Discovering Near Symmetry in Graphs

Maria Fox and **Derek Long** and **Julie Porteous**
University of Strathclyde, Glasgow, UK
Email: {firstname.lastname@cis.strath.ac.uk}

Abstract

Symmetry is a widespread phenomenon that can offer opportunities for powerful exploitation in areas as diverse as molecular chemistry, pure mathematics, circuit design, biology and architecture. Graphs are an abstract way to represent relational structures. The search for symmetry in many contexts can thus be reduced to the attempt to find graph automorphisms. Brendan McKay's NAUTY system (McKay 1990) is an example of one of the highly successful products of research into this task. Erdős and Rényi showed that almost all large graphs are asymmetric, but it is readily observed that many graphs representing structures of real interest contain symmetry. Even more graphs are *nearly* symmetric, in the sense that to each graph there is a closely similar graph that is symmetric. In this paper we explore the problem of finding near symmetries in graphs and describe the techniques we are developing for performing this task.

1 Introduction

Graphs are used to represent relational structures throughout computer science, supporting many different concrete interpretations, including molecules, road networks, circuits, logical formulae, linkages of physical components and management communication and chains of responsibility. Many of their properties have been explored, including symmetry (McKay 1990; de Fraysseix 2000). Symmetry of graph structures is relevant in many applications, such as study of molecular structures (Yao *et al.* 2002) and circuit design, and more computing-related areas such as CSP (Cohen *et al.* 2005) and SAT-solving (Aloul, Sakallah, & Markov 2006).

Erdős and Rényi proved that almost all graphs are asymmetric (Erdős & Rényi 1963). However, it is apparent that many large graphs that arise in practical contexts, including the examples already cited, although not symmetric, exhibit *near* symmetry. The concept of near symmetry (or *almost* symmetry) has been explored in the context of graphs by a few researchers. Chuang and Yen (Chuang & Yen 2002) explore the problem of visualising graphs by exploiting near symmetry. Chen, Lu and Yen (Chen, Lu, & Yen 2000) discuss the discovery of near symmetries in graphs and Buchheim and Jünger apply integer programming techniques to the discovery of near symmetries in graphs (Buchheim & Jünger 2003), referring to them as *fuzzy* symmetries. More recently, Markov has examined almost symmetries of graphs (Markov 2007). Fox and Long also explored the exploitation of near symmetries in planning problems (Fox, Long, & Porteous 2005).

We begin by discussing the definition of the problem of finding near symmetries in graphs, including related work. We then describe the approach we are taking to the discovery of near symmetries and, finally, we present some results.

2 Near Symmetries in Graphs

Intuitively, a graph is nearly symmetric if small changes in the structure lead to the discovery of significant non-trivial automorphisms. There are two difficulties: firstly, it is not obvious what should be considered a "small" change in the graph (or even what constitutes a valid modifying operation) and, secondly, the notion of a significant non-trivial automorphism must be made more precise.

Chen, Lu and Yen consider three graph modifying operations (Chen, Lu, & Yen 2000): node removal, edge removal and edge contraction, in which two adjacent nodes are collapsed into a single node, with all the neighbours of both the original nodes becoming neighbours of the newly formed node. Where there are common neighbours of the two original nodes, the parallel edges formed by the contraction are also collapsed into a single edge. They propose applying one or more of these operations in order to arrive at a graph with a non-trivial automorphism. The induced subgraph must exhibit either axial or rotational symmetry, through an embedding in the plane, and the size of the symmetry is simply the number of nodes in the subgraph that can be so embedded. They cite Manning (Manning 1990), to support the claim that finding maximally symmetric subgraphs using any of these graph reduction operations is NP complete.

Buchheim and Jünger consider another model for graph modification, allowing edge removal and edge insertion. They attempt to find graphs that exhibit either rotational or reflective symmetry. They encode the problem as an integer linear programming problem, attempting to minimise the total cost of the graph modifications, where each edge added or deleted is assigned a cost as part of the problem specification. Reflective and rotational symmetries are handled as separate problems. The behaviour of their algorithm is exponentially expensive in the size of the input graph and they

report results that suggest that graphs of more than 20-24 nodes could not be handled in reasonable time. An advantage of their approach is that it offers a completely explicit specification of the relative costs of various graph modifications (which need not be the same for different operations) and the reward for a particular form of symmetry.

Markov (Markov 2007) has explored almost symmetries in graphs with coloured vertices, by allowing some vertices to be "chameleon" vertices that are allowed to map to vertices of any colour, including allowing chameleon vertices to map to different coloured vertices in different mappings in the collection of permutations that forms an almost symmetry. The ideas have not yet been implemented.

There are several motivations for the search for near symmetry. A symmetric graph can be laid out in an aesthetically pleasing and accessible way and near symmetry can be exploited similarly. In search problems, symmetry can be used to prune the search space by eliminating choices that are symmetric to previously explored, and failed, branches. If the modifications we apply to a graph correspond to removal of constraints (edge removal, say) then near symmetries can allow pruning of solutions to a relaxed problem. Conversely, the addition of constraints (edge addition) might lead to more efficient discovery of solutions to a more constrained, but more symmetric, problem than the original. Fox, Long and Porteous (Fox, Long, & Porteous 2005) have also shown that near symmetry can be exploited to promote choices that are nearly symmetric to others that have proved apparently successful in solving planning problems.

2.1 Formalising Near Symmetry in Graphs

In most graphs it is easy to find small sequences of modifications that lead to non-trivial automorphisms: any pair of nodes with the same set of neighbours yields an automorphism that simply swaps the pair. This means that edge addition leads to a non-trivial automorphism between vertices u and v by adding edges from u to all neighbours of v and vice versa. This requires no more edge additions in graph G than $min_{u,v \in G}\{deg(u) + deg(v) - 2|nbd(u) \cap nbd(v)|\}$, where $deg(u)$ is the out-degree of node u and $nbd(u)$ is the set of nodes adjacent to u. The same transposition symmetry can be achieved by deletion of the edges that connect u to nodes that are not neighbours of v and vice versa. Although these symmetries are easy to find, they are not very interesting since they are merely simple transpositions of the vertices u and v. In general, the symmetries of most interest are those permuting large numbers of the nodes in a graph, since the exploitation of symmetry depends on it affecting large numbers of components

We define the *atomic modifications* applicable to graphs to be: node addition, node deletion, edge addition, edge deletion and edge contraction. More complex modifications can be constructed as sequences of these atomic modifications. However, different graph modifications can be more or less appropriate in different circumstances, according to the interpretation of the graphs. In this work we focus on the discovery of reflective symmetries using only edge contractions. Edge contractions are easy to reverse when near symmetries are used to form good graph layouts, as observed in (Chuang & Yen 2002). For the following definitions, recall that the stabilizer of an element, x, in a set, S, under the action of a group, $\mathcal{G}$, is the set $stab(x) = \{g \in \mathcal{G} | gx = x\}$. We also refer to the symmetries, or automorphisms, of a graph, G, as $Aut(G)$.

Definition 1 Unfixed set of graph *Given a graph, G, let* unfix$(G) = \{v \in G | stab(v) \neq Aut(G)\}$. unfix$(G)$ *is called the* unfixed set *of nodes in G.*

Definition 2 Nearly Symmetric Graphs *Given a graph, G, a collection of identified atomic modification operators, A and two integer bounds k and s, G, is* (k,A,s)-nearly symmetric *if application of at most k operators from A to G yields a graph G' such that* |unfix$(G')| \geq$ |unfix$(G)| + s$.

G' is said to be within k A-steps of *G and s is the* increase in symmetry *offered by G' over G.*

In order to assess the performance of an algorithm for finding near symmetry, it is useful to consider the proportion of originally fixed nodes that is represented by the increase in symmetry. This value is $s/(|G| - |unfix(G)|)$ and we call it the *proportional increase in symmetry*.

Edge or node deletions can be problematic since decomposition of a graph into disconnected components can often increase its symmetry in a relatively uninteresting way. We focus on attempting to maximise the value of the increase in symmetry while minimising the number of edge-contraction steps required to achieve it. There is a tradeoff to be considered between the number of modification steps that are applied to a graph and the increase in symmetry this yields. The value of the tradeoff depends on the relative cost of modifications and reward for increases in symmetry and this will be application-dependent. In the remainder of this paper we will use the description "a nearly symmetric graph", for a graph G, without formally specifying k, A or s, when we mean k to be small (1-4), $A = \{$edge contraction$\}$ and $s \geq |unfix(G')|/2$. We want k to be small because large changes to the graph are likely to have too great an impact on the interpretation of the structure in most applications, while the constraint on s represents our requirement that the near symmetries should be "interesting".

3 Finding Near Symmetries in Graphs

We begin by considering how to find symmetries in graphs. NAUTY (McKay 1990) finds symmetries in graphs by repeated partitioning of the nodes of a graph according to their out-degrees relative to nodes in an existing partition. Initially, nodes are partitioned according to their out-degree in the whole graph. Pairs of partitions are then compared, creating sub-partitions of one partition according to the out-degrees of its nodes restricted to edges connecting to nodes in the other partition. This process continues until it stabilises, at which point any partitions containing more than one node support the existence of a non-trivial automorphism. In a second phase automorphisms are extracted by further partitioning, but, for our purposes, it is sufficient to concentrate on the first phase. The point to be emphasised in this brief review is that nodes in singleton partitions are never mapped to any other nodes in any symmetry of the

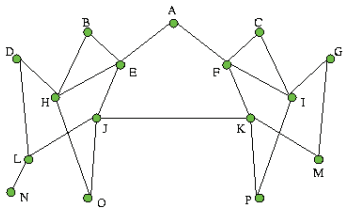

Figure 1: A nearly symmetric graph.

graph. Thus, singletons represent fixed points of symmetries and, therefore, the more of them there are the less interesting will be any symmetry of the graph.

It is instructive to observe how this algorithm behaves on a nearly symmetric graph. The initial partitioning places nodes into the same partitions if they have the same out-degrees. Since the impact of flaws in the symmetry of a graph on out-degrees is entirely local, a graph that has a near reflective symmetry will generate a first partitioning in which most pairs of nodes that transpose under the near symmetry are in the same partition. The impact of the flaws in this symmetry then percolate through the graph along paths between the nodes. Thus, identification of the break in a symmetry between a pair of vertices will take at least as many iterations of the repartitioning as the length of the shortest path from a flaw to one node in the pair. Therefore, the initial partitioning leads to a grouping of nodes that is unaffected by locality, while the impact of flaws in breaking symmetries spreads through partitions according to locality.

Consider the graph in figure 1. The initial partition is:

$$< \{N\}, \{A, B, C, D, G, M, O, P\}, \{L\}, \{E, F, H, I, J, K\} > .$$

Notice that the flaw in the (obvious) potential symmetry lies at the vertices L and N, which already fall into singleton partitions. In many nearly symmetric graphs the flaw will not be so immediately apparent. The first repartitioning, if we consider the third and fourth partitions, say, will yield:

$$< \{N\}, \{A, B, C, D, G, M, O, P\}, \{L\}, \{E, F, H, I, K\}, \{J\} >,$$

since J has one edge to L, while all the other nodes in the fourth partition have none. Notice how the flaw at L and N has already spread to cause J to enter a singleton partition. The flaw spreads its effects so that a comparison of the second and third partitions yields:

$$< \{N\}, \{A, B, C, G, M, O, P\}, \{D\}, \{L\}, \{E, F, H, I, K\}, \{J\} >$$

and then between the second and fifth yields:

$$< \{N\}, \{A, B, C, G, M, P\}, \{O\}, \{D\}, \{L\}, \{E, F, H, I, K\}, \{J\} > .$$

The propagation continues until every node is in a singleton: the graph has no symmetries whatsoever. Note that the graph is nearly symmetric with $k = 1$, $A = \{\text{edge contraction}\}$ and $s = 13$ (the unfixed set of the original graph is empty), by the contraction of L and N.

3.1 Finding and Fixing Flaws

Our approach to finding near symmetry is an interleaving two-stage process. We begin by *finding* the flaws that break the symmetry in the graph and, as each flaw is found, we identify *fixes* for the flaw. A clear separation between these tasks is designed to allow us to extend our approach to consider different ways to fix the flaws configuring the choice making up the operation set for a particular application.

The preceding analysis of the behaviour of NAUTY shows that the identification of flaws in the symmetry of a graph can be performed by applying the NAUTY strategy until singleton partitions are created. A node in a singleton is fixed by any automorphism, representing a limitation of any potential symmetry. Therefore, we want to avoid nodes entering singleton partitions. Since flaws have local effects on the graph, once they have been identified we can limit their damage by focussing on their neighbourhoods to find candidate repairs. The objective is to make the neighbourhood take the same shape as that of at least one other node in the partition from which the singleton would form, or to find a node in another neighbourhood that could be repaired to make it symmetric to the one about to form a singleton. A secondary objective is to limit the impact that the singleton can have on its neighbours, to stop the broken symmetry spreading.

Finding flaws is also a two-stage process. Firstly, we perform the iterative partitioning analysis of NAUTY until a node splits from a partition to form a singleton. Secondly, we must assess whether this singleton is a flaw or is a node on the axis of a potential reflective symmetry. In general, a graph might have several nodes lying on an axis of reflective symmetry and the reflection will be in the stabilizer for each of these nodes. In the graph in Figure 1, node A is on the axis of the reflective symmetry we hope to find, so any attempt to force it into a symmetry relation with another node increases the number of operations applied to the graph without increasing the symmetry. This observation applies universally: to remove an axial node will require additional modifications without increasing symmetry. Therefore, it is important to recognise axial nodes in order to avoid attempting to "repair" the graph to remove them.

3.2 Identifying Axial Nodes

Identifying nodes that lie on the axis of a reflective symmetry that does not yet exist is hard. We use a series of heuristics to determine whether a node is likely to be on such an axis. The techniques cannot be guaranteed to correctly determine the status of every node. Nevertheless, the empirical performance of the heuristics is very strong.

If a node, N, is on the axis then each of its neighbours must either also be on the axis or else must match with another neighbour of N. Thus, by identifying pairs of neighbours that fall into common partitions in the current partitioning, it is possible to determine whether hypothesising N to be an axial node is likely to support an increase in symmetry or lead to a significant number of other nodes being forced onto the axis, too. If N has many neighbours in different partitions then all of these nodes will be forced onto the axis if N is placed on the axis, so this property mitigates strongly against N appearing on the axis. We therefore compare the total number of neighbours of N with the number that form pairs in the same partition and we also consider how many neighbours of N appear alone in other partitions.

A second heuristic we use to decide whether to assign a node to the axis is based on the observation that, in an

odd-sized partition, unless one of the nodes from another partition can be forced to join it, or one of the nodes in it be forced into another partition, by applying some allowed atomic operation, then it is inevitable that one of the nodes must be on the axis. Thus, when a singleton partition forms by splitting from an odd-sized partition, we hypothesise that the singleton node is on the axis and see whether the other nodes in the partition fall into symmetric pairs.

A final heuristic that can be used is the value of *nodal energy*, defined to be the sum of the squared lengths of the shortest paths to all the other nodes in the graph. This value will tend to be smallest for nodes in the centre of the graph and largest for nodes on the fringe of the graph. Unfortunately, it is also a quantity that requires $O(n^3)$ operations to compute for all nodes in a graph of n nodes. This is perfectly acceptable for graphs of even 1000 nodes or so, but if we wish to find near symmetries in graphs generated from SAT formulae, for example, then we must be able to handle graphs of tens or even hundreds of thousands of nodes. Our experiments with this value suggest that it can be informative, but our intention to handle large graphs has led us to abandon the heuristic for reasons of efficiency.

Once a node has been identified as an axial node, if it has an odd-sized neighbourhood then there must be at least one neighbour that is also on the axis. By following this line of reasoning, it is possible to construct a chain of nodes that forms a spine within the axis.

3.3 Identifying Flaws

As we have seen, flaws in the symmetry structure of a graph propagate during the iterating partitioning process, causing nodes to separate into singletons. Unfortunately, in some cases the source of the flaw can lie concealed, not entering a singleton partition, while others around it are affected by the flaw and become separated. This is problematic, since the signal that allows us to identify a flaw and attempt to handle it is the separation of a node into a singleton. If the true source of the flaw does not separate first then we can be misled into perceiving the flaw to be elsewhere in the graph. To prevent a flaw from propagating its effects into the rest of the graph too quickly and to force its discovery as early as possible, we label each partition with the number of times it has been considered during the iterative repartitioning process. This allows us to balance the progress of the partitioning, by choosing to subpartition one of the partitions with the lowest count at each cycle. Where there are ties we break them in favour of odd-sized partitions, since these cannot participate in a reflective symmetry without at least one node splitting into a singleton partition.

To further encourage the early discovery of flaws, at each repartitioning, we consider the partitions generated by comparing the candidate partition against all other partitions. This allows us to identify likely candidates for pairing and potential singletons.

3.4 Fixing Flaws

Once a node appears in a singleton partition and has been designated a non-axial node, we move to the second stage: fixing the flaw. Applying an edge contraction to one of the edges incident on the node will usually impact on the valency of the neighbours of the two nodes that coalesce. This can lead to nodes moving between partitions and the impact of the changes can spread to other partitions that have previously been subpartitioned with respect to a now modified partition. It is therefore useful to return to the original initial partition in assessing the candidate contractions. We apply the following heuristics: if a contraction would lead to fewer odd-sized partitions then it is favoured. If the contraction increases the size of the largest partition then it is favoured. If the contraction decreases the size of the largest partition then it is not favoured, unless that partition is odd-sized.

```
Near_Symmetry(Graph G)
1   let π be initial partition of G
2   label cells in π with count 0
3   while π not stable
4       let p be cell in π with lowest count label
5       let t be best target cell in π for p
6       let π' be partitioning of p against t
7       if π' contains singletons then
6           for each singleton {n} in π'
7               if n is axial then
8                   newaxials = {n}
9                   while newaxials not empty
10                      remove m from newaxials
10                      if |nbd(m)| is odd then
11                                  for each v in nbd(v)
12                                      if v is axial then
13                                          add v to newaxials
14              else fix flaw(n) modifying G
15                  restart at 1
16      set labels of cells in π' to count(p) + 1
17      remove p from π and add π'
18  return G
```

Figure 2: Outline Algorithm: Finding Near Symmetry

Throughout the partitioning process, any nodes that enter partitions of size two are considered to be a hypothetical transposition in the reflective symmetry we are trying to reach. Other pairings of nodes can also arise through the identification of axial nodes. Each node in the graph has a *signature* which is the set of pairs to which its neighbours belong. By considering the signature of the node in the singleton partition, it is possible to identify the most likely candidate for transposition with this node. The signature of that candidate can then be used to filter the nodes in the neighbourhood of the singleton node in order to find the most promising edge to contract. An edge to a neighbouring node of the singleton that shares the largest signature with this candidate is considered the most promising to contract.

Although a search across candidate contractions is possible, we have not found it necessary in the tests we have carried out. It is possible that in graphs with higher density of edges than those we have considered there is more uncertainty about which candidate edge to contract, but in the problems we have solved the choice of candidate is clear using these heuristics alone.

3.5 Outline Algorithm

Our algorithm, shown in figure 2, takes as input a graph G, iteratively identifies flaws and finds resolutions for them and

Nodes	Edges	Flaws	Inc (Prop%)	Time	Buchheim
7	6	1	4 (80%)	0.00	
8	8	1	2 (33%)	0.00	0.02
9	12	1	6 (67%)	0.01	0.05
12	11	1	8 (67%)	0.01	0.84
16	21	2	14 (88%)	0.02	79.0
19	19	1	18 (95%)	0.02	
20	28	2	18 (90%)	0.04	
41	44	1	30 (91%)	0.18	
50	50	2	32 (89%)	1.00	
100	103	3	78 (93%)	9.82	
100	113	2	60 (88%)	13.21	
100	112	5	55 (77%)	14.94	

Figure 3: Representative results. Figures under Buchheim, taken from (Buchheim & Jünger 2003), are averages for finding optimal near symmetries on graphs of the same size, using a machine with similar specification.

then returns the modified graph. The algorithm begins by finding the initial partition of G, π (line 1). For clarity, we refer to the parts of this partition as cells. The algorithm proceeds to iterate on π while it continues to generate new subpartitions. If a flaw is identified during this process and fixed, the algorithm restarts (line 15).

Throughout the algorithm, whenever partitions are generated, if any size 2 cells are created then this pair of vertices is treated as a *hypothesised transposition*. Such pairs are used in several places in the algorithm, as indicated below. Following a restart we clear this collection and start afresh. Partitions generated during the search for a target cell (line 5) can generate pairs that are stored even though the partition is not with the cell finally selected as the target.

In each iteration, a cell is selected with lowest count, in order to maintain a balanced descent in the partitioning process. Ties are broken in favour of odd-sized partitions and then in favour of largest partitions (line 4). We then identify a cell to use as a partner in the subpartitioning process (line 5). The target is selected to generate singletons if possible (favouring most singletons in ties). Otherwise, we select to generate most odd-sized subpartitions, then most subpartitions. If the subpartitioning between p and the target, t, generates singletons then we proceed to evaluate them, one by one (line 6). We begin by deciding whether the singleton is (best considered) axial.

To test whether a node n is axial, we begin by considering the neighbourhood of n. If it contains any hypothesised transpositions then we consider n to be axial. If the neighbourhood nodes appear in many different partition cells then n is non-axial. If the neighbourhood is odd-sized then we prefer n to be non-axial. We also consider p: if it is odd-sized then we check for pairs amongst the hypothesised transpositions in p. If some of the nodes in the cell are paired then n is axial, otherwise we accept the determination based on neighbourhood. If the cell is even we favour n to be non-axial. Where a node is determined to be axial and it has an odd neighbourhood, we attempt to follow the spine of nodes that are on the axis (lines 7-13).

Singletons that are not identified as axial are then considered flaws and an attempt made to fix them (line 14), according to the strategy described in section 3.4.

Finally, as the cells in the new subpartition replace p in π, they are labelled with an appropriate count, one greater than the label on p (line 16).

4 Results

The algorithm is implemented and we report some results in figure 3. We generated data by creating random graphs with symmetry and introducing random flaws into the symmetry. We also used examples of graphs from other sources, including a molecular structure, examples taken from (Chuang & Yen 2002) and some graphs constructed by hand. We report the number of **Nodes**, **Edges**, **Flaws** (the number of edges contracted to restore symmetry to the graph) and the **Inc**rease in symmetry this series of contractions yields. We also report the proportional increase in symmetry (Prop%). As can be seen, particularly for larger graphs, the proportion of nodes that are newly brought into symmetrical relation with other nodes in the graph is consistently more than 75%. The data we report includes a graph that is almost rotationally symmetric, with a symmetry of order 3 (size 19). The repair performed on this graph strives to achieve the reflective symmetry, but actually achieves it by restoring a three-fold symmetry that also offers reflective symmetry along three axes. In general, we would not expect rotational symmetry to be identified by our algorithm, except as a side-effect of restoring reflective symmetry. We report the **time** taken by our algorithm on these problems and, for comparison, the times (for similar sized graphs) given by Buchheim and Jünger (Buchheim & Jünger 2003). They report results for graphs of between 8 and 16 nodes: the hugely combinatorial growth from 12 to 16 node graphs indicates the infeasibility of applying their approach to larger graphs.

As an example illustrating the behaviour of our algorithm, figure 4 is a sample input graph laid out using *neato*, which is a spring-based graph layout algorithm. As can be seen, the graph layout hints at some possible symmetry, but application of NAUTY yields a disappointing collection of symmetries: just three symmetries consisting of pairs of transpositions and two symmetries that are simple transpositions (the symmetries are given in cycle notation in figures 4 and 5). These are all very localised symmetries that occur between nodes in fringe parts of the graph. Application of our algorithm identifies three edge contractions to apply to the graph and the resulting graph is shown in figure 5. Application of NAUTY identifies several local symmetries (mostly those there in the original graph, although the contractions lead to two new ones) and one huge symmetry that transposes 92 nodes in 47 pairs and leaves five nodes on the axis of symmetry. The three edge contractions dramatically increase the symmetry that is available in the graph. This result is typical of the behaviour of the algorithm. It is also worth emphasising that this example is typical of the behaviour of NAUTY on nearly symmetric graphs. NAUTY is intended to find symmetry: where there is no actual symmetry it will find nothing, regardless of how close to a graph is to being symmetric.

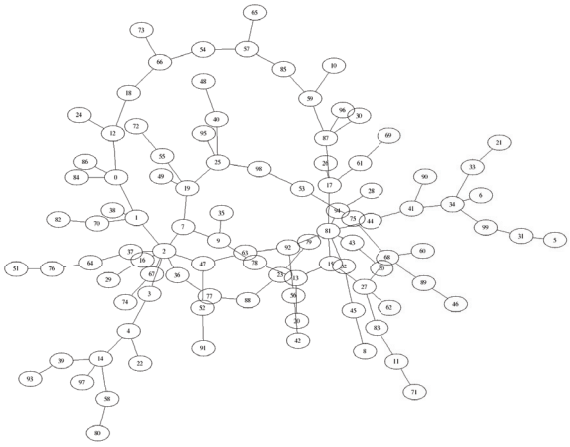

Figure 4: A nearly symmetric graph with 100 nodes. Symmetries in this graph are:
(28 50)(43 94)
(16 67)(29 74)
(30 96)
(84 86)
(39 58)(80 93)

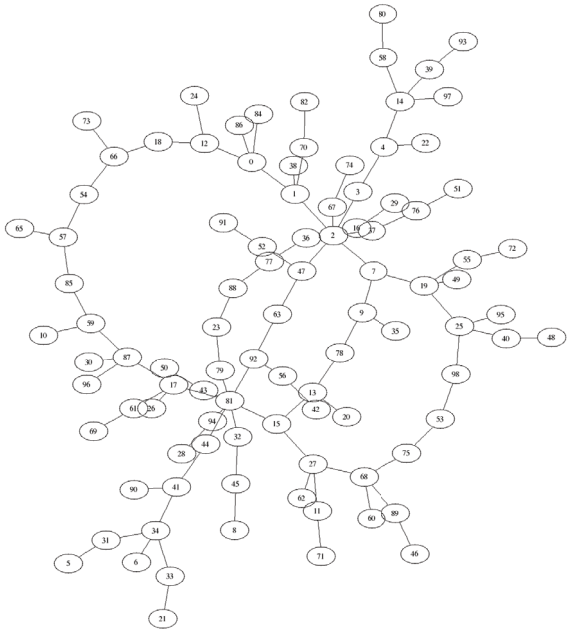

Figure 5: Graph from Figure 4, following three edge contractions identified by the algorithm, demonstrating a (3,{edge-contraction},78)-near symmetry. 92 nodes are involved in the new large reflective symmetry, while only 16 nodes appear in small local symmetries in the original graph. In addition to original symmetries we now have:
(5 21)(31 33)
(64 83)
(0 87)(1 17)(2 81)(3 44)(4 41)(5 80)(6 97)(7 15)(8 51)(9 13)(10 24)(11 55)(12 59)(14 34)(16 94)(18 85)(19 27)(20 35)(21 93)(22 90)(23 77)(25 68)(26 38)(28 29)(30 84)(31 58)(32 37)(33 39)(36 79)(40 89)(42 91)(43 67)(45 76)(46 48)(47 92)(49 62)(50 74)(52 56)(57 66)(60 95)(61 70)(65 73)(69 82)(71 72)(75 98)(86 96)

5 Conclusion

In this paper we have motivated and outlined our approach to the discovery of near symmetries in graphs, using edge contraction to find reflectively symmetric graphs. We have taken several steps in building heuristics that are very effective in identifying near symmetry in graphs that include graphs from natural sources. Its performance is very promising. The algorithm we have developed successfully identifies near symmetries that can be exploited in constructing layouts of the nearly symmetric graphs. The resulting layouts highlight the near symmetries and, as we have illustrated, help to make the structure of the graphs more accessible to human viewers. This is one of several relevant application areas of the approach. The application of our approach in other contexts depends on identifying which graph modification operators are appropriate given the interpretation of the graph. We recognise that edge contraction is not always meaningful and other operations must be available if the work is to find general application. However, our algorithm already provides a framework in which the exploitation of different operations can be easily explored.

Symmetry reduction has proved a powerful technique in a variety of search-based techniques, including SAT-solving (Aloul *et al.* 2003) and CSP (Roy & Pachet 1998; Backofen & Will 1999) and general search (Crawford *et al.* 1996). It is clear that the power of these techniques is severely curtailed by the impact of small flaws in the potential symmetry of the structures over which they must reason. The identification of these flaws and appropriate repairs to them that restore large scale symmetry to a problem could offer access to the search-reduction techniques and greatly improve performance. Our future work will include a careful exploration of these possibilities.

References

Aloul, F.; Ramani, A.; Markov, I.; and Sakallah, K. 2003. Solving difficult SAT instances in the presence of symmetry. *Transactions on Computer-aided Design* September issue.

Aloul, F.; Sakallah, K.; and Markov, I. 2006. Efficient symmetry breaking for boolean satisfiability. *IEEE Transactions on Computers* 55(5):549–558.

Backofen, R., and Will, S. 1999. Excluding symmetries in constraint-based search. In *Proc. Constraint Programming*.

Buchheim, C., and Jünger, M. 2003. An integer programming approach to fuzzy symmetry detection. In *Proc. Int. Symp. on Graph Drawing*, 166–177.

Chen, H.-L.; Lu, H.-I.; and Yen, H.-C. 2000. On maximum symmetric subgraphs. In *Proc. 8th Int. Symp. on Graph Drawing*, 372–383.

Chuang, M.-C., and Yen, H.-C. 2002. On nearly symmetric drawings of graphs. In *Proc. Information Visualisation*, 489–495.

Cohen, D.; Jeavons, P.; Jefferson, C.; Petrie, K.; and Smith, B. 2005. Symmetry definitions for constraint satisfaction problems. In *Proc. Constraint Programming*, 17–31.

Crawford, J.; Ginsberg, M.; Luks, E.; and Roy, A. 1996. Symmetry breaking predicates for search problems. In *Proc. 5th Int. Conf. on Knowledge Rep. and Reasoning (KR '96)*, 148–159.

de Fraysseix, H. 2000. An heuristic for graph symmetry detection. *Lecture Notes in Computer Science* 1731.

Erdős, P., and Rényi, A. 1963. Asymmetric graphs. *Acta Math. Acad. Sci. Hungar.* 14:295–315.

Fox, M.; Long, D.; and Porteous, J. 2005. Abstaction-based action ordering in planning. In *Proc. Int. Joint Conf. on AI (IJCAI)*.

Manning, J. 1990. *Geometric symmetry in graphs*. Ph.D. Dissertation, Department of Computer Science, Purdoe University.

Markov, I. 2007. Almost-symmetries of graphs. In *Proc. International Symmetry Conference (ISC)*, 60–70.

McKay, B. 1990. *Nauty* user's guide 1.5. Technical Report TR-CS-90-02, ANU, Canberra.

Roy, P., and Pachet, F. 1998. Using symmetry of global constraints to speed up the resolution of CSPs. In *Workshop on Non-binary Constraints, ECAI*.

Yao, J.; Cole, J.; Pidcock, E.; Allen, F.; Howard, J.; and Motherwell, W. 2002. *CSDSymmetry*: the definitive database of point-group and space-group symmetry relationships in small-molecule crystal structures. *Acta Crystallographica Section B* 58(4):640–646.