Statistics and Computing (1997) **7**, 19–34

Decomposing Bayesian networks: triangulation of the moral graph with genetic algorithms

PEDRO LARRAÑAGA[1], CINDY M. H. KUIJPERS[2], MIKEL POZA[1] and ROBERTO H. MURGA[1]

[1]*Department of Computer Science and Artificial Intelligence, University of the Basque Country, PO Box 649, 20080 San Sebastián, Spain*
[2]*Department of Applied Mathematics, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands*

Received August 1995; accepted June 1996

In this paper we consider the optimal decomposition of Bayesian networks. More concretely, we examine empirically the applicability of genetic algorithms to the problem of the triangulation of moral graphs. This problem constitutes the only difficult step in the evidence propagation algorithm of Lauritzen and Spiegelhalter (1988) and is known to be NP-hard (Wen, 1991). We carry out experiments with distinct crossover and mutation operators and with different population sizes, mutation rates and selection biasses. The results are analysed statistically. They turn out to improve the results obtained with most other known triangulation methods (Kjærulff, 1990) and are comparable to results obtained with simulated annealing (Kjærulff, 1990; Kjærulff, 1992).

Keywords: Bayesian networks, genetic algorithms, optimal decomposition, graph triangulation, moral graph, NP-hard problems, statistical analysis

1. Introduction

Bayesian networks constitute a reasoning method based on probability theory. A Bayesian network consists of a set of nodes and a set of arcs which together constitute a directed acyclic graph (DAG). The nodes represent random variables, all of which, in general, have a finite set of states. The arcs indicate the existence of direct causal connections between the linked variables, and the strengths of these connections are expressed in terms of conditional probabilities.

For determining the joint probability distribution $P(x_1, \ldots, x_n)$ in a Bayesian network, it is sufficient to know the conditional probabilities $P(x_i \mid \pi_{X_i})$, where π_{X_i} is the *parent set* of variable X_i, i.e. the set of variables by which X_i is affected:

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid \pi_{X_i}).$$

Excellent introductions on Bayesian networks can be found in Pearl (1988), Neapolitan (1990) and Jensen (1996).

One of the best-known problems, in the context of Bayesian networks, is related to the propagation of evidence. It consists of the assignment of probabilities to the values of the rest of the variables, once the value of some variables are known. Cooper (1990) demonstrated that this problem is NP-hard. Dagum and Luby (1993) proved that even the problem of finding approximate solutions belongs to the class of NP-hard problems. Fundamentally, the problem of the propagation of evidence has been tackled in two different ways:

- With *exact* (or *deterministic*) *algorithms*. Most noteworthy are the methods of Pearl (1986), Shachter (1988) and Lauritzen and Spiegelhalter (1988). Jensen (1994) improved aspects of the algorithm which was proposed in Lauritzen and Spiegelhalter (1988).
- With *approximate algorithms*, based on a simulation of the corresponding Bayesian network. We mention the algorithms introduced by Chavez and Cooper (1990), Dagum and Horvitz (1993), Fung and Chang (1990), Henrion (1988), Hryceij (1990), Jensen *et al.* (1993), Pearl (1987), Shachter and Peot (1990), and Shwe and Cooper (1991).

BEGIN Triangulation
{ Given:
moral graph $G = (V, E)$, $V = \{v_1, v_2, \ldots, v_k\}, |E| < \infty$,
n_j, the number of states of v_j($j = 1, \ldots, k$), with $n_j < \infty$.
node elimination sequence: x_i is the ith node to eliminate
$(i = 1, \ldots, k)$.
}
$i = 1$;
WHILE there is a node to eliminate
{ Let $Adj(x_i)$ be the set of nodes adjacent to x_i;
REPEAT
{ Select two nodes in $Adj(x_i)$ that are not connected
and add the edge between them;
}
UNTIL $Adj(x_i)$ is a complete subgraph;

$$\text{Clique } C_i = \begin{cases} \emptyset & \text{if } \exists_{j<i}\, x_i \cup Adj(x_i) \subseteq C_j, \\ x_i \cup Adj(x_i) & \text{otherwise;} \end{cases}$$

Delete node x_i and all its incident edges;
$i := i + 1$;
}
{ Output:
$G_{\text{triangulated}} = (v, E \cup E')$ where E' is the set of added edges:
triangulated graph.
Weight of $G_{\text{triangulated}}$: $\log_2 \sum_C \prod_{v_i \in C} n_i$.
}
END Triangulation.

Fig. 1. *Pseudo-code of the basic triangulation method*

We take as point of departure the evidence propagation algorithm proposed by Lauritzen and Spiegelhalter (1988). The first step of this algorithm consists of the *moralization* of the network structure. This means that all variables with a common child are linked, after which all directions on the arcs are deleted. The resulting graph is called a *moral graph.* The second step of the algorithm of Lauritzen and Spiegelhalter is the so-called *triangulation* of the moral graph. A graph is *triangulated* if any cycle of length greater than 3 has a chord. The remaining steps of the algorithm are of no importance for our paper.

We consider the problem of the triangulation, since in the algorithm of Lauritzen and Spiegelhalter, as can be read in Jensen (1996), the

> ... only problematic step ... is the triangulation. Since any elimination sequence will produce a triangulation it may not seem as a problem, but for the propagation algorithm it is. In probability propagation the cliques in the junction graph shall have joint probability tables attached to them. The size of the table is the product of the number of states of the variables. So, the size increases exponentially with the size of the clique. A good triangulation, therefore, is a triangulation yielding small cliques; or to be more precise, yielding small probability tables.

Wen (1991) demonstrated that the search for an optimal triangulation is NP-hard.

The basic technique for triangulating a moral graph is described in Fig. 1. Notation and terminology are defined in Section 2. There we see that the quality of a triangulation is completely determined by the order in which the nodes are eliminated, where the elimination of a node v (see also Fig. 2) consists of adding edges to the graph in such a way that all nodes adjacent to v become pairwise adjacent, and subsequently deleting v and its adjacent edges. Hence, the search for an optimal triangulation is equivalent to the search for an optimal node elimination sequence. The cliques obtained during the triangulation define a decomposition of the Bayesian network. Depending on the problem to be solved by the graph triangulation, optimality can be defined in different ways. The most frequently used optimality criteria are:

- the *minimum fill* criterion, i.e. minimize $|T|$.
- the *minimum size* criterion, i.e. minimize $s(G^{\mathrm{t}}_{\#})$.
- the *minimum weight* criterion, i.e. minimize $w(G^{\mathrm{t}}_{\#})$.

Since our interest in graph triangulations originates from the evidence propagation algorithm of Lauritzen and Spiegelhalter (1988), we want to obtain Bayesian network decompositions with small probability tables. Therefore our objective is to obtain triangulations of minimum weight.

Kjærulff (1990) performed an empirical comparison of triangulation methods which were proposed by various authors: *simple random elimination*, *maximum cardinality search* (Tarjan and Yannakakis, 1984) which continuously selects a vertex to be ordered next, with the highest number of ordered neighbours; *lexicographic search* (Rose *et al.*, 1976) which does something similar to the maximum cardinality search, but in addition it is guaranteed to produce minimal triangulations; *extended random elimination* (Fujisawa and Orino, 1974) which does not necessarily add fill edges between all non-adjacent vertices in the adjacency set of a vertex being eliminated; the FMINT algorithm applied to the triangulations obtained by simple random elimination and maximum cardinality search (Kjærulff, 1990); the three heuristical algorithms *minimum size*, *minimum fill* and *minimum weight* (Kjærulff, 1990) each of them successively chooses the next vertex to be eliminated as the one that produces the smallest clique, the fewest fill edges as possible, or the clique with least weight, respectively; and *simulated annealing* (Kjærulff, 1990; Kjærulff, 1992). So far, the best results have been obtained with simulated annealing (Kjærulff, 1990; Kjærulff, 1992; Wen, 1990).

We choose to tackle the graph triangulation problem with *genetic algorithms.* These algorithms are, like simulated

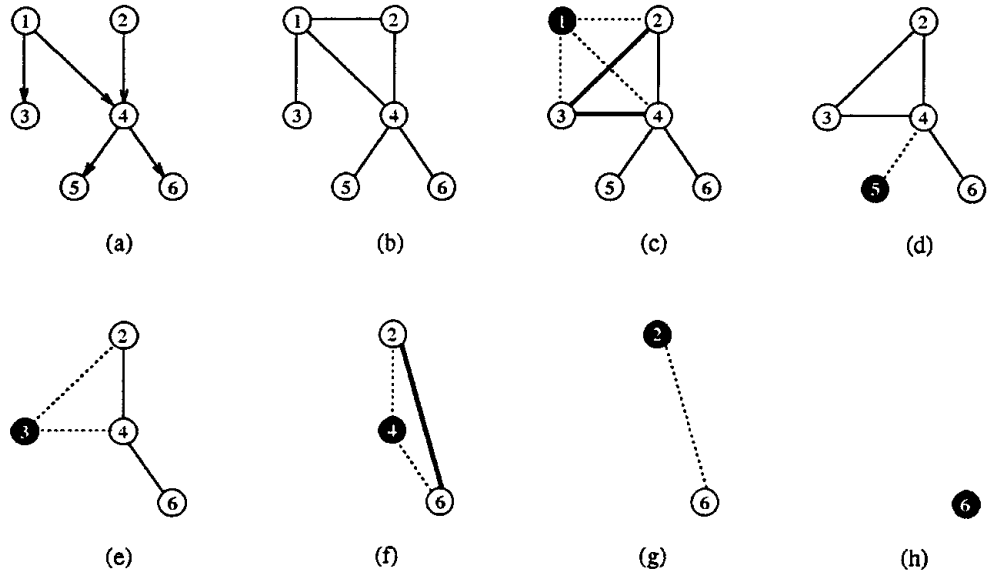

Fig. 2. *Example of a graph moralization and of a triangulation of the resulting moral graph.*(*a*) *DAG*; (*b*) *moral graph. Suppose that the nodes are eliminated in order*: v_1, v_5, v_3, v_4, v_2, v_6. Let $n_i = i + 1 (i = 1, 2, \ldots, 6)$; (*c*) *elimination* v_1: $C_1 = \{v_1, v_2, v_3, v_4\}$, *added edges*: $\{v_2, v_3\}, \{v_3, v_4\}$; (*d*) *elimination* v_5: $C_2 = \{v_4, v_5\}$; (*e*) *elimination* v_3: $C_3 = \emptyset$; (*f*) *elimination* v_4: $C_4 = \{v_2, v_4, v_6\}$, *added edge*: $\{v_2, v_6\}$; (*g*) *elimination* v_2: $C_5 = \emptyset$; (*h*) *elimination* v_6: $C_6 = \emptyset$. *Total weight of the triangulated graph*: $\log_2(2 \cdot 3 \cdot 4 \cdot 5 + 5 \cdot 6 + 3 \cdot 5 \cdot 7) = \log_2 255$

annealing, based on natural optimization processes. In recent years, they have been applied to a large variety of combinatorial optimization problems. Larrañaga *et al.* (1993) studied the relation between the genetic algorithms and simulated annealing. For other applications of genetic algorithms to combinatorial optimization problems related to Bayesian networks, see Larrañaga *et al.* (1996a; 1996b; 1996c).

The structure of the paper is as follows. We start by explaining our notation and terminology in Section 2. A brief introduction on genetic algorithms is given in Section 3. In Section 4, we show that the problem of the search for an optimal graph triangulation has many similarities to the travelling salesman problem (TSP) and that therefore we can make use of some results of the research already carried out on the TSP. The experiments performed are described in Section 5. In Section 6, the results of the experiments are presented and a statistical analysis of them is carried out. Conclusions and suggestions for further research are given in Section 7. Finally, in the appendix the genetic operators that are used for the experiments are described. Many of these operators were originally developed for the TSP.

2. Notation and terminology

A *graph* G consisting of a finite set of vertices $V = \{v_1, v_2, \ldots, v_n\}$ and a finite set of edges E is denoted by $G = (V, E)$. G is a *belief* graph if a finite set of elements, called *states*, is attached to each vertex. For all vertices $v_k \in V$ we denote its number of states by n_k $(k = 1, \ldots, n)$. Following Kjærulff (1990), we define a (connected) graph $G = (V, E)$ to be *sparse* if $n - 1 \leq |E| \leq 1/10\,(n^2 - n)/2$, and to be *dense* in case $|E| \geq 1/4(n^2 - n)/2$.

An *ordering of* V is a bijection $\# : V \leftrightarrow \{1, 2, \ldots, n\}$. $G_\#$ is an ordered graph. The base-2 logarithm of the number of states of a vertex v_k is denoted by $w(v_k)$ and is called the *weight* of vertex v_k.

Two vertices $v_k, v_l \in V$ are *adjacent* (or *neighbours*) if $\{v_k, v_l\} \in E$.

The set

$$\mathrm{adj}(v_k, G) = \{v_l \in V \mid \{v_k, v_l\} \in E\}$$

is the set of vertices adjacent to v_k in G.

The set of vertices *monotonely adjacent* to v_k in $G_\#$ is defined by

$$\mathrm{madj}(v_k, G_\#) = \mathrm{adj}(v_k, G) \cap \{v_l \in V \mid \#(v_k) < \#(v_l)\}$$

A *complete subgraph* of G is a set of vertices of G of which all are pairwise adjacent.

A *clique* is a maximal complete subgraph of G.

A vertex $v_k \in V$ is said to be *eliminated* if edges are added to G until the adjacency set of v_k is a complete subgraph of G after which v_k and its incident edges are removed from G.

Let $G_{i,\#} = (V_i, E_i)$ denote the ordered graph obtained by eliminating the vertices of $G_\# = (V, E)$ in order $\#^{-1}(1), \ldots, \#^{-1}(i)$, where

$$G_{0,\#} = G,$$

$$V_i = \{v \in V \mid \#(v) > i\}, \quad V_0 = V, \quad V_n = \emptyset$$

$$E_i = \{\{v_k, v_l\} \in E_{i-1} \mid v_k, v_l \in V_i\} \cup \{\{v_k, v_l\} \mid v_k, v_l \in \mathrm{madj}(v, G_{i-1,\#}), \#(v) = i\},$$

$$\text{and } E_0 = E.$$

By $C_i(G_\#)$ we denote the clique, if any, produced in the elimination process of the vertex $\#^{-1}(i)$ from $G_{i-1,\#}$:

$$C_i(G_\#) = \begin{cases} c_i(G_\#) & \text{if } \forall_{j=1,2,\ldots,i-1}\ c_i(G_\#) \not\subseteq C_j(G_\#), \\ \emptyset & \text{otherwise,} \end{cases}$$

where $c_i(G_\#) = \{\#^{-1}(i)\} \cup \{\text{adj}(\#^{-1}(i), G_{i-1,\#})\}$.

A *path* in $G = (V, E)$ is a sequence of distinct vertices $\langle v_1, \ldots, v_m \rangle$ such that $\{v_k, v_{k+1}\} \in E$, for $k = 1, \ldots, m-1$. A *cycle* is a path $\langle v_1, \ldots, v_m \rangle$ with $v_1 = v_m$ and $m > 3$. A *chord* in a cycle is an edge $\{v_k, v_l\}$ between two non-consecutive vertices in the cycle. A graph G is *triangulated* if any cycle of length greater than 3 has a chord.

Let $G_\# = (V, E)$ be an ordered graph. Then $G_\#^t$ denotes the triangulated graph obtained by eliminating the vertices of G in the order defined by $\#$. $G_\#^t$ has vertex set V and edge set $E \cup T$, where the set of filled edges T is a triangulation. Let $C = \{v_1, \ldots, v_k\}$ be a clique of triangulated graph $G_\#^t$. Then, the *size of clique* C is $s(C) = k$ and we define the *weight* $w(C)$ of C as follows: $w(C) = \sum_{i=1}^{k} w(v_i)$.

The size of triangulated graph $G_\#^t$ is denoted by $s(G_\#^t)$ and it is defined as the sum of the sizes of all of its cliques: $s(G_\#^t) = \sum_C s(C)$. The *weight* $w(G_\#^t)$ of triangulated graph $G_\#^t$ is defined as

$$w(G_\#^t) = \log_2 \sum_C \prod_{v_i \in C} n_i = \log_2 \sum_C 2^{w(C)}.$$

3. Genetic algorithms

Holland (1975) introduced *genetic algorithms*. In these algorithms the search space of a problem is represented as a collection of *individuals*. These individuals are represented by character strings, which are often referred to as *chromosomes*. The purpose of the use of a genetic algorithm is to find the individual from the search space with the best 'genetic material'. The quality of an individual is measured with an evaluation function. The part of the search space to be examined is called the *population*.

Roughly, a genetic algorithm works as follows (see Fig. 3). Firstly, the initial population is chosen, and the quality of this population is determined. Next, at every iteration parents are selected from the population. These parents produce children, which are added to the population. For all newly created individuals of the resulting population a probability near to zero exists that they 'mutate', i.e. that they change their hereditary distinctions. After that, some individuals are removed from the population according to a selection criterion in order to reduce the population to its initial size. One iteration of the algorithm is referred to as a *generation*.

The operators which define the child production process and the mutation process are called the *crossover* operator and the *mutation* operator, respectively. Mutation and crossover play different roles in the genetic algorithm. Mutation is needed to explore new states and helps the algorithm to avoid local optima. Crossover should increase the average quality of the population. By choosing adequate crossover and mutation operators, the probability that the genetic algorithm provides a near-optimal solution in a reasonable number of iterations is enlarged. Under certain circumstances, the genetic algorithms evolve to the optimum with probability 1 (Chakraborty and Dastidar, 1993; Eiben *et al.*, 1990 and Rudolph, 1994).

Further descriptions of genetic algorithms can be found in Davis (1991) and Goldberg (1989).

```
Algorithm GA
  start with an initial time
  t:= 0;
  initialize a (usually random) population of individuals
  initpopulation P(t);
  evaluate fitness of all initial individuals of population
  evaluate P(t);
  test for termination criterion (time, fitness, etc.)
  while not done do
    increase time counter
    t:= t + 1;
    select subpopulation of parents for offspring production
    P':= selectparents P(t);
    recombine the 'genes' of selected parents
    crossover P'(t);
    perturb the mated population stochastically
    mutate P'(t);
    evaluate its new fitness
    evaluate P'(t);
    select the survivors from actual fitness
    P:= survive P,P'(t);
  od
end GA.
```

Fig. 3. *Pseudo-code of a genetic algorithm*

4. Resemblance to the travelling salesman problem

As already stated in Section 1, the problem of obtaining an optimal graph triangulation is equivalent to the problem of finding a node elimination sequence with minimum weight. The problem of the search for an optimal node elimination sequence, however, resembles the intensively studied travelling salesman problem (TSP): given a collection of cities,

determine the shortest tour which visits each city precisely once and then returns to its starting point. In both problems an optimal ordering is searched for.

Several representations and operators have been used in tackling the TSP with genetic algorithms, like the *binary* representation (Holland, 1975; Lidd, 1991; Whitley *et al.*, 1989, 1991), the *adjacency* representation (Grefenstette *et al.*, 1985; Jog *et al.*, 1989; Suh and Van Gucht, 1987), the *ordinal* representation (Grefenstette *et al.*, 1985), the *matricial* representations (Fox and McMahon, 1991; Homaifar and Guan, 1991; Seniw, 1991) and the *path* representation. See Larrañaga *et al.* (1996d) for a review on representations and genetic operators used in relation with the TSP.

For our problem, we choose to use the *path* representation. Therefore, we represent a node elimination sequence by a list of numbers, where the ith element of the list is a j if node j is the ith node to be eliminated. For example, in case $V = \{v_1, v_2, v_3\}$ the string (1 2 3) represents the elimination sequence in which first node v_1 is eliminated, after that node v_2 and finally node v_3.

For the creation of legal offspring when the path representation is used, many genetic operators have been developed (Larrañaga *et al.*, 1996d). Many of these operators were especially developed for the TSP.

Although the TSP and also our problem of finding an optimal node elimination sequence are ordering problems, one important difference between the problems exists: in the TSP, in general, only the relative order is important while in the node elimination problem also the absolute order matters. For example, in the six-cities TSP the string (1 2 3 4 5 6) represents the same tour as the string (4 5 6 1 2 3). In the six-nodes elimination problem both strings represent different elimination sequences. Therefore, we expect that crossover operators that only transmit relative order information from parents to offspring will give bad results for our problem. Note also that the node elimination problem is an asymmetrical problem; the string (1 2 3 4 5 6) does not represent the same elimination sequence as the string (6 5 4 3 2 1). The TSP is often assumed to be symmetrical.

5. Description of the experiments

With a view to uncovering the pros and cons of the application of the genetic algorithms to the graph triangulation problem, we have carried out a large number of experiments. For these experiments we used two artificial graphs which were introduced by Kjærulff (1990): the dense graph *Dense* and the sparse graph *Sparse*. Both these graphs were obtained by simulation and contain 50 nodes, all of which have a number of states chosen at random between 2 and 5. Since the number of edges of the graph *Dense* (359) is much larger than the number of edges of the graph *Sparse* (100), the elimination of a vertex from *Dense* will in general be much more complex and concern quite larger cliques than the elimination of a node from *Sparse*. Therefore, a decomposition of *Dense* will in general have a higher value of the evaluation function than a decomposition of *Sparse*. See the appendix for more details about the graphs.

Table 1. *Results obtained by Kjærulff (1990; 1992)*

Algorithm	*Sparse* w(G)	*Dense* w(G)
Random elimination	32.86	62.84
Max. cardinality search	27.24	59.14
Lexicographic search	26.41	54.39
Ext. random elimination	24.72	53.72
FMINT (random)	25.01	52.88
FMINT (max. card.)	24.62	56.42
Min. size heuristic	23.05	55.01
Min. fill heuristic	24.82	54.40
Min. weight heuristic	24.82	53.48
Simulated annealing	22.61	50.88

In Table 1 Kjærulff's results are presented. For the first nine algorithms the results are related to 100 executions of the algorithm. With respect to the triangulation by simulated annealing, 10 executions of the algorithm were carried out for all combinations of control parameters considered.

For our experiments we use the crossover and mutation operators described in the appendix. All operators but one (the AP operator) have already been used in tackling the TSP with genetic operators.

For evaluating the genetic operators, we consider three different mutation rates p_m ($p_m = 1/10$, $p_m = 1/100$ and $p_m = 1/1000$), three different population sizes λ ($\lambda = 10$, $\lambda = 50$ and $\lambda = 250$) and three different selection biasses b ($b = 0.75$, $b = 1.25$ and $b = 2.00$) for all the 48 (8×6) possible combinations of genetic operators. Here, the selection bias is a number used in the selection of parents for crossover. This number specifies the amount of preference to be given to the superior individuals in the population. For example, a bias of 2.0 indicates that the best individual has twice the chance of being chosen as the median individual. See Whitley (1989) for more explanation on selection bias. For all the 1296 (8 crossover operators $\times$ 6 mutation operators $\times$ 3 mutation rates $\times$ 3 population sizes $\times$ 3 selection biasses) possible combinations of the above parameters we carry out 10 executions of the algorithm.

We use an algorithm based on the principles of the GENITOR algorithm (Whitley, 1989; Whitley and Hanson, 1987), in which the generation reproduction rate, i.e. the proportion of the created individuals in every iteration of the algorithm, corresponds to the inverse of the population size. Moreover, the reduction criterion is *elitist*. Therefore, in every iteration of the genetic algorithm, only one new individual is created, which replaces the worst existing individual in case its adaptation to the problem is larger.

Table 2. *Results obtained with graph* Sparse (I), *respectively: the best, average and worst evaluation found, the average number of iterations of the algorithm before convergence, and the average percentage of converged bits*

	PMX			CX			OX1			OX2		
λ	10	50	250	10	50	250	10	50	250	10	50	250
	25.88	22.68	22.66	22.62	22.61	22.61	26.65	22.98	22.64	24.10	22.64	22.61
	32.36	24.95	23.03	23.41	22.82	22.69	31.80	26.08	23.55	28.55	23.29	22.75
DM	37.89	30.06	24.62	26.04	23.42	22.95	38.71	30.73	26.95	35.54	24.73	23.09
	132	4,043	20,954	7,327	40,056	126,750	185	4,565	39,015	322	6,190	22,252
	100	97	53	99	58	31	100	99	72	100	67	3
	26.43	22.80	22.61	22.63	22.61	22.61	27.38	22.92	22.66	23.52	22.63	22.61
	32.50	25.61	23.14	23.43	22.80	22.71	31.75	26.28	23.59	28.65	23.22	22.75
EM	36.27	29.00	25.18	25.47	23.30	23.28	37.32	31.05	25.86	34.01	25.09	23.28
	102	2,223	19,949	7,104	39,131	128,388	186	4,309	38,480	285	6,116	23,178
	100	100	73	99	63	35	100	100	77	100	70	3
	25.30	22.72	22.61	22.63	22.61	22.61	26.26	22.69	22.63	24.69	22.64	22.61
	31.99	24.99	23.08	23.56	22.82	22.70	31.72	26.05	23.63	28.74	23.36	22.76
ISM	36.30	29.00	24.12	25.98	24.19	22.85	37.32	31.05	25.95	34.01	25.84	23.26
	127	4,377	20,876	6,831	40,580	127,751	182	4,994	38,537	281	6,342	22,669
	100	92	53	99	56	29	100	99	73	100	68	3
	27.19	23.28	22.66	22.61	22.61	22.61	26.57	23.47	22.78	24.17	22.66	22.61
	32.49	25.78	23.17	23.39	22.82	22.71	32.05	26.75	23.97	28.92	23.45	22.75
SIM	37.38	30.11	25.51	26.64	24.08	23.55	37.14	31.05	25.86	34.53	24.94	22.96
	114	3,460	18,539	8,233	40,233	133,450	195	4,919	30,958	260	4,973	22,149
	100	91	60	97	60	32	100	93	69	100	66	4
	27.00	22.72	22.61	22.66	22.61	22.61	26.46	22.66	22.61	22.85	22.61	22.61
	32.14	25.04	23.02	23.54	22.81	22.71	31.85	26.29	23.52	28.43	23.23	22.75
IVM	37.89	30.06	24.75	25.33	24.29	23.28	37.74	30.73	26.52	35.34	25.58	23.28
	125	4,752	20,014	6,656	39,311	124,763	180	4,696	38,934	324	6,222	22,437
	100	93	56	100	61	30	100	99	76	100	68	3
	27.57	23.06	22.67	22.64	22.61	22.61	26.86	24.83	22.72	23.95	22.72	22.61
	32.31	25.58	23.30	23.62	22.81	22.69	31.88	27.04	23.97	29.12	23.54	22.74
SM	36.28	28.84	26.94	28.11	24.08	22.88	36.98	31.65	26.31	36.47	26.84	23.09
	126	3,509	17,881	7,028	40,509	126,973	194	3,956	29,553	249	5,014	22,041
	100	90	58	98	60	30	100	95	72	100	69	3

This means that during the crossover process the parent solutions do not necessarily die, they can survive and be part of the next population if their evaluation are good enough.

The stop criterion is based on the definition of convergence of a population formulated by De Jong (1975). We say that a gene has converged at level α, if this gene has the same value in at least $\alpha\%$ of the individuals in the population. A population converges at level β, if at least $\beta\%$ of the genes has converged. We choose α and β to be equal to 95 and 100, respectively.

Unfortunately, the convergence criterion above does not always guarantee the termination of the algorithm. This is because in general several node elimination sequences exist with practically the same weight. Therefore, it may occur that during the execution of the algorithm the population converges to a set of individuals instead of to one unique individual.

We decide that the population has also converged if in 2000 subsequent iterations the average weight of the population has not decreased.

In all experiments we carry out, we keep the weight of the best individual found, the percentage of converged bits and the number of iterations performed.

In our implementation of the VR operator (see the appendix) we choose p and the threshold equal to 3 and to 2, respectively. We decide that the substrings that are selected by the mutation operators contain at most three nodes.

6. Results

In this section we describe the results and analyse them statistically.

Table 3. *Results obtained with graph* Sparse (*II*), *respectively: the best, average and worst evaluation found, the average number of iterations of the algorithm before convergence, and the average percentage of converged bits*

	POS			ER			VR			AP		
λ	10	50	250	10	50	250	10	50	250	10	50	250
	23.54	22.61	22.61	24.56	22.75	22.63	23.65	22.76	22.73	23.94	22.86	22.66
	28.44	23.36	22.77	28.41	24.26	22.86	26.48	24.25	26.06	32.24	26.55	24.38
DM	34.61	25.35	23.35	31.91	26.74	23.92	30.57	27.71	27.74	38.09	30.45	27.21
	447	6,443	22,032	2,832	41,376	471,217	1,928	16,928	141,746	697	8,609	21,784
	100	62	1	95	75	35	94	97	20	90	88	41
	23.92	22.63	22.61	24.98	22.80	22.61	23.45	22.83	22.79	25.34	22.76	22.65
	28.59	23.35	22.78	28.22	24.41	22.85	26.92	24.47	26.25	32.02	26.55	24.39
EM	34.50	25.68	23.28	31.96	27.14	23.63	30.72	28.36	28.23	37.22	31.84	26.88
	372	6,561	21,655	3,247	40,877	478,239	2,925	16,753	126,261	799	7,945	21,419
	100	65	1	97	74	34	92	96	14	91	78	37
	23.57	22.61	22.61	24.79	22.74	22.64	23.62	22.87	22.72	24.36	22.66	22.66
	28.20	23.34	22.75	28.43	24.34	22.84	26.71	24.17	26.18	32.19	26.56	24.29
ISM	34.50	26.38	23.45	31.61	27.33	23.55	32.80	26.90	27.95	37.78	30.78	27.20
	381	6,322	22,035	3,237	40,668	460,730	1,619	16,213	135,243	719	7,253	23,580
	100	59	1	89	72	31	93	97	17	92	82	40
	23.92	22.64	22.61	25.03	22.81	22.61	24.19	23.02	22.72	28.75	24.81	23.20
	28.96	23.54	22.76	28.49	24.32	22.83	27.23	24.44	26.09	33.30	27.96	25.14
SIM	35.53	25.84	22.97	32.29	27.21	23.62	30.57	27.86	27.79	37.96	32.36	27.65
	361	5,287	21,789	3,092	39,374	455,318	2,141	16,422	135,060	868	3,472	11,486
	100	57	1	94	77	32	93	95	21	84	84	41
	23.49	22.61	22.61	25.12	22.85	22.63	23.40	22.73	22.77	27.08	22.63	22.63
	28.45	23.28	22.77	28.34	24.25	22.81	26.79	24.13	26.15	32.62	26.47	24.45
IVM	34.50	25.26	23.28	33.76	27.00	23.39	30.57	27.57	27.90	38.09	30.91	27.21
	374	6,067	21,826	2,935	43,889	473,370	1,772	18,218	139,172	586	7,852	23,016
	100	60	1	93	75	35	94	96	18	89	85	40
	23.92	22.64	22.61	25.06	23.10	22.61	25.01	22.92	22.97	28.55	24.09	23.23
	29.30	23.49	22.78	28.66	24.42	22.85	27.38	24.47	26.27	33.33	27.94	24.98
SM	34.50	26.26	23.31	32.90	27.76	23.85	31.52	28.08	28.09	38.09	31.07	27.00
	290	4,930	21,246	2,705	38,325	443,222	1,855	16,167	127,463	1,180	2,822	12,696
	100	60	2	97	75	30	94	95	19	71	90	51

Tables 2 and 3 contain for the graph *Sparse*, for all possible combinations of genetic operators and population sizes, respectively: the best, the average and the worst result obtained, the average number of iterations of the algorithm before convergence and the average percentage of converged bits. Note that every value shown is calculated from 90 (3 mutation rates $\times$ 3 selection biases $\times$ 10) executions of the algorithm. In Tables 4 and 5 the same results are given for the graph *Dense*.

6.1. *Objective function*

By comparing Tables 1, 2 and 3 (1, 4 and 5) we see that for the graph *Sparse* (*Dense*) all crossover operators except the AP and the VR (the OX1 and the VR) operator found the best value obtained by simulated annealing. This value was also obtained with all mutation operators. All combinations of crossover and mutation operators, except the combinations AP + SIM and AP + SIM for the graph *Sparse*, were able to give better results than 9 of the 10 triangulation methods of Table 1. Remarkable are the results obtained with the CX operator. For the graph *Sparse*, using the population sizes 50 and 250, this operator always managed to find the best results obtained by simulated annealing, independent of the chosen mutation operator. For the graph *Dense*, the CX operator even managed to find the best result of simulated annealing for all combinations of mutation operators and population sizes, except for the combination DM + $\lambda = 10$.

For the graph *Sparse*, we found for 334 of the 1296 possible parameter combinations an average weight smaller than 23.05, which is the best value obtained by the second best algorithm of Table 1. In 3814 of the 12 960 executions carried out, this value of 23.05 was improved. From these

Table 4. *Results obtained with graph* Dense (*I*), *respectively: the best, average and worst evaluation found, the average number of iterations of the algorithm before convergence, and the average percentage of converged bits*

	PMX			CX			OX1			OX2		
λ	10	50	250	10	50	250	10	50	250	10	50	250
	57.39	50.95	50.88	50.91	50.88	50.88	57.33	52.00	50.92	54.11	50.95	50.88
	62.84	55.52	52.56	52.69	51.88	51.09	63.51	57.03	53.78	59.25	53.02	51.30
DM	67.67	61.29	55.59	56.64	54.44	52.70	68.12	61.15	57.02	63.83	59.18	52.61
	148	5,624	16,765	10,318	24,038	82,688	186	4,316	39,537	406	5,256	13,318
	100	77	35	74	29	14	100	98	72	100	48	3
	59.35	51.27	50.88	50.88	50.88	50.88	59.11	51.81	50.90	53.34	50.88	50.88
	63.07	56.12	52.42	52.54	51.77	51.14	63.12	57.73	54.03	59.32	52.96	51.31
EM	67.24	61.73	55.89	55.72	54.49	53.01	68.12	62.54	58.65	66.28	57.13	53.18
	112	3,509	17,276	10,287	25,499	89,069	180	3,909	39,162	404	4,761	13,218
	100	97	45	73	32	16	100	99	77	100	49	2
	56.15	50.99	50.88	50.88	50.88	50.88	57.33	52.01	50.89	54.15	50.88	50.88
	62.79	55.57	52.51	52.57	51.66	51.09	63.10	57.45	53.60	59.36	52.83	51.22
ISM	68.27	61.76	55.62	56.38	54.54	52.70	68.12	62.80	57.66	63.83	56.04	52.71
	137	5,002	17,396	10,068	25,597	86,897	196	4,794	40,770	340	4,883	13,187
	100	78	34	70	28	16	100	98	70	100	45	2
	59.35	52.06	50.88	50.88	50.88	50.88	57.86	53.41	51.05	52.11	50.99	50.88
	63.26	56.72	52.81	52.59	51.73	51.06	63.41	58.13	54.68	60.00	53.38	51.27
SIM	66.65	61.51	56.63	56.01	55.36	51.91	68.12	62.54	58.11	66.33	56.85	52.70
	133	4,122	15,768	10,751	25,788	82,104	187	4,922	29,181	308	4,336	12,882
	100	87	39	66	30	14	100	92	71	100	47	3
	55.47	50.88	50.88	50.88	50.88	50.88	58.24	51.59	51.01	51.59	50.88	50.88
	62.90	55.60	52.60	52.67	51.93	51.08	63.14	57.08	53.83	59.02	52.96	51.21
IVM	67.67	61.75	55.64	56.66	55.07	52.39	68.12	61.52	58.13	64.00	56.30	53.06
	141	5,394	16,689	10,193	24,771	81,019	184	4,790	40,215	543	4,664	13,180
	100	79	36	69	30	13	100	98	74	100	45	2
	57.04	51.45	50.91	50.88	50.88	50.88	58.38	54.09	52.01	54.79	50.88	50.88
	63.30	57.07	52.79	52.60	51.73	51.07	63.43	57.72	54.65	60.44	53.28	51.24
SM	68.40	61.91	56.92	57.51	54.63	52.70	68.12	62.15	58.19	64.75	56.53	52.70
	139	3,969	15,804	11,011	24,873	84,865	173	4,394	29,012	270	4,257	12,918
	100	87	39	68	31	16	100	93	71	100	50	2

3814 executions, 175 were obtained with $\lambda = 10$, 1081 with $\lambda = 50$ and 2558 with $\lambda = 250$.

For the graph *Dense*, we obtained for 418 possible parameter combinations an average weight lower than 52.88, which is the second best value of Table 1. In 4409 executions this value of 52.88 was improved: in 313 executions with $\lambda = 10$, in 1303 executions with $\lambda = 50$ and 2753 executions with $\lambda = 250$.

We studied the differences between the different parameter combinations with the help of the Kruskal–Wallis test. We found that statistically significant differences exist between the different crossover operators, for both graphs the CX, the POS and the OX2 being the best operators, while the worst results were obtained by the PMX, the OX1 and the AP operator.

Statistically significant differences were found also between the mutation operators. For both graphs, the best results were obtained with the ISM and the DM operators, the worst with the SIM and the SM operators.

As the population size grows, the results improve, the differences becoming statistically significant. For the graph *Sparse* (*Dense*), we found an average weight of 29.12 (59.86) for $\lambda = 10$, of 24.61 (54.70) for $\lambda = 50$ and of 23.58 (52.77) for $\lambda = 250$.

With respect to the mutation rate, we found that the results improved as the mutation rate grew, the differences becoming statistically significant. For the graph *Sparse* (*Dense*), we found an average weight of 26.05 (56.13) for $p_m = 1/1000$, of 25.93 (56.02) for $p_m = 1/100$ and of 25.33 (55.18) for $p_m = 1/10$.

As the selection bias increases, the results get worse, the differences becoming statistically significant. For $b = 0.75$, we found an average weight of 25.31 (for the graph *Sparse*, 55.25 for the graph *Dense*), while for $b = 1.25$ and $b = 2.0$

Table 5. *Results obtained with graph* Dense (*II*), *respectively: the best, average and worst evaluation found, the average number of iterations of the algorithm before convergence, and the average percentage of converged bits*

	POS			ER			VR			AP		
λ	10	50	250	10	50	250	10	50	250	10	50	250
	52.14	50.88	50.88	55.73	51.38	50.88	52.00	51.36	51.50	58.23	51.17	50.88
	59.20	52.83	51.26	59.20	54.08	51.61	58.04	54.26	56.08	63.38	56.97	53.98
DM	65.37	56.23	52.76	62.99	57.31	54.04	64.09	57.98	59.30	68.87	62.10	56.79
	622	5,343	13,242	3,272	38,139	202,581	1,774	18,800	129,408	523	6,172	12,300
	100	43	2	92	38	13	91	84	18	93	78	22
	53.79	50.97	50.88	55.76	51.14	50.88	51.99	50.99	51.07	57.37	51.12	51.00
	59.05	53.16	51.41	59.30	54.02	51.67	57.90	54.57	56.48	63.12	57.01	53.71
EM	65.04	56.96	53.54	63.03	58.12	54.38	62.66	58.44	58.74	68.87	62.73	56.89
	490	4,856	12,542	3,618	36,376	203,178	2,361	16,584	125,778	398	5,490	12,336
	100	43	1	88	41	13	92	88	14	94	76	23
	51.36	50.99	50.88	55.30	51.51	50.88	54.26	51.04	51.19	56.99	50.99	50.95
	59.08	52.91	51.24	59.32	53.91	51.79	58.01	54.40	56.42	63.23	57.31	53.57
ISM	65.04	55.39	52.71	64.09	57.65	54.63	61.92	58.10	59.17	68.87	62.45	56.27
	537	4,924	13,084	3,309	37,191	199,827	1,567	16,791	135,590	501	6,025	14,091
	100	41	1	92	44	11	95	83	13	92	81	27
	55.62	50.99	50.88	55.73	51.22	50.88	53.80	51.32	51.47	58.23	53.87	51.31
	60.22	53.15	51.23	59.52	54.29	51.54	58.15	54.73	56.22	63.95	58.43	54.22
SIM	66.11	57.38	52.93	64.20	59.00	53.49	61.58	58.17	58.64	68.87	62.23	57.07
	321	4,381	12,741	3,527	35,833	195,678	1,990	16,767	132,972	718	3,647	10,599
	100	45	2	89	40	14	89	84	20	85	83	31
	51.87	50.88	50.88	55.15	51.02	50.88	52.81	51.07	51.13	57.36	50.96	50.91
	59.13	53.20	51.22	59.51	54.10	51.87	57.85	54.43	56.10	63.26	57.05	53.89
IVM	65.04	57.95	52.50	63.73	57.35	53.11	64.09	58.58	58.67	68.87	62.10	57.51
	614	4,848	12,548	2,909	35,556	198,027	1,765	17,539	135,004	624	5,400	13,545
	100	43	2	90	42	13	91	82	16	90	79	25
	54.44	50.88	50.88	56.69	51.09	50.88	54.96	51.57	51.04	58.23	54.01	51.63
	59.95	53.10	51.26	59.70	54.12	51.82	58.73	54.54	56.09	63.65	58.53	54.21
SM	66.31	57.71	53.54	64.45	57.54	54.05	63.25	58.12	58.45	68.87	62.28	57.01
	304	4,214	12,495	3,267	36,777	193,324	1,482	16,139	125,667	577	2,914	10,818
	100	44	2	94	42	11	93	81	20	88	83	30

we found an average of 25.70 (55.76) and 26.30 (56.32), respectively.

6.2. *Convergence*

Number of iterations

Statistically significant differences exist with respect to the crossover operators. For the graph *Sparse* (*Dense*), the operators PMX, AP and POS (AP, POS and OX2) lead to the fastest convergence, while ER, VR and CX (CX, VR and ER) give the slowest algorithms.

The differences in the number of iterations with respect to the different mutation operators were not statistically significant.

With respect to the population size, statistically significant differences exist. The average number of iterations until convergence is reached increases as the population size grows.

As the selection bias is lowered, the number of iterations needed to reach convergence increases, the differences becoming statistically significant.

The higher the mutation rate, the larger the number of iterations of the algorithm, the differences becoming statistically significant.

Percentage of converged bits

Statistically significant differences exist with respect to the crossover operator, the population size, the mutation rate and the selection bias.

For the graph *Sparse* (*Dense*) the lowest percentages of converged bits are obtained with the POS, the OX2 and the CX (the CX, the ER and the POS) operator and the highest percentages with the AP, the PMX and the OX1 (the AP, the PMX and the OX1) operator.

The percentage of converged bits decreases, as the population size grows, as the mutation rate increases and as the selection bias is lowered.

With respect to the different mutation operators, no statistically significant differences exist.

6.3. *Crossover operators*

We see that the ER operator gives algorithms which converge extremely slowly. This is not surprising, because the ER operator tries to pass on relative order information from parents to offspring, and moreover, it was developed for the symmetrical TSP.

Since the graph triangulation problem is asymmetrical, we expected that the best results could be found with operators that conserve absolute order information, as the AP, the PMX, the VR and the POS operator. However, it turned out that these operators easily led to algorithms with a premature convergence.

The best results were obtained with the CX operator, which passes on both position and order information from parents to offspring without converging too fast.

For more results and for a more extended analysis, see Larrañaga *et al.* (1994).

7. Conclusions and further research

7.1. *Conclusions*

We tackled the problem of the search for an optimal graph triangulation with genetic algorithms. We tried several combinations of crossover and mutation operators in combination with different population sizes, mutation rates and selection biases.

The results that we obtained improved, for most combinations of parameters, the results of the other known triangulation methods. Only the results obtained with simulated annealing are comparable to our results.

We observed that in general the best results were attached to the relatively slower algorithms. Fast algorithms often went hand in hand with premature convergence.

The crossover operators that passed on mainly absolute position information from parents to offspring were better than the ones that passed on mainly relative order information, although they often led to premature convergence– this means that the algorithm converges before the population contains interesting individuals .

The results obtained with the distinct mutation operators were not very different, although statistically significant differences existed. With respect to the other parameters of the genetic algorithm, we observed that too small a population size, too small a mutation rate and too large a selection bias led to premature convergence, while a very large population size, a very large mutation rate and a very small selection bias gave a very slow algorithm (Larrañaga *et al.*, 1994).

Of course, we would like to answer the question of what is the best triangulation method. Unfortunately, this is not easy to say. It depends on the situation. If enough time is available, a well-designed genetic or simulated annealing algorithm should give the best results. However, for developing such an algorithm a thorough parameter study like the one described in this paper is initially necessary. If there is enough time to carry out such a study probably a stable algorithm could be developed that would lead to acceptable results in an acceptable length of time. However, if a run-time triangulation method is required, a genetic or simulated annealing algorithm would probably be too slow. In general we can say that the time/space complexity of the triangulation method has to be traded off with the expected quality of the resulting triangulation.

7.2. *Further research*

There are several fields in which further research may be done.

We observed that, with respect to the crossover operator to use, we have to weigh up good results with a fast algorithm. In general, choosing for a fast algorithm is at the expense of the goodness of the results.

For example, the results obtained with the CX operator are quite good, but the accompanying algorithms are relatively slow. Therefore, it is interesting to find out if a genetic algorithm with CX crossover can be speeded up without losing its good results. We expect that one way to do this is by carrying out a local search after every certain number of iterations of the algorithm.

Another possibility is to solve the premature convergence related with the use of some specific crossover operators (e.g. the AP and the PMX operator).

We carried out some experiments with the AP operator, adding in every iteration, as well as the new offspring, also an individual created at random. We observed that the average results improved considerably. Obviously, the algorithm also became somewhat slower.

Another interesting possibility is to experiment on other genetic operators or on combinations of several operators. Dynamic operators could also be tried.

It would be interesting to carry out some research to find the differences between the genetic algorithms and simulated annealing for the graph triangulation problem. We would welcome more insight into which of the two methods has the greater chance if coming up with a good decomposition.

Acknowledgements

We thank Uffe Kjærulff for providing us with the data of the graphs *Sparse* and *Dense*. We also thank the referees for their work and comments.

This work was supported by the Diputación de Gipuzkoa, under grant OF 95/1127, and by the grant PI94/78 from the Gobierno Vasco, Departamento de Educación, Universidades e Investigación.

References

Banzhaf, W. (1990) The 'molecular' traveling salesman. *Biological Cybernetics*, **64**, 7–14.

Chakraborty, U. K. and Dastidar, D. G. (1993) Using reliability analysis to estimate the number of generations to convergence in genetic algorithms. *Information Processing Letters*, **46**, 199–209.

Chavez, R. M. and Cooper, G. F. (1990) A randomized approximation algorithm for probabilistic inference on Bayesian belief networks. *Networks*, **20**, 661–85.

Cooper, G. F. (1990) The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, **42**, 393–405.

Dagum, P. and Luby, M. (1993) Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, **60**, 141–53.

Dagum, P. and Horvitz, E. (1993) A Bayesian analysis of simulation algorithms for inference in belief networks. *Networks*, **23**, 499–516.

Davis, L. (1985) Applying adaptive algorithms to epistatic domains. In *Proceedings International Joint Conference on Artificial Intelligence*, Los Angeles, CA, pp. 162–4.

Davis, L. (1991) *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York.

De Jong, K. A. (1975) An analysis of the behaviour of a class of genetic adaptive systems. Ph.D. Dissertation, University of Michigan.

Eiben, A. E., Aarts, E. H. L. and Van Hee, K. M. (1990) Global convergence of genetics algorithms: an infinite Markov chain analysis. Computing Science Notes, Eindhoven University of Technology, The Netherlands.

Fogel, D. B. (1990) A parallel processing approach to a multiple traveling salesman problem using evolutionary programming. In *Proceedings of the Fourth Annual Parallel Processing Symposium*, Fullerton, CA, pp. 318–26.

Fox B. R. and McMahon, M. B. (1991) Genetic operators for sequencing problems. In *Foundations of Genetic Algorithms: First Workshop on the Foundations of Genetic Algorithms and Classifier Systems* (G. Rawlins, ed.), pp. 284–300. Morgan Kaufmann, Los Altos, CA.

Fujisawa, T. and Orino, H. (1974) An efficient algorithm of finding a minimal triangulation of a graph. In *IEEE International Symposium on Circuits and Systems*, San Francisco, California, pp. 172–5.

Fung, R. M. and Chang, K. C. (1990) Weighing and integrating evidence for stochastic simulation in Bayesian networks. In *Uncertainty in Artificial Intelligence 5*, (M. Henrion, R. D. Shachter, L. N. Kanal, and J. F. Lemmer, eds), pp. 209–20, Elsevier, Amsterdam.

Goldberg, D. E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA.

Goldberg, D. E. and Lingle, Jr. R. (1985) Alleles, loci and the traveling salesman problem. In *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, Pittsburgh, PA, pp. 154–9.

Grefenstette, J., Gopal, R., Rosmaita, B. and Van Gucht, D. (1985) Genetic algorithms for the traveling salesman problem. In *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, Pittsburgh, PA, pp. 160–5.

Henrion, M. (1988) Propagating uncertainty in Bayesian networks by probabilistic logic sampling. In *Uncertainty in Artificial Intelligence 2*, (J. F. Lemmer and L. N. Kanal, eds), pp. 149–63, North-Holland, Amsterdam.

Holland, J. (1975) *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.

Homaifar, A. and Guan, S. (1991) A new approach on the traveling salesman problem by genetic algorithm. Technical Report, North Carolina A & T State University.

Hryceij, T. (1990) Gibbs sampling in Bayesian networks. *Artificial Intelligence*, **46**, 351–63.

Jensen, F. (1994) Implementation aspects of various propagation algorithms in Hugin. Technical Report R 94-2014, University of Aalborg, Denmark.

Jensen, F. V. (1996) *An Introduction to Bayesian Networks*. UCL Press, London.

Jensen, C. S., Kong, A. and Kjærulff, U. (1993) Blocking Gibbs sampling in very large probabilistic expert systems. Technical Report R 93-2031, University of Aalborg, Denmark.

Jog, P., Suh, J. Y. and Van Gucht, D. (1989) The effects of population size, heuristic crossover and local improvement on a genetic algorithm for the traveling salesman problem. In *Proceedings on the Third International Conference on Genetic Algorithms*, Arlington, VA, pp. 110–15.

Kjærulff, U. (1990) Triangulation of graphs–Algorithms giving small total state space. Technical Report R 90-09, University of Aalborg, Denmark.

Kjærulff, U. (1992) Optimal decomposition of probabilistic networks by simulated annealing. *Statistics and Computing*, **2**, 7–17.

Larrañaga, P., Graña, M., D'Anjou, A. and Torrealdea, F. J. (1993) Genetics algorithms elitist probabilistic of degree 1, a generalization of simulated annealing. In *Proceedings of the Third Congress of the Italian Association for Artificial Intelligence, IA*AI '93*, Torino, Italy, pp. 208–17.

Larrañaga, P., Kuijpers, C. M. H., Poza, M. and Murga, R. H. (1994) Optimal decomposition of Bayesian networks by genetic algorithms. Internal Report EHU-KZAA-IKT-3-94, University of the Basque Country, Spain.

Larrañaga, P., Kuijpers, C. M. H., Murga, R. H. and Yurramendi, Y. (1996a) Learning Bayesian network structures by searching for the best ordering with genetic algorithms. *IEEE Transactions on Systems, Man and Cybernetics*. Vol. 26, No. 4, pp. 487–93.

Larrañaga, P., Poza, M., Yurramendi, Y., Murga, R. H., and Kuijpers, C. M. H. (1996b) Structure learning of Bayesian network by genetic algorithms: a performance analysis of control parameters. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (to appear).

Larrañaga, P., Murga, R. H., Poza, M. and Kuijpers, C. M. H. (1996c) Structure learning of Bayesian networks by hybrid genetic algorithms. In *Learning from Data: Artificial Intelligence and Statistics V*, (D. Fisher and H. Lenz, eds), Springer-Verlag, New York, pp. 165–74.

Larrañaga, P., Kuijpers, C. M. H. and Murga, R. H. (1996d)

> Evolutionary algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review* (to appear).

Lauritzen, S. L. and Spiegelhalter, D. J. (1988) Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society, Series B*, **50**, 157–224.

Lidd, M. L. (1991) The traveling salesman problem domain application of a fundamentally new approach to utilizing genetic algorithms. Technical Report, MITRE Corporation.

Michalewicz, Z. (1992) *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin.

Mühlenbein, H. (1989) Parallel genetic algorithms, population genetics and combinatorial optimization. In *Proceedings on the Third International Conference on Genetic Algorithms*, Arlington, VA, pp. 416–21.

Neapolitan, R. E. (1990) *Probabilistic Reasoning in Expert Systems, Theory and Algorithms*. Wiley, New York.

Oliver, I. M., Smith, D. J. and Holland, J. R. C. (1987) A study of permutation crossover operators on the TSP. In *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference*, Cambridge, MA, pp. 224–30.

Pearl, J. (1986) Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, **29**, 241–88.

Pearl, J. (1987) Evidential reasoning using stochastic simulation of causal models. *Artificial Intelligence*, **32**, 245–57.

Pearl, J. (1988) *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, Palo Alto, CA.

Rose, D. J., Tarjan, R. E. and Lueker, G. S. (1976) Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on Computing*, **5**, 266–83.

Rudolph, G. (1994) Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, **5**, 96–101.

Seniw, D. (1991) A genetic algorithm for the traveling salesman problem. M.Sc. Thesis, University of North Carolina at Charlotte.

Shachter, R. D. (1988) Probabilistic inference and influence diagrams. *Operations Research*, **36**, 589–604.

Shachter, R. D. and Peot, M. A. (1990) Simulation approaches to general probabilistic inference on belief networks. In *Uncertainty in Artificial Intelligence 5* (M. Henrion, R. D. Shachter, L. N. Kanal and J. F. Lemmer, eds), pp. 221–34. Elsevier, Amsterdam.

Shwe, M. and Cooper, G. (1991) An empirical analysis of likelihood-weighting simulation on a large, multiply connected medical belief network. *Computers and Biomedical Research*, **24**, 453–75.

Suh, J. Y. and Van Gucht, D. (1987) Incorporating heuristic information into genetic search. In *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference*, Cambridge, MA, pp. 100–7.

Syswerda, G. (1991) Schedule optimization using genetic algorithms. In *Handbook of Genetic Algorithms*, (L. Davis, ed.), pp. 332–49. Van Nostrand Reinhold, New York.

Tarjan, R. E. and Yannakakis, M. (1984) Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, **13**, 566–79.

Wen, W. X. (1990) Decomposing belief networks by simulated annealing. In (C. P. Tsang, ed.) *Proceedings of the Australian 1990 Joint Conference on Artificial Intelligence*, pp. 103–18. World Scientific Publishers, Perth, WA.

Wen, W. X. (1991) Optimal decomposition of belief networks. In *Uncertainty in Artificial Intelligence 6* (P. P. Bonissone, M. Henrion, L. N. Kanal, and J. F. Lemmer, eds), pp. 209–24. North-Holland, Amsterdam.

Whitley, D. (1989) The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings on the Third International Conference on Genetic Algorithms*, Arlington, VA, pp. 116–21.

Whitley D. and Hanson, T. (1987) Optimizing neural networks using faster, more accurate genetic search. In *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference*, Cambridge, MA, pp. 391–6.

Whitley, D., Starkweather, T. and Fuquay, D. (1989) Scheduling problems and travelling salesman: The genetic edge recombination operator. In *Proceedings on the Third International Conference on Genetic Algorithms*, Arlington, VA, pp. 133–40.

Whitley, D., Starkweather, T. and Shaner, D. (1991) The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination. In *Handbook of Genetic Algorithms* (L. Davis, ed), pp. 350–72. Van Nostrand Reinhold, New York.

Appendix

A. *Genetic operators*

A.1. *Crossover operators*

The *partially-mapped crossover operator* (PMX) (Goldberg and Lingle, 1985) transmits ordering and value information from the parents strings to the offspring. A portion of one parent string is mapped onto a portion of the other parent string and the remaining information is exchanged. Consider, for example, the two parents (1 2 3 4 5 6 7 8) and (3 7 5 1 6 8 2 4). The PMX operator creates an offspring in the following way. It begins by selecting uniformly at random two cut points along the strings, which represent the parents. Suppose, for example, that the first cut point is selected between the third and the fourth string element, and the second one between the sixth and the seventh string element. Hence, (1 2 3 | 4 5 6 | 7 8) and (3 7 5 | 1 6 8 | 2 4). The substrings between the cut points are called the *mapping sections*. In our example they define the mappings $4 \leftrightarrow 1$, $5 \leftrightarrow 6$ and $6 \leftrightarrow 8$. Now the mapping section of the first parent is copied into the second offspring, and the mapping section of the second parent is copied into the first offspring. Offspring 1: $(x\,x\,x \mid 1\,6\,8 \mid x\,x)$ and offspring 2: $(x\,x\,x \mid 4\,5\,6 \mid x\,x)$. Then offspring i $(i = 1, 2)$ is filled up by copying the elements of the ith parent. If a number is already present in the offspring it is replaced according to the mappings. For example, the first element of offspring 1 would be a 1, like the first element of the first parent. However, there is already a 1 present in offspring 1. Hence, because of the mapping $1 \leftrightarrow 4$ we choose the first element of

offspring 1 to be a 4. The second, third and seventh elements of offspring 1 can be taken from the first parent. However, the last element of offspring 1 would be an 8, which is already present. Because of the mappings $8 \leftrightarrow 6$, and $6 \leftrightarrow 5$, it is chosen to be a 5. Hence, offspring 1: (4 2 3 | 1 6 8 | 7 5). Analogously, we find offspring 2: (3 7 8 | 4 5 6 | 2 1). The absolute positions of some elements of both parents are preserved.

The *cycle crossover operator* (CX) (Oliver *et al.*, 1987) attempts to create an offspring from the parents where every position is occupied by a corresponding element from one of the parents. For example, consider again the parents (1 2 3 4 5 6 7 8) and (2 4 6 8 7 5 3 1). Now we choose the first element of the offspring equal to either the first element of the first parent string or the first element of the second parent string. Hence, the first element of the offspring has to be a 1 or a 2. Suppose we choose it to be 1, (1 * * * * * * *). Now consider the last element of the offspring. Since this element has to be chosen from one of the parents, it can only be an 8 or a 1. However, if a 1 were selected, the offspring would not represent a legal individual. Therefore, an 8 is chosen, (1 * * * * * * 8). Analogously, we find that the fourth and the second element of the offspring also have to be selected from the first parent, which results in (1 2 * 4 * * * 8). The positions of the elements chosen up to now are said to be a cycle. Now consider the third element of the offspring. This element we may choose from any of the parents. Suppose that we select it to be from parent 2. This implies that the fifth, sixth and seventh elements of the offspring also have to be chosen from the second parent, as they form another cycle. Thus, we find the following offspring: (1 2 6 4 7 5 3 8). The absolute positions of on average half the elements of both parents are preserved.

The *order crossover operator* (OX1) (Davis, 1985) constructs an offspring by choosing a substring of one parent and preserving the relative order of the elements of the other parent. For example, consider the two parent strings (1 2 3 4 5 6 7 8) and (2 4 6 8 7 5 3 1), and suppose that we select a first cut point between the second and the third bit and a second one between the fifth and the sixth bit. Hence, (1 2 | 3 4 5 | 6 7 8) and (2 4 | 6 8 7 | 5 3 1). The offspring are created in the following way. Firstly, the string segments between the cut point are copied into the offspring, which gives (* * | 3 4 5 | * * *) and (* * | 6 8 7 | * * *). Next, starting from the second cut point of one parent, the rest of the elements are copied in the order in which they appear in the other parent, also starting from the second cut point and omitting the elements that are already present. When the end of the parent string is reached, we continue from its first position. In our example this gives the following children: (8 7 | 3 4 5 | 1 2 6) and (4 5 | 6 8 7 | 1 2 3).

The *order-based crossover operator* (OX2) (Syswerda, 1991), which was suggested in connection with schedule problems, is a modification of the OX1 operator. The OX2 operator selects at random several positions in a parent string, and the order of the elements in the selected positions of this parent is imposed on the other parent. For example, consider again the parents (1 2 3 4 5 6 7 8) and (2 4 6 8 7 5 3 1), and suppose that in the second parent the second, third, and sixth positions are selected. The elements in these positions are 4, 6 and 5 respectively. In the first parent these elements are present at the fourth, fifth and sixth positions. Now the offspring is equal to parent 1 except in the fourth, fifth and sixth positions: (1 2 3 * * * 7 8). We add the missing elements to the offspring in the same order in which they appear in the second parent. This results in (1 2 3 4 6 5 7 8). Exchanging the role of the first parent and the second parent gives, using the same selected positions, (2 4 3 8 7 5 6 1).

The *position-based crossover operator* (POS) (Syswerda, 1991), a second modification of the OX1 operator, was also suggested in connection with schedule problems. It also starts with selecting a random set of positions in the parent strings. However, this operator imposes the position of the selected elements on the corresponding elements of the other parent. For example, consider the parents (1 2 3 4 5 6 7 8) and (2 4 6 8 7 5 3 1), and suppose that the second, third and the sixth positions are selected. This leads to the following offspring: (1 4 6 2 3 5 7 8) and (4 2 3 8 7 6 5 1).

The *genetic edge recombination crossover operator* (ER) (Whitley *et al.*, 1989, 1991) uses a so-called 'edge map', which gives for each node the edges of the parents that start or finish in it. Consider for example these parent strings: (1 2 3 4 5 6) and (2 4 3 1 5 6). The edge map for these strings is as follows:

Node 1 is connected with the nodes : 2 6 3 5
Node 2 is connected with the nodes : 1 3 4 6
Node 3 is connected with the nodes : 2 4 1
Node 4 is connected with the nodes : 3 5 2
Node 5 is connected with the nodes : 4 6 1
Node 6 is connected with the nodes : 1 5 2

The genetic edge recombination operator works according to the following algorithm:

1. Choose the initial node from one of the two parent strings. (It can be chosen at random or according to criteria outlined in step 4). This is the 'current node'.
2. Remove all occurrences of the 'current node' from the left-hand side of the edge map. (These can be found by referring to the edge list for the current node.)
3. If the current node has entries in its edge list, go to step 4; otherwise, go to step 5.
4. Determine which of the nodes in the edge list of the current node has the fewest entries in its own edge list. The node with the fewest entries becomes the 'current node'. Ties are broken at random. Go to step 2.

5. If there are no remaining 'unused' nodes, then STOP. Otherwise, choose at random an "unused" node and go to step 2.

For our example strings we get:

1. The new child string is initialized with one of the two initial nodes from its parents. Initial nodes 1 and 2 both have four edges; randomly choose node 2.
2. The edge list for node 2 indicates the candidates for the next node are the nodes 1, 3, 4 and 6. The nodes 3, 4 and 6 all have two edges: the initial three minus the connection with node 2. Node 1 now has three edges and therefore it is not considered. Assume that node 3 is randomly chosen.
3. Node 3 now has edges to node 1 and node 4. Node 4 is chosen next, since it has fewer edges.
4. Node 4 only has an edge to node 5, so node 5 is chosen next.
5. Node 5 has edges to the nodes 1 and 6, both of which have only one edge left. Randomly choose node 1.
6. Node 1 must now go to node 6.

The resulting string is (2 3 4 5 1 6), and is composed entirely of edges taken from the two parents.

The *voting recombination crossover operator* (VR) (Mühlenbein, 1989) can be seen as a p-sexual crossover operator, where p is a natural number greater than, or equal to, 2. It starts by defining a threshold, which is a natural number smaller than, or equal to, p. Next, for every $i \in \{1, 2, \ldots, n\}$ the set of ith elements of all the parents is considered. If in this set an element occurs at least the threshold number of times, it is copied into the offspring. For example, if we consider the parents ($p=4$) (1 4 3 5 2 6), (1 2 4 3 5 6), (3 2 1 5 4 6), (1 2 3 4 5 6) and we define the threshold to be equal to 3 we find (1 2 x x x 6). The remaining positions of the offspring are filled with mutations. Hence, our example might result in (1 2 4 5 3 6).

The *alternating-position crossover operator* (AP) (Larrañaga *et al.*, 1994) which simply creates an offspring by selecting alternately the next element of the first parent and the next element of the second parent, omitting the elements already present in the offspring. For example, if parent 1 is (1 2 3 4 5 6 7 8) and parent 2 is (3 7 5 1 6 8 2 4), the AP operator gives the following offspring (1 3 2 7 5 4 6 8). Exchanging the parents results in (3 1 7 2 5 4 6 8).

A.2. *Mutation operators*

The *displacement mutation operator* (DM) (e.g. Michalewicz, 1992) first selects a substring at random. This substring is removed from the string and inserted in a random place. For example, consider the string (1 2 3 4 5 6 7 8), and suppose that the substring (3 4 5) is selected. Hence, after the removal of the substring we have (1 2 6 7 8). Suppose that we randomly select element 7 to be the element after which the substring is inserted. This gives (1 2 6 7 3 4 5 8).

The *exchange mutation operator* (EM) (e.g. Banzhaf, 1990) randomly selects two elements in the string that represents the individual and exchanges them. For example, consider the string (1 2 3 4 5 6 7 8), and suppose that the third and the fifth element are randomly selected. This results in (1 2 5 4 3 6 7 8).

The *insertion mutation operator* (ISM) (e.g. Michalewicz, 1992) randomly chooses an element in the string that represents the individual, removes it from this string, and inserts it in a randomly selected place. For example, consider again the string (1 2 3 4 5 6 7 8), and suppose that the insertion mutation operator selects element 4, removes it, and randomly inserts it after element 7. The resulting offspring is (1 2 3 5 6 7 4 8).

The *simple-inversion mutation operator* (SIM) (e.g. Holland, 1975) selects randomly two cut points in the string that represents the individual, and reverses the substring between these two cut points. For example, consider the string (1 2 3 4 5 6 7 8), and suppose that the first cut point is chosen between element 2 and element 3, and the second cut point between the fifth and the sixth element. This results in (1 2 5 4 3 6 7 8).

The *inversion mutation operator* (IVM) (e.g. Fogel, 1990) randomly selects a substring, removes it from the string and inserts it, in reversed order, in a randomly selected position. Consider again (1 2 3 4 5 6 7 8), and suppose that the substring (3 4 5) is chosen, and that this substring is inserted immediately after element 7. This gives (1 2 6 7 5 4 3 8).

The *scramble mutation operator* (SM) (e.g. Syswerda, 1991) selects a random substring and scrambles the elements in it. For example, consider the string (1 2 3 4 5 6 7 8), and suppose that the substring (4 5 6 7) is chosen. This might result in (1 2 3 5 6 7 4 8).

B. *Sparse* and *Dense* graphs

The syntax of the specification of the *Sparse* and *Dense* graphs is as follows:

$$\langle node \rangle \{ \langle neighbour \rangle \} + \langle newline \rangle$$

The syntax of a cardinality specification is:

$$\{ \langle node \rangle \langle number\text{-}of\text{-}states \rangle \} *$$

Sparse

V1 V14 V13 V2
V2 V44 V1 V3
V3 V41 V24 V2 V4
V4 V19 V32 V8 V3 V5
V5 V9 V4 V6
V6 V40 V27 V23 V5 V7
V7 V49 V21 V16 V44 V6 V8
V8 V24 V20 V4 V7 V9
V9 V23 V25 V5 V8 V10
V10 V9 V11

V11 V32 V10 V12
V12 V30 V11 V13
V13 V32 V41 V37 V1 V12 V14
V14 V39 V1 V13 V15
V15 V14 V16
V16 V32 V37 V7 V15 V17
V17 V45 V25 V16 V18
V18 V42 V21 V27 V17 V19
V19 V23 V4 V18 V20
V20 V40 V8 V19 V21
V21 V24 V7 V18 V20 V22
V22 V44 V46 V21 V23
V23 V26 V19 V9 V6 V22 V24
V24 V21 V47 V8 V3 V23 V25
V25 V47 V17 V9 V24 V26
V26 V23 V25 V27
V27 V41 V39 V34 V6 V18 V26 V28
V28 V43 V27 V29
V29 V28 V30
V30 V12 V29 V31
V31 V30 V32
V32 V44 V13 V16 V11 V4 V31 V33
V33 V44 V32 V34
V34 V27 V33 V35
V35 V46 V34 V36
V36 V35 V37
V37 V16 V13 V36 V38
V38 V48 V37 V39
V39 V41 V27 V14 V38 V40
V40 V6 V20 V39 V41
V41 V39 V27 V13 V3 V40 V42
V42 V18 V48 V41 V43
V43 V28 V42 V44
V44 V33 V32 V22 V7 V2 V43 V45
V45 V17 V44 V46
V46 V35 V22 V45 V47
V47 V25 V24 V46 V48
V48 V38 V42 V47 V49
V49 V7 V48 V50
V50 V49

Dense

V1 V47 V37 V31 V30 V27 V17 V8 V12 V25 V2
V2 V46 V39 V38 V24 V23 V4 V30 V1 V3
V3 V43 V42 V41 V26 V24 V8 V32 V6 V16 V25 V10 V2 V4
V4 V40 V30 V19 V8 V26 V2 V37 V11 V3 V5
V5 V47 V42 V41 V37 V33 V32 V22 V20 V11 V7 V30 V45 V4 V6
V6 V37 V33 V16 V9 V15 V34 V3 V5 V7
V7 V43 V39 V32 V30 V11 V46 V18 V40 V9 V28 V35 V22 V5 V34 V49 V26 V17 V33 V6 V8
V8 V46 V39 V37 V34 V28 V27 V25 V23 V17 V48 V1 V4 V50 V3 V13 V47 V7 V9
V9 V47 V45 V43 V39 V38 V34 V33 V31 V23 V6 V20 V36 V11 V44 V12 V48 V16 V22 V14 V7 V8 V10
V10 V49 V45 V41 V39 V35 V33 V31 V26 V23 V22 V12 V40 V3 V9 V11
V11 V33 V31 V30 V24 V23 V16 V7 V5 V25 V9 V4 V10 V12
V12 V46 V45 V30 V24 V20 V26 V35 V23 V31 V17 V18 V47 V10 V37 V21 V29 V9 V1 V11 V13
V13 V49 V37 V27 V26 V17 V47 V44 V18 V33 V31 V8 V12 V14
V14 V43 V33 V30 V32 V9 V13 V15
V15 V45 V37 V20 V29 V47 V28 V44 V30 V6 V14 V16
V16 V49 V37 V34 V28 V26 V25 V22 V20 V11 V6 V32 V42 V47 V9 V3 V15 V17
V17 V45 V41 V39 V38 V24 V19 V1 V8 V42 V13 V12 V7 V16 V18
V18 V49 V46 V38 V37 V32 V25 V13 V12 V7 V17 V19
V19 V21 V4 V17 V30 V35 V37 V25 V18 V20
V20 V39 V25 V24 V23 V34 V36 V5 V15 V16 V29 V12 V9 V19 V21
V21 V49 V42 V40 V38 V19 V48 V35 V12 V20 V22
V22 V45 V44 V39 V25 V5 V27 V10 V46 V49 V47 V16 V9 V7 V21 V23
V23 V47 V37 V30 V46 V39 V20 V9 V25 V2 V10 V8 V11 V48 V12 V22 V24
V24 V47 V46 V45 V39 V33 V17 V2 V11 V20 V36 V3 V44 V12 V23 V25
V25 V49 V45 V43 V40 V33 V28 V41 V36 V18 V37 V32 V8 V22 V16 V46 V20 V31 V23 V19 V11 V3 V1 V24 V26
V26 V42 V36 V37 V3 V30 V16 V13 V10 V48 V33 V12 V7 V4 V25 V27
V27 V46 V41 V38 V8 V13 V1 V22 V26 V28
V28 V50 V35 V31 V8 V16 V32 V37 V25 V33 V30 V38 V15 V7 V27 V29
V29 V49 V40 V32 V20 V15 V12 V28 V30
V30 V46 V38 V37 V32 V1 V12 V39 V7 V43 V49 V23 V4 V11 V42 V14 V35 V28 V26 V19 V15 V5 V2 V29 V31
V31 V45 V39 V38 V28 V9 V11 V10 V1 V25 V13 V12 V30 V32
V32 V39 V18 V45 V29 V5 V30 V7 V50 V41 V28 V25 V16 V14 V3 V31 V33
V33 V46 V42 V41 V38 V37 V10 V9 V14 V11 V24 V6 V5 V25 V44 V28 V26 V13 V7 V32 V34
V34 V43 V40 V16 V8 V9 V20 V7 V6 V33 V35
V35 V28 V10 V30 V21 V19 V12 V7 V34 V36
V36 V47 V38 V45 V41 V26 V25 V24 V20 V9 V35 V37
V37 V47 V45 V41 V39 V33 V43 V5 V8 V23 V1 V18 V13 V15 V42 V30 V6 V16 V28 V26 V25 V19 V12 V4 V36 V38
V38 V47 V45 V44 V27 V33 V30 V31 V17 V18 V2 V50 V21 V41 V9 V36 V28 V37 V39
V39 V50 V45 V8 V20 V10 V37 V2 V31 V49 V24 V48 V17 V9 V22 V7 V32 V30 V23 V38 V40
V40 V34 V29 V25 V4 V47 V42 V21 V10 V7 V39 V41
V41 V49 V48 V43 V37 V27 V46 V33 V17 V3 V10 V5 V38 V36 V32 V25 V40 V42
V42 V48 V46 V44 V49 V5 V21 V26 V33 V3 V40 V37 V30 V17 V16 V41 V43
V43 V14 V7 V25 V9 V34 V46 V3 V48 V41 V37 V30 V42 V44
V44 V47 V42 V22 V38 V33 V24 V15 V13 V9 V43 V45
V45 V49 V31 V37 V38 V12 V24 V10 V17 V39 V15 V25 V22 V9 V36 V32 V5 V44 V46
V46 V49 V33 V24 V8 V12 V2 V42 V18 V27 V30 V43 V41 V25 V23 V22 V7 V45 V47
V47 V37 V1 V38 V24 V5 V9 V44 V36 V23 V40 V22 V16 V15 V13 V12 V8 V46 V48
V48 V41 V42 V43 V39 V26 V23 V21 V9 V8 V47 V49
V49 V46 V21 V25 V13 V41 V18 V45 V10 V29 V16 V42 V39 V30 V22 V7 V48 V50
V50 V39 V28 V38 V32 V8 V49

Number of states for both graphs

V1 5 V2 2 V3 4 V4 4 V5 4 V6 4 V7 5 V8 4 V9 3 V10 4 V11 3 V12 3 V13 3 V14 3
V15 3 V16 3 V17 2 V18 5 V19 4 V20 4 V21 3 V22 3 V23 3 V24 5 V25 5
V26 4 V27 5 V28 5 V29 5 V30 5 V31 2 V32 3 V33 2 V34 3 V35 4 V36 5
V37 3 V38 5 V39 4 V40 3 V41 3 V42 3 V43 3 V44 3 V45 2 V46 3 V47 3
V48 2 V49 2 V50 3