo	hi
hi	hi	lo	hi

Table 1: Tabulated function $y = F(x_1, x_2, x_3)$.

Suppose that we want to discover a description of a new feature $c = H(x_2, x_3)$. For this purpose, we represent F by a *partition matrix* that uses the values of x_1 for row labels, and the combinations of values of x_2 and x_3 for column labels (Figure 1.a). Partition matrix entries with no corresponding instance in the tabulated representation of F are denoted with "-" and treated as *don't-care*.

	x_2	lo	lo	med	med	hi	hi
x_1	x_3	lo	hi	lo	hi	lo	hi
lo		lo	lo	lo	med	lo	hi
med		-	-	med	-	med	hi
hi		hi	-	-	-	hi	-
c		1	1	1	2	1	3

(a)



(b)

Figure 1: Partition matrix (a) and the corresponding incompatibility graph (b) for the function from Table 1. The node labels of the incompatibility graph found by graph coloring are circled.

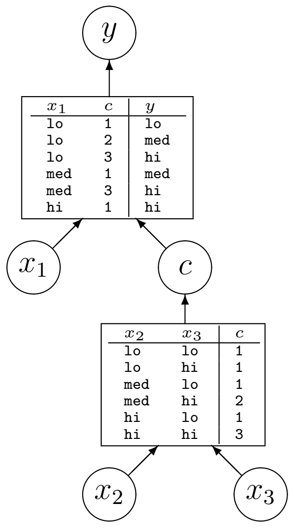
Each column in the partition matrix denotes the behavior of F for a specific combination of x_2 and x_3. Columns that have pairwise equal row entries or at least one row entry is a don't-care are called *compatible*. The decomposition has to assign each column a label that corresponds to a value of c. If two columns are compatible, they can be assigned the same label. To preserve the consistency, different labels have to be used for incompatible columns.

The partition matrix column labels are found by coloring an *incompatibility graph* (Figure 1.b). This has a distinct node for each column of the partition matrix. Two nodes are connected if the corresponding columns of partition matrix are incompatible. For instance, the nodes `hi,hi` and `lo,hi` are connected because their corresponding columns are incompatible due to the entries in the first row (`hi`$\neq$`lo`).

With optimal coloring of the incompatibility graph, the new feature c obtains a minimal set of values needed for consistent derivation of H and G from F. The optimal coloring of the graph from Figure 1.b requires three different colors, i.e., three abstract values for c. The tabulated functions G and H can then be derived straightforwardly from the labeled partition matrix.

The resulting decomposition is given in Figure 2. The following can be observed:

- The tabulated functions G and H are overall smaller than the original function F.



x_1	c	y
lo	1	lo
lo	2	med
lo	3	hi
med	1	med
med	3	hi
hi	1	hi

x_2	x_3	c
lo	lo	1
lo	hi	1
med	lo	1
med	hi	2
hi	lo	1
hi	hi	3

Figure 2: Decomposition of tabulated function from Table 1 using a new feature $c = H(x_2, x_3)$.

- By combining the features x_2 and x_3 we obtained a new feature c and the corresponding tabulated function H, which can be interpreted as $c = \mathrm{MIN}(x_2, x_3)$. Similarly, the function G that relates x_1 and c to y can be interpreted as $y = \mathrm{MAX}(x_1, c)$.

- The decomposition generalizes some undefined entries of the partition matrix. For example, $F(\texttt{hi}, \texttt{lo}, \texttt{hi})$ is generalized to `hi` because the column `lo,hi` has the same label as columns `lo,lo` and `hi,lo`.

2 Finding the best feature partition

So far, we have assumed that a partition of the features to free and bound sets is given. However, for each function F there are many possible partitions, each one yielding a different feature c and a different pair of functions G and H. In feature transformation, it is important to identify partitions that lead to simple but useful new features. Typically, we prefer features with a small number of values, and those that yield functions G and H of low complexity.

The simplest measure for finding good feature partitions is the number of values required for the new feature c. That is, when decomposing F, a set of candidate partitions is examined and the one that yields c with a smallest set of possible values is selected for decomposition.

An alternative measure for the selection of partitions can be based on the complexity of functions. Let $I(F)$ denote a number of bits to encode some function F [6, 4]. Then, the best partition is the one that minimizes the overall complexity of newly discovered functions H and G, i.e., the partition with minimal $I(H) + I(G)$. The complexity of G and H should be lower than that of F, and decomposition takes place only if $I(H) + I(G) < I(F)$ for the best partition.

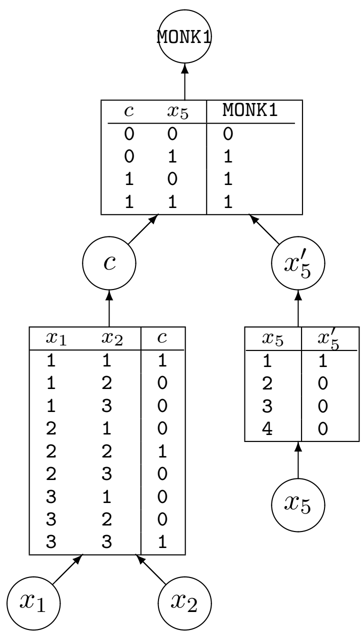
The number of possible partitions increases exponentially with the number of input features. To limit the complexity of search, the decomposition should examine only a subset of partitions. In HINT, the partitions examined are only those with less than b features in the bound set, where b is a user-defined parameter usually set to 2 or 3.

The partition matrix can be very large. However, it is never explicitly represented in the program as HINT derives the incompatibility graph directly from the data [6]. Partition matrix is thus a formal construct used for explanation or analysis of the decomposition algorithm. The coloring of the incompatibility graph is another potential bottleneck. HINT uses an efficient heuristic algorithm for graph coloring [6].

3 Redundancy discovery and removal

A special characteristic of function decomposition is its capability to identify and remove redundant features and/or their values. Namely, whenever a feature $c = H(x_i)$ is discovered such that c uses only a single value (i.e., H is a constant function), the feature x_i is redundant and can be removed from the dataset. Similarly, whenever a feature $c = H(x_i)$ uses less values than x_i, the original feature x_i can be replaced by c.

Both these dataset transformations are particularly useful for data preprocessing, since they can reduce the size of the problem and increase the coverage of the problem space by the learning examples. HINT removes redundant features in the increased order of their relevance: the less relevant features are checked first, and if any type of redundancy mentioned above is discovered, the dataset is accordingly transformed. The features are estimated by a ReliefF measure of relevance [7].



c	x_5	MONK1
0	0	0
0	1	1
1	0	1
1	1	1

x_1	x_2	c
1	1	1
1	2	0
1	3	0
2	1	0
2	2	1
2	3	0
3	1	0
3	2	0
3	3	1

x_5	x'_5
1	1
2	0
3	0
4	0

Figure 3: MONK1 feature hierachy as discovered by HINT.

4 Discovering feature hierarchies

With all the above ingredients, decomposition enables the discovery of feature hierarchies. Given a dataset, it is first pre-processed to remove redundant features and their values. Next, a single-step decomposition is used to decompose the pre-processed tabulated function F to G and H. This process is recursively repeated on G and H until they can not be decomposed further, i.e., their further decomposition would increase the overall complexity of resulting functions.

Let us illustrate this process with several examples. The first example is a well-known machine learning problem MONK1 [8]. The dataset uses six 2 to 4-valued input features (x_1 to x_6) and contains 124 (of 435 possible) distinct learning examples that partially define the target concept $x_1 = x_2$ OR $x_5 = 1$. HINT develops a feature hierarchy (Figure 3) that

1. correctly excludes irrelevant input features x_3, x_4, and x_6,

2. transforms x_5 to x'_5 by mapping four values of x_5 to only two values of x'_5,

3. includes a new feature c and its tabular representation for $x_1 = x_2$, and
4. relates c and x'_5 with a tabular representation of the OR function.

In other words, the resulting hierarchy correctly represents the target concept.

Similarly as MONK1, the MONK2 learning problem [8] uses the same set of input features but defines the concept: $x_i = 1$ for exactly two choices of $i \in \{1, 2, \ldots, 6\}$. The dataset contains 172 (of 435 possible) distinct learning examples. Although the discovered structure (Figure 4) does not directly correspond to the original concept definition, it correctly reformulates the target concept by introducing features that count the number of ones in their arguments. Also note that all input features that use more than two values are replaced by new binary features.

Figure 4: The feature hierarchy discovered for MONK2. Each node gives a name of the feature and cardinality of its set of values.

For a real-world example, consider a HOUSING dataset from a management decision support system for allocating housing loans. This system was developed for the Housing Fund of Slovenia and used since 1991 in 13 floats of loans with a total value of approximately 90 million ECU. Basically, its task is to rank applicants into one of nine priority classes based on 12 input features. The system uses a hierarchical decision model.

For this experiment, we wanted to assess the ability of decomposition to reconstruct the system's model given a random selection of classified cases.

Figure 5: The feature structure discovered for housing loan allocation problem. Each node gives a name of the feature and cardinality of its set of values.

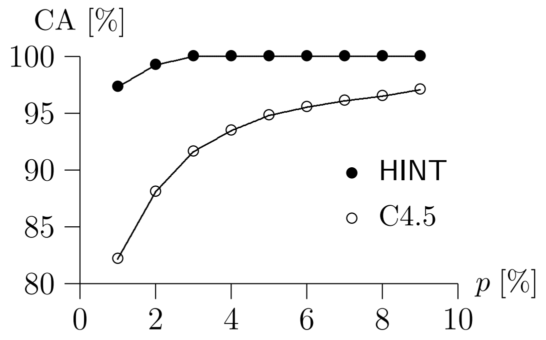


Figure 6: Averaged classification accuracy of HINT and C4.5 for 10 experiments with learning sets consisting of a random sample of p percents of the housing problem space.

First, it was observed that in most cases when HINT was given 8000 or more learning examples (4% coverage of problem space), it discovered the same feature hierarchy (Figure 5). When presented to a domain expert, the discovered features were found meaningful. For instance, the feature c_4 represents the present housing status of the applicant. Features c_2, c_6, and c_8 respectively specify the way of solving the housing problem, applicant's current status, and his/her social and health conditions. Thus, the structure of the original model was successfully reconstructed. This reconstruction takes about 2.5 minutes of processor time on a HP J210 Unix workstation.

The quality of the functions discovered for housing problem was then assessed by classification of the remaining examples that completely cover this domain but were not included in the learning set. The generalization of HINT was assessed as a learning curve (Figure 6) and compared to that of the state-of-the-art learning tool C4.5 [9] that induces decision trees. For this domain, HINT generalizes well as it achieves 100% classification accuracy by using learning sets that cover 3 or more percents of the problem space.

Housing problem domain can be viewed as a typical representative of practical domains in the field of decision making that HINT was tested on. Similar tests and experimental conclusion were for instance obtained by re-discovery of hierarchical decision models for nursing school applications, job application evaluation, and performance evaluation of public enterprises [6].

5 Feature construction

Function decomposition can also be used for feature construction. This is defined as a process of augmenting the feature space by inferring or creating additional features. We here show how the single-step decomposition can be used both for finding appropriate combinations of original features and using them to construct new features which are then added to the original dataset. Again, the new features that use small number of values are preferred.

For example, consider again the dataset from Table 1. Suppose we want to augment it with a single new feature. For this purpose, single-step decomposition examines all pairs of original input features and for each pair derives a new feature. A new feature that has the fewest values is preferred. Such feature is $c = H(x_2, x_3)$, and is defined by the example set from Figure 2. Function H can now be used for each example from Table 1 to obtain the corresponding

x_1	x_2	x_3	c	y
lo	lo	lo	1	lo
lo	lo	hi	1	lo
lo	med	lo	1	lo
lo	med	hi	2	med
lo	hi	lo	1	lo
lo	hi	hi	3	hi
med	med	lo	1	med
med	hi	lo	1	med
med	hi	hi	3	hi
hi	lo	lo	1	hi
hi	hi	lo	1	hi

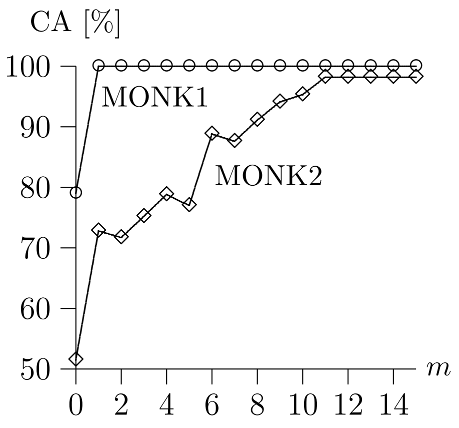
Table 2: A dataset from Table 1 augmented with a new feature $c = H(x_2, x_3)$.

value for c; such augmented dataset is shown in Table 2.

Obviously, an original dataset can be augmented with more than just a single new feature. We propose a schema where, using the single-step decomposition, m new features are added to a dataset: a candidate set of combinations of the original features are examined, and corresponding m new features that have the fewest values are selected. Ties are resolved arbitrarily. Only original features are used to generate new features. Note also that in the process no hierarchy of features is constructed. The feature construction and corresponding augmentation of the original dataset results in a new dataset which may then be further used by some machine learning algorithm.

We evaluate this schema on the above-mentioned domains. For new features, only those that depend only on two original input features are considered. For MONK1 and MONK2 domains, the benefit of adding new features was assessed by a 10-fold cross validation. For HOUSING, 10 experiments were performed with a learning set consisting of a random sample of 1% of the problem space, while the test set consisted of the remaining 99%. Each learning set was first augmented with m new features as described above, and then given to C4.5 [9] which was used to induce the target concept from the augmented dataset. The induced decision tree was then tested on a test set, which was also augmented by the same set of new features, i.e., using their definition as derived from the learning set.

In all the cases, a considerable increase of classification accuracy (Figure 7) and considerable decrease of the sizes of induced decision trees (Figure 8) occur when new features are added. In other words, adding new features improves



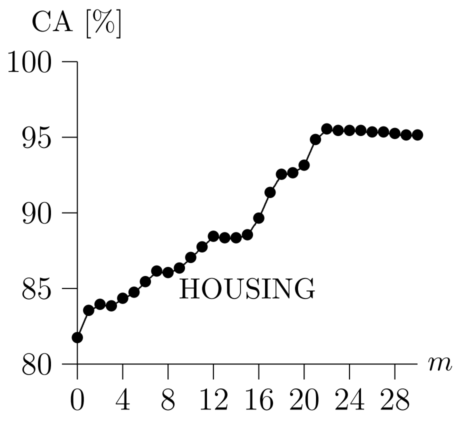
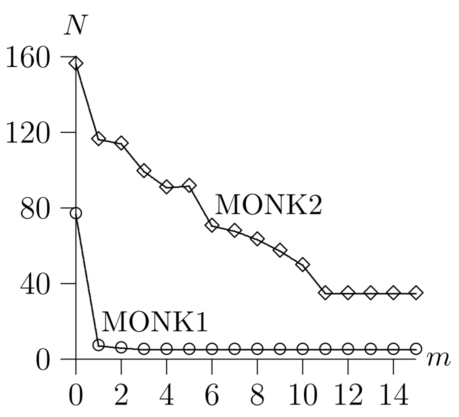


Figure 7: Classification accuracy of C4.5 as a function of number of new features m added to MONK1 and MONK2 (a) and housing loans dataset (b).



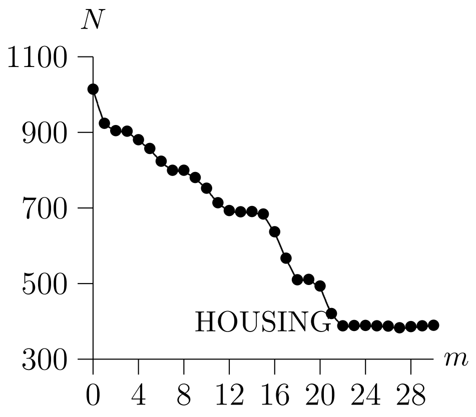


Figure 8: Number of nodes N of C4.5-induced decision trees as a function of number of new features m added to MONK1 and MONK2 (a) and housing loans dataset (b).

both the classification accuracy and the transparency of decision trees. The results indicate that function decomposition discovers features that are relevant for these domains. It is further interesting to note how well a simple feature selection criterion based on the number of new feature's values performs in terms of yielding augmented datasets that enable C4.5 to derive classifiers with high classification accuracies.

6 Further work

In the framework of feature transformation, function decomposition is a promising approach to discover and construct new features that are either added to the original dataset, or transform this dataset to a hierarchy of less complex datasets. The function decomposition algorithm in HINT performs a kind of generalization, so it can also be viewed as a machine learning algorithm. The approach described in this article is limited to consistent datasets and nominal features. It is therefore desired to extend the approach to discover new features from noisy data, and from data that uses continuous features.

To handle noisy data, a minimal-error decomposition was recently proposed [6]. It is based on a representation of learning examples with class distributions and uses successive column merging of partition matrix, so that the expected error of classification is minimized. So far, the method was evaluated only as a machine learning tool, while further work is needed to assess its appropriateness for feature transformation problems.

References

[1] R. L. Ashenhurst, "The decomposition of switching functions," tech. rep., Bell Laboratories BL-1(11), pages 541–602, 1952.

[2] H. A. Curtis, *A New Approach to the Design of Switching Functions.* Van Nostrand, Princeton, N.J., 1962.

[3] W. Wan and M. A. Perkowski, "A new approach to the decomposition of incompletely specified functions based on graph-coloring and local transformations and its application to FPGA mapping," in *Proc. of the IEEE EURO-DAC '92*, (Hamburg), pp. 230–235, 1992.

[4] T. D. Ross, M. J. Noviskey, D. A. Gadd, and J. A. Goldman, "Pattern theoretic feature extraction and constructive induction," in *Proc. ML-COLT '94 Workshop on Constructive Induction and Change of Representation*, (New Brunswick, New Jersey), July 1994.

[5] B. Zupan, M. Bohanec, I. Bratko, and J. Demšar, "Machine learning by function decomposition," in *Proc. Fourteenth International Conference on Machine Learning* (J. D. H. Fisher, ed.), (San Mateo, CA), pp. 421–429, Morgan Kaufmann, 1997.

[6] B. Zupan, *Machine learning based on function decomposition.* PhD thesis, University of Ljubljana, Apr. 1997. Available at `http://www-ai.ijs.si/BlazZupan/papers.html`.

[7] I. Kononenko, "Estimating attributes," in *Proc. of the European Conference on Machine Learning (ECML-94)* (F. Bergadano and L. D. Raedt, eds.), vol. 784 of *Lecture Notes in Artificial Intelligence*, (Catania), pp. 171–182, Springer-Verlag, 1994.

[8] S. B. Thrun et al., "A performance comparison of different learning algorithms," tech. rep., Carnegie Mellon University CMU-CS-91-197, 1991.

[9] J. R. Quinlan, *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers, 1993.

Blaž Zupan is a research assistant at Jožef Stefan Institute, Ljubljana, Slovenia. His research interest include machine learning, intelligent data analysis and knowledge discovery in databases, computer-aided support for medical decision making, and medical informatics. He has his MS in computer science from University of Houston, Texas, and PhD from University of Ljubljana, Slovenia. He can be reached at Department of Intelligent Systems, J. Stefan Institute, Jamova 39, SI-1000 Ljubljana, Slovenia; blaz.zupan@ijs.si; http://www-ai.ijs.si/BlazZupan.

Marko Bohanec is a research associate at Jožef Stefan Institute, Ljubljana, Slovenia, and assistant professor in information systems at the Faculty of Organizational Sciences, University of Maribor. His research and development interests are in decision support systems and machine learning. He obtained his PhD in Computer Science at the Faculty of Electrical Engineering and Computer Science, University of Ljubljana. He has published in journals such as *Machine Learning*, *Acta Psychologica* and *Information & Management.* Contact him at Department of Intelligent Systems, J. Stefan Institute, Jamova 39, SI-1000 Ljubljana, Slovenia; marko.bohanec@ijs.si; http://www-ai.ijs.si/MarkoBohanec/mare.html.

Ivan Bratko is a professor of computer science at the Faculty of Computer and Information Science, University of Ljubljana, Slovenia. He heads the AI laboratories at Jožef Stefan Institute and the University. He is the chairman of ISSEK, International School for the Synthesis of Expert Knowledge. He has conducted research in machine learning, knowledge-based systems, qualitative modeling, intelligent robotics, heuristic programming and computer chess. His main interests in machine learning have been in learning from noisy data, combining learning and qualitative reasoning, and various applications of machine learning and Inductive Logic Programming, including medicine, ecological modeling and control of dynamic systems. He is the author of widely adopted text *PROLOG Programming for Artificial Intelligence* (Addison-Wesley 1986, second edition 1990). He co-edited (with N. Lavrač) *Progress in Machine Learning* (Sigma Press 1987) and co-authored (with I. Mozetič and N. Lavrač) *KARDIO: a Study in Deep and Qualitative Knowledge for Expert Systems* (MIT Press, 1989). Contact him at the Faculty of Computer and Information Science, University of Ljubljana, Tržaška 25, SI-1000 Ljubljana, Slovenia; ivan.bratko@fri.uni-lj.si.

Janez Demšar is a research assistant at the Faculty of Computer and Information Science, University of Ljubljana, Slovenia. He has BSc in computer science from the same faculty. His research interest are machine learning and intelligent data analysis. He can be reached at the Faculty of Computer and Information Science, University of Ljubljana, Tržaška 25, SI-1000 Ljubljana, Slovenia; janez.demsar@fri.uni-lj.si; http://ai.fri.uni-lj.si/janezd.