A Grouping Genetic Algorithm for Graph Colouring and Exam Timetabling

Wilhelm Erben

Department of Computer Science, FH Konstanz – University of Applied Sciences,
78462 Konstanz, Germany
erben@fh-konstanz.de

Abstract. It has frequently been reported that pure genetic algorithms for graph colouring are in general outperformed by more conventional methods. There is every reason to believe that this is mainly due to the choice of an unsuitable encoding of solutions. Therefore, an alternative representation, based on the *grouping* character of the graph colouring problem, was chosen. Furthermore, a fitness function defined on the set of partitions of vertices, guiding the Grouping Genetic Algorithm well in the search, was developed. This algorithm has been applied to a choice of hard-to-colour graph examples, with good results. It has also been extended to the application to real-world timetabling problems. As a by-product, phase transition regions of a class of randomly generated graphs have been located.

1 Introduction

1.1 Graph Colouring

A *k-colouring* of a undirected graph G is a partition of its set of nodes into k subsets ("colours") such that no two nodes connected by an edge belong to the same subset. In a graph colouring problem, the goal is to find a k-colouring for either a given k, or with k as small as possible. The first variant is a pure constraint satisfaction problem involving binary constraints that forbid particular pairs of nodes to have the same colour. The other also includes an optimisation, the search for the *chromatic number* K of graph G.

The graph colouring decision and the optimisation problem are known to be NP-complete and NP-hard, respectively. Therefore, techniques based on heuristics have to be used for large-sized problems. Among them, the *greedy algorithm* is the most simple. Taking the nodes of the graph one by one in given order, the first colour used so far that does not cause a conflict is assigned; if such a colour cannot be found, a new one is requested. The greedy algorithm becomes more powerful if the nodes are first decreasingly ordered by their *vertex degree* (the number of neighbours), before the greedy method is applied. A much better ordering heuristic is used in Brelaz' *Dsatur algorithm* [1]: The next node to be coloured is chosen first among those with the largest number of already (differently) coloured neighbours (the nodes with a maximum *saturation degree*); whenever two of the yet uncoloured nodes have the same saturation degree, the one with higher vertex degree in the uncoloured subgraph is taken. The *Iterated Greedy algorithm* of Culberson et al. [8] uses the greedy algorithm repeatedly; on each iteration, a new permutation of the nodes based on the previously found colouring is produced and presented to the greedy algorithm.

Several other (probabilistic) techniques have been applied to graph colouring problems, such as tabu search (see [14], for example), simulated annealing [15], genetic and other evolutionary algorithms (including hybrids; see [10], [11], [13], for example).

1.2 Timetabling

In a simple timetabling problem, given events have to be scheduled in a number of *periods* in such a way that binary constraints of the form

"Event A must not be assigned to the same period as event B" (*)

are satisfied. In exam timetabling, for instance, no student may be scheduled to take two exams at the same time.

This is easily mapped into a graph colouring problem. The events are represented as the nodes of a graph, and an edge between two nodes corresponds to a binary constraint of the type (*). A valid k-colouring is at the same time a feasible timetable using k periods.

Real-world problems, however, usually involve additional, *non-binary* constraints. Capacity constraints such as seating limitations must be obeyed, for instance. *Near-clashes* should be avoided, if possible: events with an edge constraint should be assigned not only to different, but also non-consecutive periods. Some events must or must not take place at a specific period. Ordering conditions stipulating that an event A must precede a set of other events, or take place at the same time as event B, may occur.... Numerous other examples of constraints could be listed here, where *hard constraints*, fundamental for a solution to be valid, should be distinguished from *soft constraints,* stipulating desirable, though not essential, properties of a timetable.

As in graph colouring, two variants of the timetabling problem can be considered: events have to be scheduled either in a fixed number of periods, or in as few periods as possible, leaving the number of violated soft constraints within allowed limits.

Some graph colouring techniques can be extended to deal also with non-binary constraints. In [18], for instance, a modified Dsatur algorithm that takes seating limitations into account and tries to reduce near-clashes was introduced. Similarly, Carter et al. [4] apply graph colouring heuristics to real-world exam timetabling problems.

1.3 Genetic Algorithms for Graph Colouring/Timetabling

It has frequently been reported that pure genetic algorithms for graph colouring are in general outperformed by more conventional methods (see [20], for instance). This is why various hybrid schemes, incorporating local search procedures into a genetic algorithm, have been proposed in order to improve results (see [13]). Alternatively, adaptive schemes, dynamically changing control parameters of the evolutionary search process depending on the achieved progress, have been developed:

In the evolutionary algorithm presented by Eiben et al. [11], chromosomes are penalised according to the number of uncoloured nodes they contain. In their *Stepwise Adaptation of Weights* (SAW) mechanism, nodes that could not be coloured after a certain number of search steps without violating an edge constraint are considered "hard", and their weights in the penalty function are increased in order to focus the attention of the algorithm on colouring them, as things develop. Results reported in [11] are very good, even a backtracking version of the Dsatur algorithm is mostly outperformed. Another approach yielding similar results is presented by Ross and Hart [19]. In their *adaptive mutation* scheme the mutation probability of a gene is dynamically changed in such a way that wrongly coloured nodes receive higher chances to be mutated and get better colour assignments. A similar approach using *targeted mutation* had been described before by Paechter et al. [16].

For timetabling problems, hybrids seem to yield better results than pure genetic algorithms (see the memetic algorithm of Burke et al. [2] for an example). However, it is, in general, only those genetic algorithms which use a simple direct representation that have been compared with other methods (see [18], for instance, but also [21] for a totally different approach using non-direct representation).

1.4 An Alternative Approach: Grouping Genetic Algorithms

Graph colouring is an instance of the larger family of *grouping problems* (cf. Falkenauer [12]) that consist of partitioning, due to some hard constraints, a set of entities into k mutually disjoint subsets (*groups*). In graph colouring, it is required that connected nodes are placed into distinct groups, whereas the hard constraints for the bin packing problem – the most prominent member of the grouping family – stipulate that items are "packed" into k "bins" without exceeding the capacity limits of these partition sets. Task allocation problems, piling problems, and some vehicle scheduling problems can also be mentioned as members of the grouping family. The grouping decision problems give rise to associated optimisation problems: a cost function, which in most cases is proportional to the number of groups used for the partition, is to be minimised.

In a standard genetic algorithm for a grouping problem, a straightforward direct representation of a solution is used: If N entities (nodes, exams, items, etc.) are to be placed in groups, a chromosome is a string of N genes, and the value of the i-th gene indicates the number of the group that contains the i-th entity. As Falkenauer convincingly argues, this *standard encoding* is not the best choice: It is highly redundant because two distinct chromosomes using different numberings of the groups may encode the same solution. Worse, combined with the classic one- or two-point crossover, good schemata are likely to be disrupted, and "while the schemata are well transmitted with respect to the *chromosomes* under the standard encoding/crossover, their meaning with respect to the *problem* to solve...is lost in the process of recombination" (see [12]).

In an *order-based* (indirect) representation, solutions are encoded as permutations of the entities, and a decoder is applied to retrieve corresponding solutions. In graph colouring, for instance, the greedy algorithm could be used to construct a valid colouring from a permutation of the nodes. Falkenauer gives similar reasons as in the case of direct representation as to why he believes that also this approach is not suitable for grouping problems.

In a *Grouping Genetic Algorithm* (GGA) that he proposes, a chromosome is made up of *groups* as genes. In a graph colouring problem a chromosome could, for instance, consist of the groups {1, 2, 5}, {3, 7}, {4} and {6, 8}, indicating that nodes 1, 2 and 5 have the same colour (no matter which one!) and that another colour has been assigned to nodes 3 and 7, etc. Genetic operators working on these groups have to be defined: In crossover, groups from one parent are injected into the other chromosome while some existing groups are deleted or their contents somehow re-allocated; a mutation could essentially consist of the elimination of a group, or the creation of a new one. In this way it is taken into account that the value of the fitness function actually depends on the *grouping* of the entities, rather than on the single entities themselves, and the genetic operators make sure that whole groups, forming the building blocks used by the genetic algorithm, are processed.

1.5 The Aims of This Paper

A GGA has been developed and tested using some "hard" graph colouring problems and a real-world exam timetabling problem.

A similar algorithm has been implemented before by Eiben et al. [11]. However, results reported by them were quite disappointing: while their algorithm seemed to perform better than a genetic algorithm using standard encoding, it was outperformed by the SAW-ing evolutionary algorithm, which is also presented in [11]. It is assumed that the main reason for this is that the fitness function they chose leads to an inhospitable landscape of the search space, and cannot guide the search for an optimal solution satisfactorily. This is detailed in Sections 2.2 and 3.1.

In Section 2.2, an alternative fitness function is proposed which takes into account, as do many colouring heuristics, that graph vertices of high degree are usually the "hardest" ones. As the test results presented in the subsequent sections show, the GGA equipped with this fitness function, and with the other features given in Section 2, compares well with other powerful techniques.

Graph colouring is known to be NP-hard, even though most graphs are easily coloured [22]. There are, however, hard problem instances, arising at a certain *critical value* of the edge connectivity of a graph [6]. In Section 3, such hard K-colourable (randomly generated) graphs are investigated and taken as a benchmark for the GGA. The test results are compared with those in [11]. Furthermore, a new approximation of the critical value – the point where the *phase transition* occurs – is given for the special class of *equipartite* graphs, showing that this value strongly depends on the size of the graph and its chromatic number. We hope that investigations of such kind will help to develop a good understanding of *real* timetabling problems as well and will make it possible to predict whether a particular problem is difficult to solve or not.

In Section 4, finally, the GGA is applied to a real-world timetabling problem. First test results are presented.

2 The Grouping Genetic Algorithm

The GGA has essentially been implemented following Falkenauer's concepts [12]. It is a steady-state algorithm using tournament selection.

As applications, Falkenauer choses grouping problems such as bin packing and line balancing, but not graph colouring problems. In the following section, therefore, the basic ideas of the GGA are only roughly outlined, whereas its graph colouring specifics are depicted in more detail.

2.1 The Encoding and the Genetic Operators

Given the set $V=\{1,2,....,N\}$ of vertices, a k-colouring of a graph is (most naturally) represented as a set of k mutually disjoint *groups* $G_i \subseteq V$, where

$$\bigcup_{i=1}^{k} G_i = V \ .$$

Due to the hard constraints, two nodes connected by an edge must be assigned to distinct groups (cf. Section 1.4).

Note that the length of the chromosomes, which is the number of groups (the number of colours used), is variable. Also, as long as pure colouring problems are considered, a chromosome may be regarded as a *set* of groups rather than an *array*. For timetabling problems, however, the groups must be ordered according to increasing time.

In the group-oriented encoding, groups are the genes, and hence the genetic operators will work on these groups of equally coloured nodes. The mutation operator selects – according to a given probability (the mutation rate) – groups of the population at random and eliminates them. The nodes of these eliminated groups have to be reallocated in order to receive valid colourings again. To do this, the existing groups of a chromosome are randomly ordered, and the nodes are greedily coloured using the first group that does not cause a conflict; if such a group cannot be found, a new one is created.

For timetabling problems, another mutation operator makes sense: The positions of two randomly chosen groups in a chromosome are exchanged, i.e. the entities originally assigned to period i are scheduled in period j, and vice versa. Other operators, involving more than two periods in the exchange, or inverting the order of groups scheduled to consecutive periods, have been tested, although without showing significant performance improvement.

The crossover operator has been implemented following a general pattern proposed by Falkenauer [12]. Suppose the parent chromosomes are denoted by $P_1 = \{G_1, G_2, ..., G_k\}$ and $P_2 = \{H_1, H_2, ..., H_m\}$, respectively. A non-empty subset $T \subseteq P_1$ of groups is randomly selected from the first parent, and injected into the second one. However, $P_2 \cup T$ is not yet a valid partition of the vertex set, in general, because it will contain groups of the second parent, $H_{i_1}, H_{i_2}, ..., H_{i_r}$, say, which have elements in common with some injected groups. $H_{i_1}, H_{i_2}, ..., H_{i_r}$ are therefore eliminated, and those nodes that now no longer belong to a group are re-inserted in the same way as described above for the mutation operator. The resulting chromosome, containing, in general, groups from both parents as well as some new groups, represents a first offspring. The second one is produced in the same way, with exchanged roles of both parents.

For the initialisation of a population either the greedy algorithm – applied to a random order of the vertices – is used, or else a random colouring producing bad solutions in the beginning, as a challenge for the GGA.

2.2 The Fitness Function

The GGA is intended to be applied to the graph colouring optimisation problem, i.e. to find the minimum number K of colours needed for a colouring. If this chromatic number K is known beforehand, the cost function proposed by Eiben et al. [11] can be used and minimised: In their approach, colourings using more than K colours are simply penalised by counting the number of extra colours, and adding the number of nodes assigned to such unnecessary colours.

$$\mathrm{cost}(P) = k - K + \sum_{j=K+1}^{k} |G_j|, \tag{2.1}$$

where $P=\{G_1,G_2,....,G_k\}$ is a chromosome (a partition) consisting of k groups ($k \geq K$), and $G_{K+1},G_{K+2},....,G_k$ are assumed to be the k – K smallest groups.

In most real-world problems, however, the chromatic number K is not known. Yet more serious, though, is the fact that cost function (2.1) cannot guide the search for an optimal solution satisfactorily: in general, there will be a huge number of sub-optimal (K+1)-colourings, all sharing the same cost value, and it will be difficult to escape from such a plateau. This will be further illustrated in Section 3.1.

For these reasons, an alternative fitness function has been created for the GGA: Given a chromosome $P=\{G_1,G_2,....,G_k\}$, for each group j the *total degree*

$$D_j=\sum_{i \in G_j} d_i ,$$

with d_i denoting the vertex degree of the i-th node, is determined first. The fitness of P is then defined by

$$f(P)=\frac{1}{k}\cdot\sum_{j=1}^{k} D_j^2 \tag{2.2}$$

Obviously, f is to be maximised. Groups containing nodes of high degrees are preferred to groups having the same cardinality but a lower total degree. The idea behind this is nothing else than the well-known heuristic recommending colouring as many nodes of high degree as possible with the same colour. Simultaneously, the number k of used colours will reduce.

Because it is possible to distinguish between two colourings using the same number of colours, the fitness function f provides the algorithm with a search space landscape that is almost free of plains, and thus offers more information to be exploited. The test results presented in the subsequent sections show that, even for hard problems this fitness function is usually able to guide the search to an optimum colouring within a reasonable number of search steps.

3 Results for Graph Colouring Problems

For a number of “hard” graph colouring problem instances, test runs have been performed. In all cases, the chromatic number K is known beforehand, and the task is to find this minimum number of colours and a K-colouring (in short: an optimal solution). The results are compared with those reported in [18], [19] or [11].

3.1 Pyramidal Chain Problems

As a first object of investigation an "artificial counter-example" introduced by Ross, Hart and Corne [18] (see also [19]) has been chosen. It was constructed in order to reveal weaknesses of genetic algorithms with standard encoding (cf. Section 1.4). The inspiration for this class of problems came from studies in [17].

A *pyramidal chain* is a graph arranged as a ring made up of C cliques, each of size K, numbered 1...C where C is an even number (see Figure 1). Each pair of "pyramidal" cliques $2i-1$ and $2i$ ($i \in \{1,....,C/2\}$) overlaps by $K-1$ nodes, and thus takes the shape of a diamond. Two adjacent "diamonds" overlap by exactly 1 node.

It is clear that the graph is K-colourable, and all the "single" nodes which link successive "diamonds" must have the same colour. Brelaz' Dsatur algorithm, or any heuristic based on colouring nodes with maximal degree (which are the single nodes, here) first, will easily find an optimal solution. However, genetic algorithms using the standard representation of solutions fail to solve even small-sized instances of pyramidal chains (e.g., with $K=6$ and $C=20$, hence just $N=60$ nodes and $E=200$ edges), as reported in [18]. This is because "nothing is pushing the GA towards ensuring that all these single nodes have the same colour".

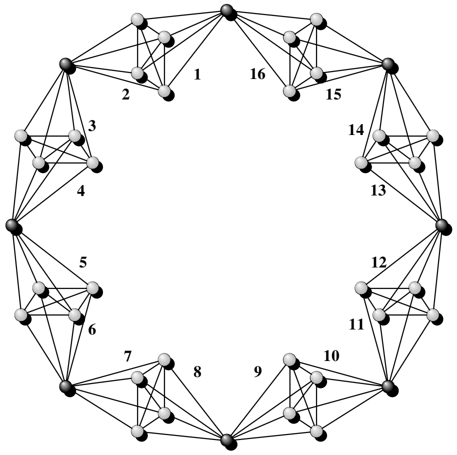

Fig. 1. A "pyramidal chain" with K=5 and C=16

But exactly this "pushing" is done in our grouping oriented approach.

Tables 1–4 present – for the pyramidal chain of Figure 1 – sample calculations of fitness values in different conceivable stages of the evolutionary search process. In an optimal solution using 5 colours (Table 1), all 8 single nodes must belong to the same group j. (In the table, this j is assumed to be 1.) Because each single node has degree 8, and there are no other nodes in group 1, its total degree is $D_1=8 \cdot 8=64$. Each of the other 4 groups contains exactly 1 (non-single) node from each "diamond", and thus the total degree of such a group amounts to 8•5 = 40. According to (2.2), the fitness value of an optimal solution P is now

$$f(P)=\frac{1}{5}\cdot\sum_{j=1}^{5}D_j^2=2{,}099\ .$$

Tables 2–4 show three examples of colourings, all using exactly 1 colour more than necessary. A solution with just 6 of the 8 single nodes assigned to the same group is compared with situations where even 3 or 4 single nodes are wrongly coloured. Although the same number of colours is used in all three solutions, the sums of squared total degrees, and hence the fitness values, differ. The larger the fitness, the closer the optimal solution is approached. Note that the cost function (2.1), in contrast, yields the same value for all three sub-optimal instances!

Tables 1–4. Fitness values for sample colourings of the pyramidal chain of Figure 1

All single nodes have the same colour.

group j	single nodes	other nodes	total degree D_j	D_j^2
1	8	0	64	4,096
2	0	8	40	1,600
3	0	8	40	1,600
4	0	8	40	1,600
5	0	8	40	1,600
6	-	-	-	-
total	8	32	224	10,496
			fitness function (2.2)	2,099
			cost function (2.1)	0

Just 6 single nodes have the same colour.

group j	single nodes	other nodes	total degree D_j	D_j^2
1	6	0	48	2,304
2	2	4	36	1,296
3	0	8	40	1,600
4	0	8	40	1,600
5	0	8	40	1,600
6	0	4	20	400
total	8	32	224	8,800
			fitness function (2.2)	1,467
			cost function (2.1)	5

Just 5 single nodes have the same colour.

group j	single nodes	other nodes	total degree D_j	D_j^2
1	5	0	40	1,600
2	3	4	44	1,936
3	0	8	40	1,600
4	0	8	40	1,600
5	0	8	40	1,600
6	0	4	20	400
total	8	32	224	8,736
			fitness function (2.2)	1,456
			cost function (2.1)	5

Just 4 single nodes have the same colour.

group j	single nodes	other nodes	total degree D_j	D_j^2
1	4	0	32	1,024
2	4	3	47	2,209
3	0	8	40	1,600
4	0	8	40	1,600
5	0	8	40	1,600
6	0	5	25	625
total	8	32	224	8,658
			fitness function (2.2)	1,443
			cost function (2.1)	5

In fact, the GGA equipped with fitness function (2.2) easily solved all test instances of pyramidal chain problems (with between a few hundred and 5,000 evaluation steps, depending on the problem size), whereas cost function (2.1) often failed to find the optimum within the given limit of 10,000 search steps. Note that in order to make the search for an optimal solution more difficult, a simple initialisation procedure yielding random colourings (using, for instance, more than 70 colours for 6-colourable graphs) was used in all test runs, deliberately. For each problem instance several test runs with different seeds for the random generator have been performed.

3.2 "Sequences of Cliques" and Their Phase Transitions

This class of problems was also presented by Ross et al [18]. A quasi-random K-colourable graph is constructed by a graph generator as a sequence of C distinct cliques, each of size K, which are connected by a number of randomly generated edges. Some of the edges between nodes from different cliques, however, are forbidden in order to ensure that the graph-colouring problem is solvable with K colours: Suppose that the vertices of each clique are already assigned the colours 1...K, then no edge should join a node from one clique with the node of the same colour from another clique (cf. figure 2). In other words: The vertices are partitioned into K groups, each group containing exactly one node from each clique.

Hence, only $K(K-1)$ of the K^2 possible edges between two cliques are permissible, and, because there are $C(C-1)/2$ pairs of cliques, at most

$$\frac{C(C-1)}{2} \cdot K(K-1)$$

edges are allowed to be created. Each of these permissible edges is assigned by our graph generator with probability q.

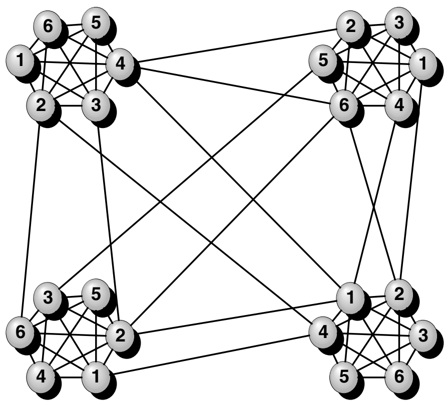

Fig. 2. A "sequence of cliques" with K=6 and C=4

A Preliminary Example. As initial test instances, 6-colourable graphs consisting of a sequence of 20 cliques ($K=6$, $C=20$, $N=120$ nodes) have been chosen. For each edge probability q, ranging from 0.05 to 0.25 in steps of 0.01, 10 problems were randomly generated. For each problem 10 runs were performed. The GGA used a population size of 100, crossover rate 0.2 and mutation rate 0.05. The initial populations were randomly created without using any known heuristic, leaving behind solutions with 70 or more colours at the very beginning. As a limit, at most 150,000 fitness evaluations were allowed before the search process was stopped.

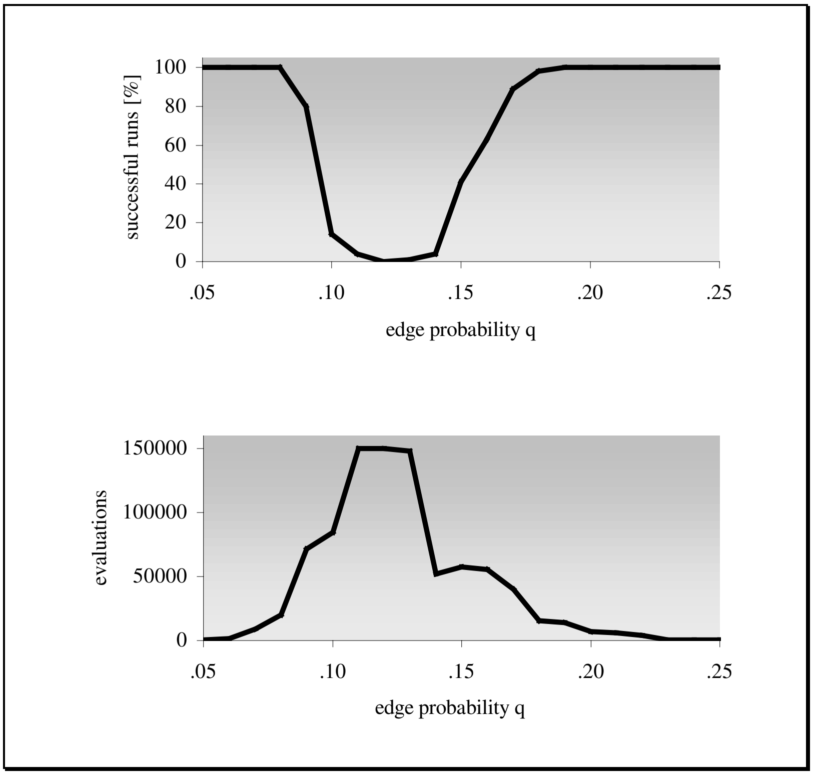

Fig. 3. “Sequences of cliques” with K=6 and C=20 (N=120 nodes):performance at the phase transition

Figure 3 shows the performance of the GGA – in dependence on the edge probability q – with regard to the number of successful runs within the given limit of steps and the average number of fitness evaluations in successful runs.

Obviously, problems with a low edge probability ($q < 0.08$) are easily solved. They belong to an under-constrained region in the space of problems; there are many optimal solutions around, and it is quite easy to find one of them. Problems with a large edge probability q are over-constrained and likely to contain only a few optimal solutions (or just one); such colourings are also found by the GGA with ease because they belong to prominent maxima in the fitness landscape. The hard problems occur at the boundary between the two regions. In our example, this *phase transition* appears to be at $q=0.12$, and the more the edge probability approaches this *critical value,* the more often the algorithm fails to solve the corresponding graph-colouring problems within a reasonable amount of search steps. As observed, adapting the parameter settings of the GGA to the hardness of the problem (for instance, by increasing the mutation rate) would help slightly. The overall behaviour, however, wouldn't change much.

Phase transitions for graph colouring and other NP-complete constraint satisfaction problems have been widely discussed: see for instance Cheeseman et al. [6] or Clearwater and Hogg [7]. With respect to the "sequence of cliques" class, Ross et al. [18] also report on the existence of the "fallible region" in which their genetic algorithm (using standard encoding) as well as other algorithms (among them the Dsatur algorithm) often fail or need a large number of search steps: For the problem instance with $K=4$ and $C=10$ they found that the *critical value* q_{crit} must be close to 0.20.

Their and our observations are supported by the following theoretical considerations. It will be described how the critical values depend on the parameters K and C.

Locating the Phase Transitions. Let us suppose that colours 1...K have been assigned arbitrarily to the K nodes of each clique (cf. figure 2). In order for the colouring to be valid, no nodes of different cliques sharing the same colour should be joined by an edge. For each $i \in \{1,2,....,K\}$ there exist $C(C-1)/2$ pairs of nodes with colour i. The graph generator does not create an edge between two such nodes with probability $1-\tilde{q}$, where $\tilde{q}=(K-1)\cdot q/K$ (recall that for each node only $K-1$ of K possible edges leading to another clique are permissible). Hence, the chosen colouring of the whole graph does not violate any constraints with probability

$$(1-\tfrac{K-1}{K}q)^{K\cdot\frac{C(C-1)}{2}} . \tag{3.1}$$

On the other hand, knowing about the special structure of the graph, there are $K!^{C-1}$ ways to assign colours to its nodes (without obeying the constraints): having assigned K different colours to the nodes of a first clique, for every other clique there exist $K!$ permutations of the colour set.

Now, taking into account that the underlying probability space suffers from maximal uncertainty (or entropy) if and only if all possible outcomes occur with equal probabilities, we conclude that the most difficult problems arise around an edge probability q_{crit} satisfying the equation

$$(1-\tfrac{K-1}{K}q_{crit})^{K\cdot\frac{C(C-1)}{2}} = \frac{1}{K!^{C-1}}$$

which implies

$$q_{crit} = \frac{K}{K-1} \cdot \left(1 - \left(\frac{1}{K!} \right)^{\frac{2}{N}} \right) . \qquad (3.2)$$

According to this formula, the phase transition has to occur at an edge probability of $q_{crit} = 12.5\%$ in the preliminary example above ($K=6$ and $C=20$). Looking at our test results, this seems to be quite a good estimate. Also the findings of Ross et al [18] for $K=4$ and $C=10$ could have been predicted: $q_{crit} = 19.6\% \approx 0.20$.

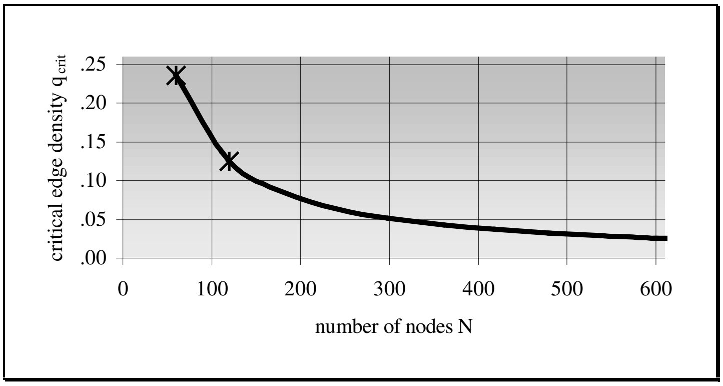

Fig. 4. Critical values $q_{crit} = q_{crit}(N)$ for 6-colourable "sequences of cliques"

Figure 4 shows, for fixed chromatic number $K=6$, how the critical edge probabilities q_{crit} decrease with increasing graph size $N=C \cdot K$. Test runs with the GGA for different values of N (and also K) empirically confirmed the predictions of phase transition regions.

Further Test Results. Figure 5 presents the results for small 6-colourable graphs with a sequence of just 10 cliques. The same parameter settings as before (population size 100, crossover rate 0.2 and mutation rate 0.05) were used. For each edge probability q, ranging from 0.18 to 0.30 in steps of 0.01, 10 problems were generated at random, and for each problem 10 runs were performed.

According to (3.2), the phase transition should occur at $q_{crit} = 23.6\%$. In fact, for some problems the success rate was below 100%, but this occurred only in a region around q_{crit} . The lowest percentage of successful runs was achieved for an edge probability of exactly 24%, and at this point the peak of the average number of evaluations, which are needed to find an optimal solution, is also found.

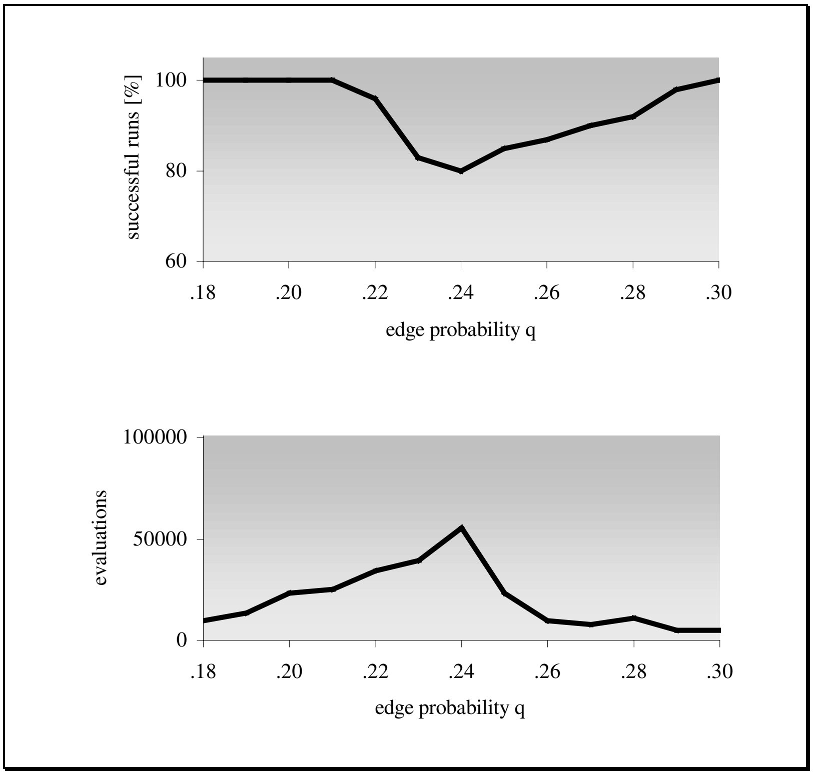

Fig. 5. "Sequences of cliques" with K=6 and C=10 (N=60 nodes): performance at the phase transition

The test results shown in Figure 6 belong to 6-colourable "sequences of cliques" with $C=50$. These graphs have $N=300$ nodes, and for an edge probability q in the phase transition region (which should be around $q_{\text{crit}}=5.1\%$, according to (3.2)), there are approximately $q \cdot C(C-1)/2 \cdot K(K-1) \approx 2{,}000$ edges connecting the cliques.

Again, several edge probabilities q around the phase transition region were chosen: for each $q \in \{0.01, 0.02, \ldots, 0.15\}$, ten problems were generated at random. As a result, a broad ridge between 0.04 and 0.08 in which only very few runs were successful has been found. For q between 0.05 and 0.07, the GGA even failed consistently for all problems within the given limit of 200,000 evaluation steps.

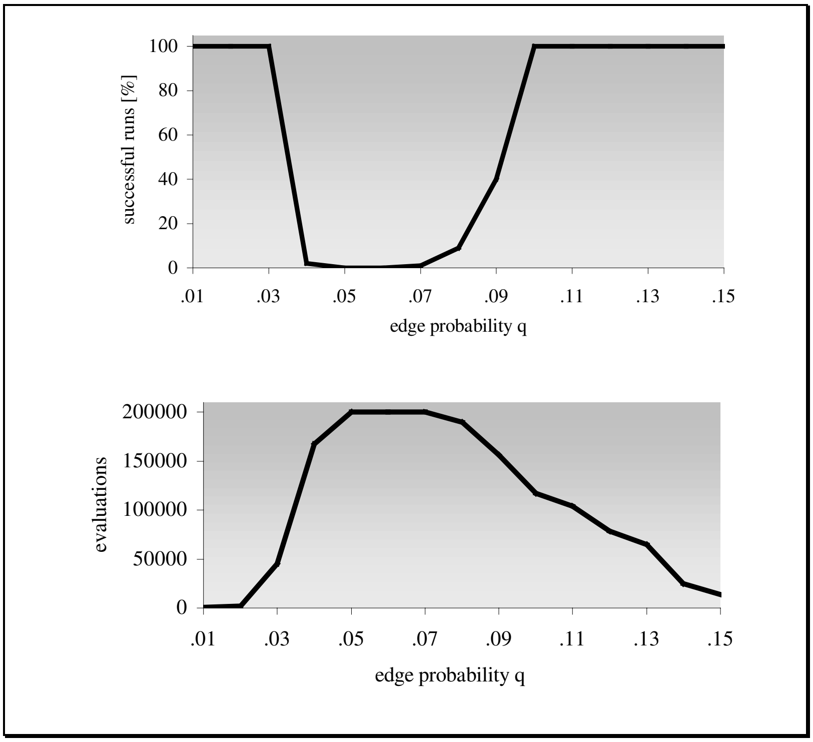

Fig. 6. "Sequences of cliques" with K=6 and C=50 (N=300 nodes) performance at the phase transition

3.3 Equipartite Graphs and Their Phase Transitions

In equipartite graphs the N nodes are partitioned into K almost equal-sized independent sets. Only edges between nodes from distinct partition sets are permitted. Hence, these graphs are K-colourable. (Note that the "sequences of cliques" of the preceding section are special cases of equipartite graphs: There, each partition set contains exactly one node from each of the C cliques.)

In order to produce test instances of equipartite graphs, Culberson's graph generator [9] was used. This generator assigns permissible edges with probability p. Again we conducted studies on phase transition regions depending on this *edge connectivity* p. (Note that for "sequences of cliques" the *edge probability* q refers to the edge density in the set of permissible edges between cliques, whereas Culberson's parameter p also includes the edges belonging to the cliques.)

Results for Equipartite 3-Colourable Graphs. For 3-colourable graphs produced with Culberson's graph generator, our findings were compared with the results obtained by Eiben et al. [11]. In all test runs, the GGA used population size 20, crossover rate 0.2 and mutation rate 0.05. Other settings were also tried, in particular larger population sizes, but hitherto no combinations of parameters yielding significantly better performance could be found. Initial populations were created by the greedy algorithm, which was applied to a random order of the nodes (Section 1.1) and typically assigned 40 or more colours to these initial solutions. The number of allowed fitness evaluations depended on the size of the graphs.

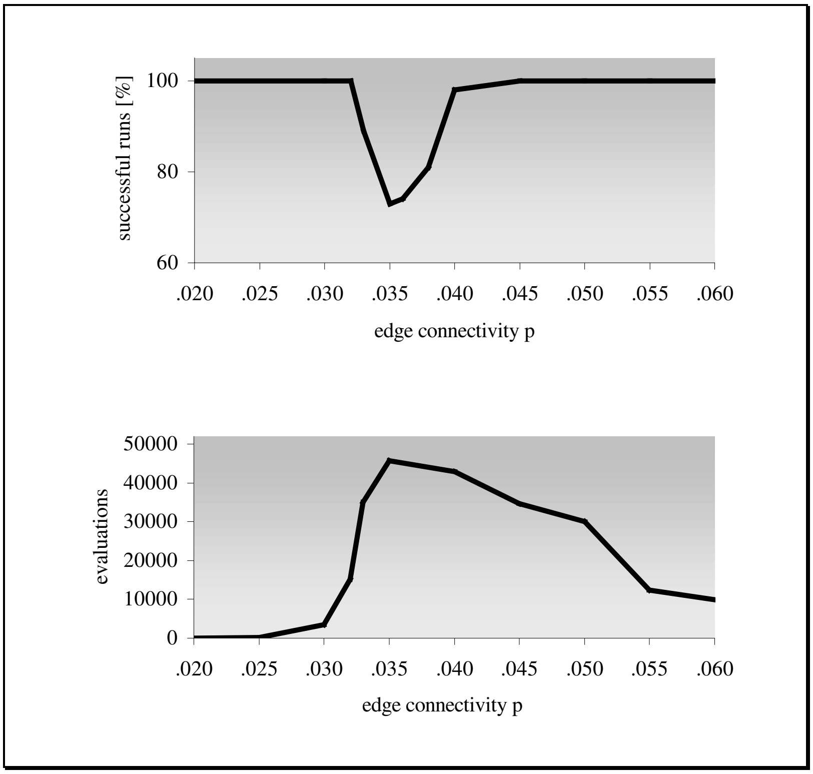

Fig. 7. Equipartite 3-colourable graphs with N=200 nodes

Figure 7 describes the phase transition region for equipartite 3-colourable graphs with $N=200$ nodes. Problem instances were generated for p from 0.02 to 0.06 in steps of at most 0.005. Obviously, the critical value p_{crit} is close to 0.035, and constitutes quite a sharp border to the easy problems to the left. The average number of evaluations needed to find an optimal colouring ranges from 60 at $p=0.02$ to 45,700 at $p_{crit}=0.035$, and about 9,850 evaluations were necessary for $p=0.06$. These results compare well with those reported by Eiben et al. [11]; only in a region to the right of the critical value p_{crit} is our GGA outperformed a little by their SAW-ing algorithm (cf. Section 1.3) with regard to the numbers of evaluations. The GGA they presented, however, shows quite a low performance, and comes off badly in comparison to ours: for $p \geq 0.045$ we always had successful runs, whereas the GGA in [11] leaves the "fallible" region only at 0.09.

The behaviour for larger edge densities p, outside the phase transition region, cannot be presented in Figure 7 due to the scaling of the diagram. To give an indication: for graphs with an edge connectivity of 0.1 and above, under 900 evaluations were needed on average; for $p=0.2$ just less than 100 were required.

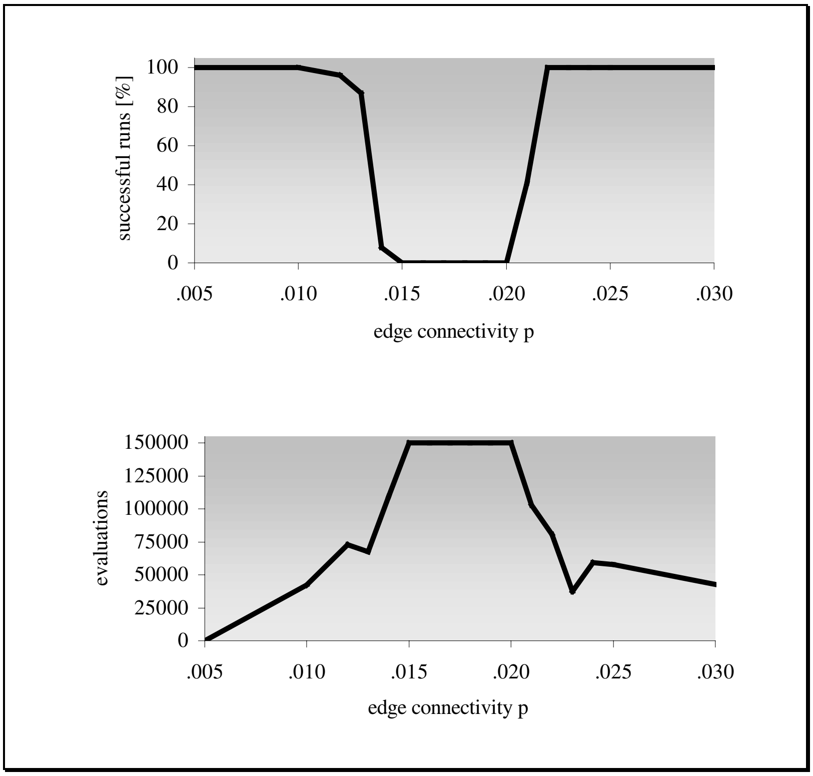

Fig. 8. Equipartite 3-colourable graphs with N=500 nodes

The performance of the GGA for equipartite 3-colourable graphs with $N=500$ nodes is depicted in Figure 8. Here, the edge connectivity p of the problem instances varies from 0.005 to 0.03 in steps of 0.001.

There is a broad ridge from $p=0.015$ to $p=0.02$, in which the GGA was unable to find a 3-colouring within the given limit of 150,000 evaluations. This region of phase transition is similar to what Eiben et al. [11] observed, although their SAW-ing algorithm was again more successful in the immediate right neighbourhood of the critical value p_{crit} (which seems to be at an edge connectivity of above 1.5%). Their GGA, however, is outperformed by far: at $p=0.025$, for instance, it could not find any optimal solution whereas ours successfully terminated all runs within 58,000 evaluations on average. (The peak at $p=0.023$ is certainly due to a sampling error.)

Further results, not shown in the diagrams of Figure 8, are, for instance: for $p=0.05$ still around 3,800 evaluations are necessary on average before an optimal colouring is found, but this number goes down to 500 at $p=0.1$, and under 90 for $p=0.20$.

Locating the Phase Transitions. As indicated before, graphs made up of "sequences of cliques" are special cases of equipartite graphs as defined by Culberson [8] and [9]. The difference is that in graphs produced with his generator, a node needs not be connected by an edge with any node of another partition set, let alone be a member of a clique consisting of K nodes of distinct partition sets. Hence, quite often the independent sets in the generated graphs are not maximal, and sometimes – in particular for very low edge densities – even $(K-1)$-colourable graphs may be produced.

Neglecting this difference for a moment, we assume that all our test instances of equipartite graphs are sequences of cliques, each made up of K nodes of distinct partition sets. This ensures that the chromatic number is K, and the independent partition sets are maximal and really equally sized (as might be the actual intention behind the term "equipartite"). Let us denote this size by C, as in Section 3.2, because $C=N/K$ is also the number of cliques.

Then we can approximate the critical edge connectivity p_{crit} by combining formula (3.2) with the following transformation between the edge probability q of Section 3.2 and Culberson's edge connectivity p:

$$p \cdot C^2 \cdot \frac{K(K-1)}{2} = 1 \cdot C \cdot \frac{K(K-1)}{2} + q \cdot \frac{C(C-1)}{2} \cdot K(K-1) . \qquad (3.3)$$

Here, $p \cdot C^2 \cdot \frac{K(K-1)}{2}$ is the expected number of permissible edges, i.e. edges between pairs of the K partition sets, in a random equipartite graph. $C \cdot \frac{K(K-1)}{2}$ refers to the number of edges in the C cliques, and $q \cdot \frac{C(C-1)}{2} \cdot K(K-1)$ is the expected number of (permissible) edges between cliques.

Hence, $p=\frac{1}{C}+\frac{C-1}{C} \cdot q=\frac{K}{N}+\frac{N-K}{N} \cdot q$, and the sought approximation is

$$p_{crit} \approx \frac{K}{N}+\frac{N-K}{N}\frac{K}{K-1} \cdot \left(1-\left(\frac{1}{K!}\right)^{\frac{2}{N}}\right) . \qquad (3.4)$$

Using formula (3.4), for $K=3$ and $N=200$, we obtain $p_{crit} \approx 0.041$. As expected with the premise we made, this estimated value seems to be a bit too large compared with the empirical results described in Figure 7, although it is still contained in the critical region. However, the approximation is much better for larger graphs because then the number of edges within (fictitious) cliques is low in comparison with the number of other permissible edges. To illustrate, for the graphs with $N=500$ nodes (Figure 8) we calculate $p_{crit} \approx 0.017$ using (3.4), which fits perfectly with the empirical findings.

A brief calculation also shows that for $K=3$

$$\frac{3}{N}+\frac{3}{2}\frac{N-3}{N}\cdot\left(1-\left(\frac{1}{6}\right)^{\frac{2}{N}}\right) \approx \frac{8}{N}.$$

This coincides with the estimation of the critical value for the special case of equipartite 3-colourable graphs in Eiben et al. [11].

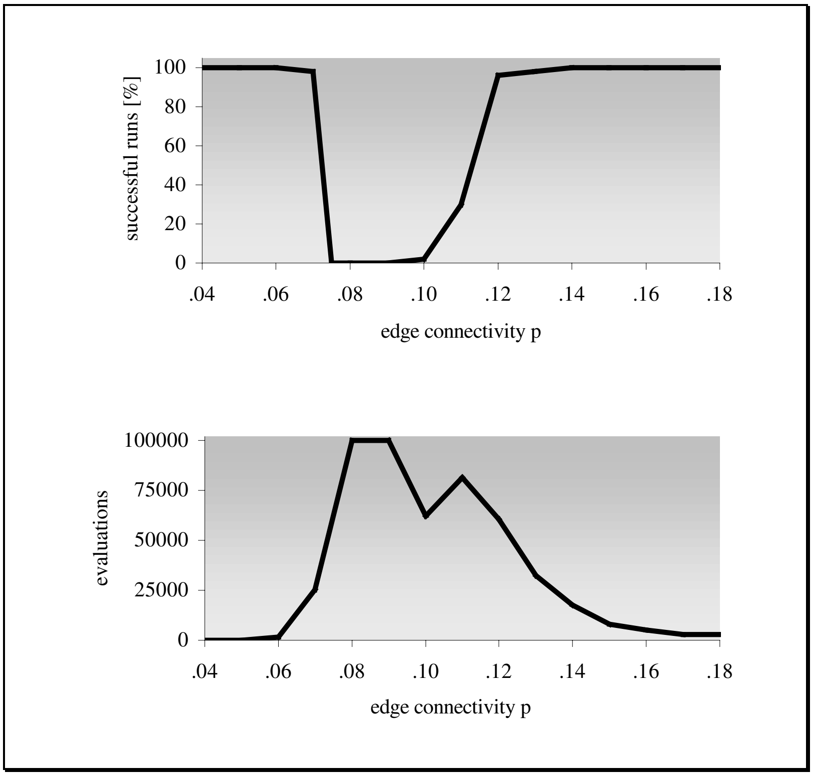

Fig. 9. Equipartite 5-colourable graphs with N=200 nodes

Results for other Equipartite Graphs. As an example for equipartite graphs with a chromatic number greater than 3, and in order to further confirm formula (3.4), the results for 5-colourable problem instances with $N=200$ nodes are presented in Figure 9. The edge probabilities p range from 0.04 to 0.18 in steps of 0.01 or lower. This region includes the phase transition: it was found by using the theoretical considerations above which are apparently again empirically supported. According to (3.4) with $N=200$ and $K=5$, p_{crit} should be about 0.082. In fact, the success rate decreased from 98% at $p=0.07$ to zero at $p=0.075$. Compared with the 3-colourable problems for graphs of the same size (see figure 7), the phase transition region is broader, and it takes longer to find an optimal colouring for $K=5$. For that reason, the maximal number of fitness evaluations was increased to 100,000. In the area at the right end of the phase transition region, the low success rates obviously give rise to sampling errors with regard to the measured evaluation numbers. Starting with $p=0.12$, however, optimal solutions are found in (almost) all runs, and the numbers of evaluations significantly dropped very fast with increasing edge connectivity p.

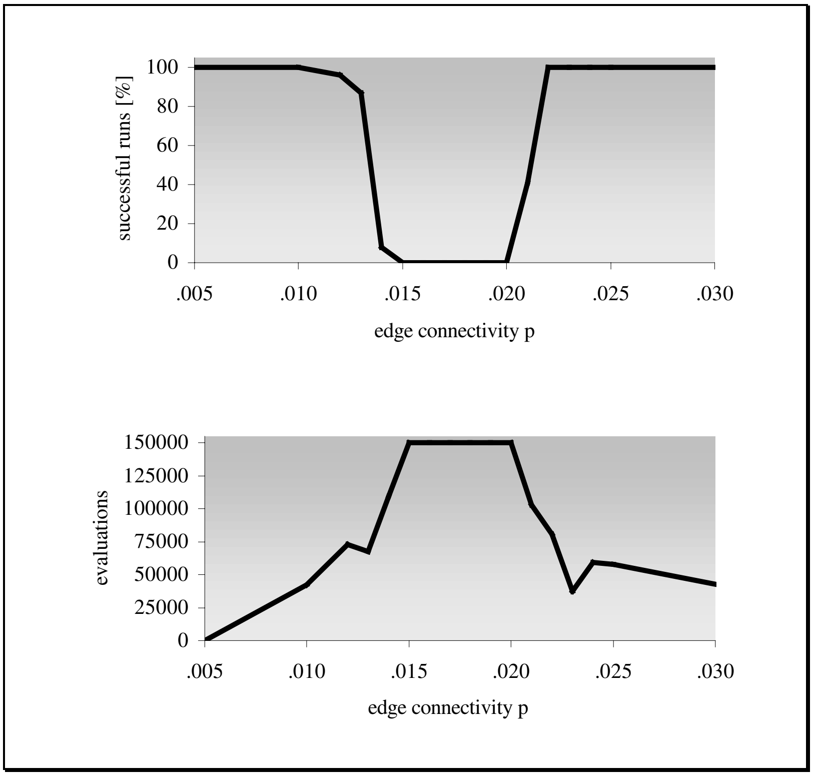

Fig. 10. Equipartite 8-colourable graphs with N=200 nodes

Similar results were obtained for other classes of equipartite graphs; just those for $N=200$ and $K=8$ shall be presented here (see Figure 10): Again, the “theoretical” value $p_{crit} \approx 0.15$ and the findings of the test runs go well together.

4 Adapting the GGA to Exam Timetabling

With some minor changes and extensions, the GGA described in the previous sections can also be applied to timetabling problems. A group represents a set of exams (nodes) scheduled for the same period (equipped with the same colour). A chromosome should reflect the chronological sequence of periods. Therefore, a solution (a timetable) is now represented as a *sequence* (rather than a *set*) of groups, although this is only essential as soon as constraints with regard to consecutive exams are introduced.

As a first test example, a data set from Carter’s collection of real-world exam timetabling problems [5] has been chosen: in the tre-s-92 problem, which contains data from Trent University, $N=261$ exams have to be scheduled taking into account $E=6{,}131$ possible student conflicts (i.e. edges in the graph of the problem).

Binary Constraints. Considering the mere binary-constrained – the graph colouring – problem, the GGA easily finds that $k=20$ periods are enough. This result compares well with those reported in [18] and [4]: A non-evolutionary backtracking algorithm by Carter et al. finds a solution which likewise requires just 20 colours, whereas Brelaz’ Dsatur algorithm only produces a 23-colouring.

Additional Fundamental Constraints. In the tre-s-92 problem, 4,360 students are involved, and there are 14,901 different enrolments. The maximum number of seats, however, is 655. Taking this limitation into consideration, a feasible chromosome in the GGA must not contain any group for which the sum of students taking an exam at this period exceeds 655. In order to implement this “hard” constraint, the greedy algorithm used for initialisation and within the genetic operators was modified in such a way that an exam is (re-)inserted into a group only if the seat capacity is still large enough.

Tests have been performed using different seeds for the random generator and the following parameter setting: population size 20, crossover rate 0.3 and mutation rate 0.06. As a result, a solution using $k=25$ periods is found by the GGA in less than 1,000 evaluation steps. After about 2,500 evaluations a timetable consisting of just 24 periods, and still meeting the seating limitation constraints, is produced. In this solution, most periods are at almost full capacity, and a short calculation shows that this seems to be the absolute optimum of the embedded bin packing problem.

"Soft" Constraints. It is desirable, though not fundamental, for a timetable that constraints like the following are also satisfied (cf. [2], for instance):

"If a student is scheduled to take two exams in any one day
there must be a complete period between the two exams."

It is not likely that such near-clashes could be avoided for all students completely. In the tre-s-92 problem, it is assumed that there are three periods a day. If an exam with a large degree (taken as a node in the problem graph) is scheduled to the middle period of a day it most probably causes near-clashes. Worse, if there are a large number of students taking not only this exam but also the same other exam (i.e., there are many *instances* of the same edge), a large number of violations of the condition may be produced at once. A good timetable, however, should have as few violations of the constraint as possible.

For the GGA this means that the number of near-clashes has to be incorporated into the fitness function, and also the genetic operators should assist the evolutionary process in finding better and better solutions. The fitness function (2.2) was slightly modified:

$$f(P)=\frac{1}{\alpha\cdot|k-k_{input}|+\beta\cdot(n_{nc}+1)}\cdot\sum_{j=1}^{k}D_j^2 \ .$$

Here, k_{input} is the constant number of available periods given as an input, n_{nc} is the number of near-clashes, and α and β are weights. Timetables using more or less than k_{input} "colours" receive lower fitness, and also each violation of the near-clashes constraint is penalised. Experiments showed that α and β should be in the ratio of at least 10 to 1.

A new mutation operator has been introduced, aimed at making the algorithm faster: it exchanges the positions (periods) of two randomly chosen groups in a chromosome. This operator was added to the existing mutation (which takes a group at random and reschedules their exams).

The modified GGA has been tested with the data of the tre-s-92 problem, obeying the seat capacity limitation. Several runs using different seeds for the random generator have been performed, and a limit of 10,000 evaluation steps was set in each run. As a result, the solutions using 35 periods (i.e. almost 12 days) contained 54 violations of the constraint, on average, and when all exams were scheduled into just 34 periods some 110 near-clashes remained.

At first glance, this seems to be unimpressive compared with the results reported in [2] or [21]: Burke et al. added a local search to the operators in their genetic algorithm, and found a solution for the tre-s-92 problem using 35 periods containing just 3 violations of the soft constraint. A completely different approach was followed up by Terashima-Marín et al. They developed a genetic algorithm searching for the best combination of strategies and heuristics to solve the timetabling problem, rather than using a direct representation of a timetable itself. The best solution obtained in this way for the tre-s-92 problem used only 27 slots and contained 586 near-clashes.

But taking into account that the GGA – originally developed for the mere graph colouring problem – has only slightly been changed for the application to the timetabling problem, the results are quite promising. Moreover, the computational expense of an evaluation in the GGA is comparatively low. Further refinements with regard to the genetic operators and the parameter settings will certainly lead to noticeable improvements.

5 Conclusion

Graph colouring and timetabling problems have been considered as special instances of grouping problems, and a GGA based on Falkenauer's [12] concepts has been developed. First test results give grounds for the supposition that this idea deserves to be pursued further.

Comparing our findings with the results reported by Eiben et al. [11], it has been confirmed that not only must an adequate representation of individuals be devised for a genetic algorithm, but – perhaps even more important – the fitness function has to convey as much information about the quality of a solution as possible.

Although counter-examples could be generated, experience shows that partitions of the set of vertices that contain a few groups of high total degree are in general better than solutions where colours are almost equally spread out over the nodes, regardless of the fact that some nodes are "harder" than others. Thus, fitness function (2.2) seems to be a good measure reflecting progress towards improved colourings. This has successfully been tested, applying the GGA not only to some artificial graphs for which other genetic algorithms often fail to find an optimal solution, but also to hard-to-colour instances of randomly generated graphs.

As a by-product, the phase transition regions of this class of random graphs have been located (by a formula which has been empirically confirmed) more precisely than known before. The results could be used to generate further hard colouring or timetabling problems serving as benchmarks for the performance of different algorithms. Further investigations of such kind could also help to develop a good understanding of *real* timetabling problems. The structure of their underlying graphs is often similar to the graph instances used in the test runs described in this paper: Typically there is a number of clusters (each made up, for instance, of the exams within a particular faculty) which are more or less separated, but with high edge connectivity within a cluster. If the location of the phase transition were known it would be possible to decide if the introduction of additional edges between the clusters might make the timetabling problem easier.

As for all genetic algorithms, it turned out to be quite difficult to find an optimal parameter setting for the GGA. Further experiments have to be carried out in this respect. According to our present findings, a population size exceeding 20 individuals does not seem to improve performance significantly. To hard problem instances, in particular, large populations are rather detrimental. With regard to crossover and mutation rate settings, the GGA seems to be quite robust. However, best results have been achieved with a crossover rate between 0.2 and 0.3; for lower rates, performance declines dramatically. Thus, the crossover operator contributes well to the search.

The fitness function prompts further investigation. It has been defined as the average, over all colours, of *squared* total degrees. The square function therein could be replaced by another monotone function in such a way that the weight of colours with high total degree is either strengthened or weakened. In particular for the application to exam timetabling problems, more subtle modifications than hitherto performed should be made in order to improve results. The fitness function should, for instance, prevent some groups – like the middle periods of a day – from containing too many nodes of high degree, because otherwise they are likely to cause a lot of near-clashes.

Further research will also be directed at applying the GGA to more general timetabling problems, incorporating other soft constraints.

References

1. Brelaz, D.: New Methods to Color the Vertices of a Graph. Commun. of the ACM **22** (1979) 251–256
2. Burke, E., Newall, J., Weare, R.: A Memetic Algorithm for University Exam Timetabling. In: Burke, E., Ross, P. (eds.): Practice and Theory of Automated Timetabling, 1st International Conference (Edinburgh, August/September 1995), Selected Papers. Lecture Notes in Computer Science, Vol. 1153. Springer-Verlag, Berlin Heidelberg New York (1996) 241–250
3. Burke, E., Newall, J.: A Multi-stage Evolutionary Algorithm for the Timetable Problem. IEEE Trans. Evol. Comput. **3** (1999) 63–74
4. Carter, M., Laporte, G., Lee, S.Y.: Examination Timetabling: Algorithmic Strategies and Applications. J. Oper. Res. Soc. **47** (1996) 373–383
5. Carter, M.: ftp://ie.utoronto.ca/pub/carter/testprob
6. Cheeseman, P., Kanefsky, B., Taylor, W.: Where the Really Hard Problems Are. In: Mylopoulos, J., Reiter, R. (eds.): Proc. 12th IJCAI-91, Vol. 1. Kaufmann, San Francisco, CA (1991) 331–337
7. Clearwater, S., Hogg, T.: Problem Structure Heuristics and Scaling Behavior for Genetic Algorithms. Artif. Intell. **81** (1996) 327–347
8. Culberson, J., Luo, F.: Exploring the *k*-colorable Landscape with Iterated Greedy. In: Trick, M., Johnson, D. (eds.): Cliques, Colors, and Satisfiability: 2nd DIMACS Implementation Challenge, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Vol. 26. American Mathematical Society (1996) 245–284
9. Culberson, J.: Graph Coloring: http://www.cs.ualberta.ca/~joe/Coloring/index.html (1995)

10. Dorne, R., Hao, J.: A New Genetic Local Search Algorithm for Graph Coloring. In: Eiben, A., Bäck, T., Schoenauer, M., Schwefel, H--P. (eds.): Parallel Problem Solving from Nature – PPSN V, Proc. 5th Int. Conf. (Amsterdam, September 1998). Lecture Notes in Computer Science, Vol. 1498. Springer-Verlag, Berlin Heidelberg New York (1998) 745–754
11. Eiben, A., Van der Hauw, J., Van Hemert, J.: Graph Coloring with Adaptive Evolutionary Algorithms. J. Heuristics **4** (1998) 25–46
12. Falkenauer, E.: Genetic Algorithms and Grouping Problems. Wiley, Chichester (1998)
13. Fleurent, Ch.; Ferland, J.: Genetic and Hybrid Algorithms for Graph Coloring. Ann. Oper. Res. **63** (1995) 437–463
14. Hertz, A., De Werra, D.: Using Tabu Search Techniques for Graph Coloring. Computing **39** (1987) 345–351
15. Johnson, D., Aragon, C., McGeoch, L., Schevon, C.: Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning. Oper. Res. **39** (1991) 378–406
16. Paechter, B., Cumming, A., Norman, M., Luchian, H.: Extensions to a Memetic Timetabling System. In: Burke, E., Ross, P. (eds.): Practice and Theory of Automated Timetabling, 1st International Conference (Edinburgh, August/September 1995), Selected Papers. Lecture Notes in Computer Science, Vol. 1153. Springer-Verlag, Berlin Heidelberg New York (1996) 251–265
17. Ross, P., Corne, D., Terashima-Marín, H.: The Phase-Transition Niche for Evolutionary Algorithms in Timetabling. In: Burke, E., Ross, P. (eds.): Practice and Theory of Automated Timetabling, 1st International Conference (Edinburgh, August/September 1995), Selected Papers. Lecture Notes in Computer Science, Vol. 1153. Springer-Verlag, Berlin Heidelberg New York (1996) 309–324
18. Ross, P., Hart, E., Corne, D.: Some Observations about GA-Based Exam Timetabling. In: Burke, E., Carter, M. (eds.): Practice and Theory of Automated Timetabling II, 2nd International Conference (PATAT'97, Toronto, August 1997), Selected Papers. Lecture Notes in Computer Science, Vol. 1408. Springer-Verlag, Berlin Heidelberg New York (1998) 115–129
19. Ross, P., Hart, E.: An Adaptive Mutation Scheme for a Penalty-Based Graph-Colouring GA. In: Eiben, A., Bäck, T., Schoenauer, M., Schwefel, H--P. (eds.): Parallel Problem Solving from Nature – PPSN V, Proc. 5th International Conference (Amsterdam, September 1998). Lecture Notes in Computer Science, Vol. 1498. Springer-Verlag, Berlin Heidelberg New York (1998) 795–802
20. Terashima-Marín, H.: A Comparison of GA-Based Methods and Graph-Colouring Methods for Solving the Timetabling Problem. Master's Thesis, Department of AI, University of Edinburgh (1994)
21. Terashima-Marín, H., Ross, P., Valenzuela-Rendón, M.: Evolution of Constraint Satisfaction Strategies in Examination Timetabling. Proc. Genetic and Evolutionary Conference 1999, Orlando, FL., July 13–17 (1999) 635–642
22. Turner, J.: Almost All *k*-colorable Graphs are Easy to Color. J. Algorithms **9** (1988) 63–82