# A Polynomially Searchable Exponential Neighbourhood for Graph Colouring

Celia A. Glass[1] and Adam Prügel-Bennett[2]

[1] *Cass Business School, City University, London, EC1Y 8TZ, U.K.*

[2] *Department of Electronics and Computer Science, University of Southampton, Southampton, SO17 1BJ, U.K.*

March 7, 2005

**Abstract**

In this paper we develop a new graph colouring strategy. Our heuristic is an example of a so called *polynomially searchable exponential neighbourhood* approach. The neighbourhood is that of permutations of the colours of vertices of a subgraph. Our approach provides a solution method for colouring problems with edge weights. Results for initial tests on unweighted K-colouring benchmark problems are presented. Our colour permutation move was found in practice to be too slow to justify its use on these problems. By contrast, our implementation of iterative descent, which incorporates a permutation kickback move, performed extremely well. Moreover, our approach may yet prove valuable for weighted K-colouring. In addition, our approach offers an improved measure of the distance between colourings of a graph.

**Key words**: Graph colouring, Optimisation, Heuristics, Local search, Exponential neighbourhood.

# Introduction

Graph colouring problems are among the most well studied combinatorial optimisation problems. The most standard graph colouring problem is the *Chromatic Number Problem* (CNP) in which each vertex of a graph must be assigned a colour, subject to the constraint that no two adjacent vertices bear the same colour, in such a way as to minimise the number of colours used. However, for many practical problems the number of colours is fixed, and colour conflicts are not all of the same importance, leading to edges bearing weights. One example is colour coding packaging of products[1]. This, so called *Weighted Colour Conflict Problem*, defined formally below, is the focus of study of this paper.

That solutions to graph colouring problems are equivalent up to permutation of the colours is well known but not easily handled. Most representations of the problem maintain $K!$ equivalent solutions, where $K$ denotes the total number of colours used. The key idea of the algorithm presented in this paper is to exploit this permutation symmetry.

The literature on heuristic approaches to the Chromatic Number Problem is vast. Algorithms which have been used to solve the CNP for large random graphs include greedy based algorithms[2,3], tabu search[4] and genetic algorithms (often incorporating tabu search)[5,6,7]. The most commonly used neighbourhood is the *vertex neighbourhood* which allows for the recolouring of a single vertex. A vertex descent move recolours a vertex so as to achieve the fewest conflicts of the adjacent edges. Most recent advances use information about maximum independent sets of vertices to form candidates for colour classes[8,4,9]. These approaches indirectly tackle the equivalence of solutions up to a colour permutation, but cannot usefully be applied in more general contexts when edge weights are present. An alternative technique, which has proved very successful, is a hybrid algorithm of Galinier and Hao[7] which bypasses the issue of colour permutation. We offer a new approach to overcome colour permutation within heuristics for colouring weighted graphs.

A recent, promising development for combinatorial optimisation is the introduction of *exponential neighbourhoods*. These neighbourhoods are large, their size being inexpressible as a polynomial in the size of the input variables of the problem instance. Exponential neighbourhoods may be useful if a good solution in the neighbourhood can be identified quickly. Therefore one generally looks to find an optimal solution in a number of operations which is a polynomial in the size of the input variables. Neighbourhoods of this type are said to be *polynomially searchable*. Local search moves are usually based upon well known polynomial optimisation algorithms used for classic problems such as Matching, Minimum Spanning Trees and Shortest Path problems.

The Travelling Salesman Problem (TSP) has proved amenable to this approach. A polynomial algorithm for an exponential neighbourhood of the TSP introduced by Saravonov and Doroshko[10], has been developed by Gutin[11] and Punnen[12] and studied by Deineko and Woeginger[13]. Potts and van de Velde employ an alternative exponential neighbourhood search for the TSP based upon Dynamic Programming, which had proved successful for certain scheduling problems[14]. The Quadratic Assignment Problem, by contrast, is thought to be unsuitable for this type of approach[13]. However, no one has yet, as far as we know, applied such an approach to graph colouring problems.

In this paper, we present a novel polynomially searchable exponential neighbourhood for the Weighted Colour Conflict Problem. In the next section, we present the graph colouring problem more formally. The exponential neighbourhood and the polynomial search algorithm are presented in the following section. At the end of that section, we propose a new distance measure for graph colouring which is invariant under colour permutation. We then report on computational experiments on benchmark unweighted graphs, which give some insight into this type of approach. The paper finishes with a discussion of our findings.

# Graph Colouring Problems

The approach proposed in this paper may be applied to a range of graph colouring problems which we first need to specify. Take an undirected graph $G = (V, E)$ consisting of a set $V$ of $N$ vertices and a set $E$ of edges between vertices. Associated with an edge, $(i, j)$ connecting the $i^{th}$ vertex to the $j^{th}$ vertex there is a cost or weight $w_{i\,j} \geq 0$. Any pair of vertices which are not connected are given weight zero. In particular, $w_{ii} = 0$. Pairs of vertices with non-zero weights are said to be *adjacent*. We denote the colour of the $i^{th}$ vertex by $\kappa_i \in \{1, 2, \ldots, K\}$, and the colouring of the whole graph by $\boldsymbol{\kappa}$. We say that a *colour conflict* occurs between adjacent vertices, $i$ and $j$, if they bear the same colour, that is $\kappa_i = \kappa_j$.

We can define the *total weight* or *cost* of a colouring of the graph, which is to be minimised, as

$$W = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} w_{i\,j} [\kappa_i = \kappa_j], \tag{1}$$

where we use square brackets to denote the indicator function

$$[predicate] = \begin{cases} 1 & \text{if } predicate \text{ is true} \\ 0 & \text{if } predicate \text{ is false.} \end{cases}$$

We find this representation more convenient than the equivalent mathematical programming formulation using $0-1$ variables. The task we are interested in is to find a colouring, with $K$ fixed, which minimises $W$. We refer to this as the *Weighted Colour Conflict Problem*. In the absence of different edge weights, the problem is referred to as the *Colour Conflict Problem* or the *K-Colouring problem*.

The standard benchmark problems are, unfortunately, all for graphs without edge weights. We therefore concentrate on these, though in doing so we limit the scope of our algorithm.

We present tests on a class of random graphs $\mathcal{G}_{N,p}$ consisting of $N$ vertices and where each edge occurs with a probability $p$. In particular, we will discuss experiments on $\mathcal{G}_{1000,1/2}$ which has become a standard test problem for new algorithms. We have found little variation in the performance of any search algorithm we tested from one instance of $\mathcal{G}_{1000,1/2}$ to another. This is also a common finding in the literature. The best colouring for any instance of this problem is 83 colours obtained by Galinier and Hao[7]. The large number of colours means there is a huge permutation symmetry in the solution space, which should favour our technique.

# Permutation Descent

In this section we describe an exponential neighbourhood structure for graph colouring problems and show how it can be searched in polynomial time to find a locally optimal solution with respect to the number of colour conflicts. Our approach is based upon the observation that permuting the colours within a subgraph does not affect the contribution to the objective function of either the subgraph itself or its complement. A subgraph, $U$, defines a partition of the graph and hence a *cut*, $(U, \bar{U})$, namely, the edges joining vertices in the subgraph to vertices outside the subgraph. An edge in the cut will contribute to the objective function only if the two vertices of the edge have the same colour. The permutation of colours of the subgraph determines the contribution to the cost made by the edges in the cut. We develop an algorithm to minimise this contribution, in the next section. The structure of the algorithm is as follows.

- select a subgraph $U$
- amalgamate the edges in the cut $(U, \bar{U})$ between vertices with the same pair of colours, adding their weights, to construct a $K \times K$ bipartite graph with one 'super-vertex' for each colour for each of the two sets, $U$ and $\bar{U}$, of the partition
- solve the corresponding Bipartite Matching Problem
- interpret the solution as a permutation of the colouring of $U$, which will be an optimal such permutation, given the colouring of $\bar{U}$.

The permutation move is illustrated by the following example. Consider the colouring on the weighted graph illustrated in figure 1(a), with its associated subgraph $U$. The measure of total weighted colour conflicts for this graph is 9. The amalgamation of vertices with the same colour across the partition between $U$ and $\bar{U}$ results in the Bipartite graph with the matching, shown in figure 2(a), with total cost 9. Solving the Bipartite matching problem then produces a zero cost matching, presented in figure 2(b), whose interpretation in the original graph results in a colouring with no colour clashes as depicted in figure 1(b).

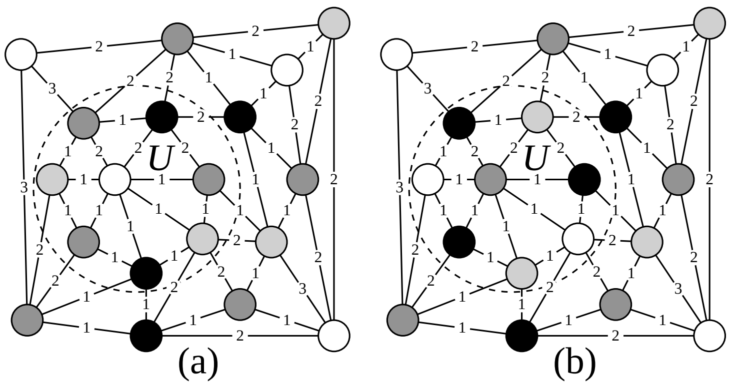


Figure 1: An example of a colour permutation move: (a) shows the colouring before applying the permutation move, and (b) shows the colouring after.

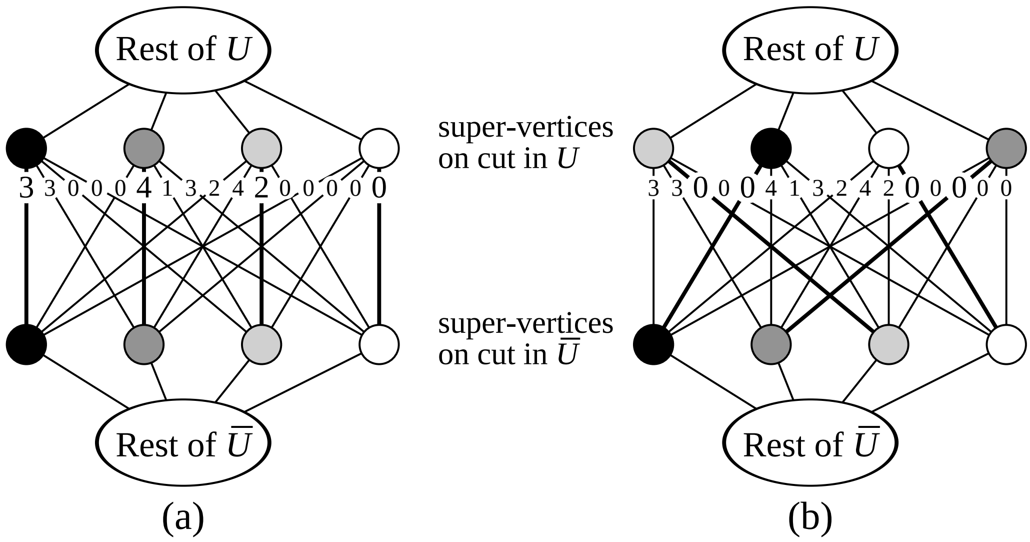


Figure 2: An illustration of the derived Bipartite Matching Problem: (a) shows the weights between the super-vertices in partitions $U$ and $\bar{U}$ corresponding to figure 1(a), and (b) shows the weights after solving the Bipartite Matching Problem.

Observe that vertex descent may be considered a permutation move, with subgraph $U$ consisting of just a single vertex. Permutation descent local optima are consequently a subset of vertex recolouring local optima. The following example illustrates how a permutation move may make improvement where vertex descent cannot. Consider the graph colouring depicted in Figure 3(a), where unit weights are assumed. It is local optimum with respect to vertex moves. However, a permutation move, applied to the three vertices on the right, gives the global optimum, presented in figure 3(b).

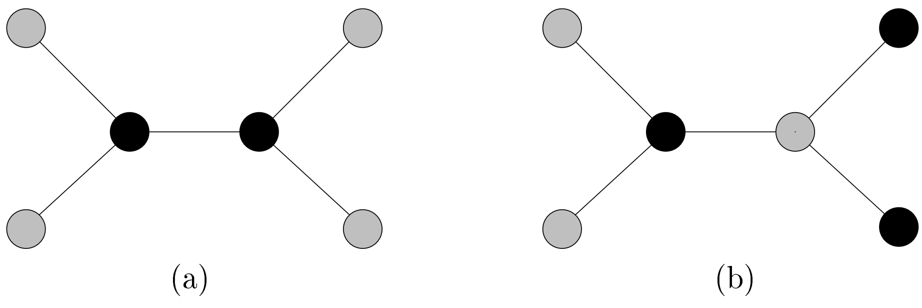


Figure 3: A graph coloured with 2 colours: (a) local optimum with respect to vertex descent, (b) global optimum, reached from graph (a) by a permutation move.

*Colour Permutation Move*

To develop the above ideas more rigorously we need some notation. We define the cost of a subset of the vertices, $U \subset V$, to be

$$W_U = \frac{1}{2} \sum_{i \in U} \sum_{j \in U} w_{ij} [\kappa_i = \kappa_j]. \tag{2}$$

Let $\bar{U}$ denote the complementary set of vertices, i.e. $\bar{U} = V \backslash U$, and $(U, \bar{U})$ denote the cut between $U$ and $\bar{U}$, namely the set of edges

$$(U, \bar{U}) = \{(i, j) | i \in U, j \in \bar{U}\}.$$

If we define

$$W_{(U,\bar{U})} = \sum_{i \in U} \sum_{j \in \bar{U}} w_{ij} [\kappa_i = \kappa_j] \tag{3}$$

as the *interface cost* of the cut, then from equations (1) to (3) we find

$$W = W_U + W_{\bar{U}} + W_{(U,\bar{U})}.$$

Performing a permutation $\pi$ on the set of colours $\{1, \ldots, K\}$, allocated to vertices in one part of the partition, $U$, leaves the colouring of $\bar{U}$, and hence the cost $W_{\bar{U}}$, unaffected. In addition, the cost, $W_U$, associated with subgraph involving vertices in $U$, is unchanged since $[\pi(\kappa_i) = \pi(\kappa_j)] = [\kappa_i = \kappa_j]$. However, the permutation $\pi$ produces a revised interface cost which may be expressed in relation to the original colouring, $\boldsymbol{\kappa} = \{\kappa_i | i \in V\}$, as

$$W_{(U,\bar{U})}(\pi) = \sum_{i \in U} \sum_{j \in \bar{U}} w_{ij} [\pi(\kappa_i) = \kappa_j]. \tag{4}$$

We therefore work with the $K \times K$ *interface cost matrix*, $\mathbf{M}(\boldsymbol{\kappa}, U)$ (referred to as $\mathbf{M}$ if no ambiguity is thus created) of terms, $M_{\mu\nu}(\boldsymbol{\kappa}, U)$ (or $M_{\mu\nu}$)

$$M_{\mu\nu}(\boldsymbol{\kappa}, U) = \sum_{i \in U} \sum_{j \in \bar{U}} w_{ij} [\kappa_i = \mu][\kappa_j = \nu],$$

for $\mu, \nu \in \{1, \ldots, K\}$. The current interface cost may be expressed as

$$\sum_{\mu=1}^{K} M_{\mu\mu}, \tag{5}$$

the sum of the diagonal elements of the interface cost matrix.

We wish to find a permutation $\pi$ of the set of colours $\{1, \ldots, K\}$ that minimises the interface cost between the subgraph $U$ and its complement $\bar{U}$. That is, we wish to minimise the expression

$$W_{(U,\bar{U})}(\pi) = \sum_{\mu=1}^{K} \sum_{\nu=1}^{K} M_{\mu\nu} [\pi(\mu) = \nu]. \tag{6}$$

This can be viewed as a Bipartite Matching Problem, as described above, with the edge between colour vertices $\mu$ in $U$ and $\nu$ in $\bar{U}$ bearing weight $M_{\mu\nu}$. An optimal permutation may therefore be found by the Hungarian Algorithm[15] in $O(K^3)$ time. If we now apply colour permutation $\pi$ to vertices in subgraph $U$, while leaving other vertices unchanged, we then have a new, possibly improved, colouring of the graph.

The nature of the colour permutation neighbourhood depends upon the choice of partition of the vertices. Experiments with different choices of partitions are reported elsewhere[16]. The partitioning which we found most suitable for random graphs is constructed from the so-called *no-conflict* subgraphs, $U$, within which there are no colour conflicts. The general method we use for this is to select one vertex of each colour, abandoning exceptional cases which give unconnected subgraphs. When the edges producing colour conflicts have reduced in number to less than 10 it is preferable to use a cut constructed from these edges. Since the position of the colour conflicts is known from our implementation of vertex descent, both these constructions are $O(|U|)$. We have thus constructed a local search move within the solution space of colourings of the graph, which we shall refer to as *Permutation Descent*.

In local search of the colouring space, one may wish to accommodate alternative objectives, to impose further constraints, or to incorporate a secondary search goal using a Lagrange multiplier. The only requirement of our approach is for entries of the interface matrix $\boldsymbol{M}(\boldsymbol{\kappa}, U)$ to make sense of expressions for current and revised interface costs, (5) and (6). Thus, any similar objective function, or a constraint which is of this form, may be accommodated. Consider, for example, the bounded $K$-colouring problem, in which there is an upper bound, $b_\mu$ say, on the number of vertices which may bear colour $\mu$ (for $\mu = 1, \ldots, K$). To accommodate the bounding constraint, the entries of the colour permutation matrix may be modified to the following expression

$$M_{\mu\nu}(\boldsymbol{\kappa}, U) = \sum_{i \in U} \sum_{j \in \bar{U}} w_{ij} [\kappa_i = \mu][\kappa_j = \nu] + \lambda [\sum_{i \in U} [\kappa_i = \mu] + \sum_{i \in \bar{U}} [\kappa_j = \nu] > b_\mu]$$

where $\lambda$ is a large number. Alternatively, the search may simply be guided by the degree of constraint violation, replacing the above secondary term with

$$\lambda \max\{0, \sum_{i \in U} [\kappa_i = \mu] + \sum_{j \in \bar{U}} [\kappa_j = \nu] - b_\mu\}.$$

## *Exponential Neighbourhood and Polynomial Search*

It is useful to view the colour permutation move described above as searching a local neighbourhood for a best neighbour. For a given subgraph, $U$, the neighbourhood of the current colouring consists of those colourings obtained by a permutation, $\pi$, of the colours of the vertices in that subgraph. We shall refer to this as a *Permutation Neighbourhood*. If the minimum number of colours in either subgraph $U$ or $\bar{U}$ is $k$, then the neighbourhood is of size $K!/(K-k)!$. If the two subgraphs contain all possible colours then this neighbourhood is of size $K!$, which by Stirling's Approximation is $\exp(\,\Theta(K \log(K))\,)$.

Now consider the computational complexity of a Permutation Descent move. Construction of subgraph $U$ and of matrix $\mathbf{M}(\boldsymbol{\kappa}, U)$ requires $O(|U|^2)$ time, while finding an optimal colour permutation associated with the interface cost matrix takes $O(K^3)$ time. Permutation Descent therefore offers a polynomial search of the exponential neighbourhood, each defined in terms of the number of colours $K$. The relative saving in computational effort is measured by $\exp(\,\Theta(K \log(K))\,)/O(K^3) = \exp(\,\Theta(K \log(K))\,)$, in the case when $k = K$. Thus, our approach is most suitable for large graphs with large numbers of available colours. For the Chromatic Number Problem, the input parameter is the number of vertices $N$. However, for the random graphs, $\mathcal{G}_{N,p}$ the chromatic number grows[17] as $N/\log(N)$. Thus, the permutation neighbourhood for random graphs around their chromatic number is $\exp(\,\Theta(N)\,)$. Therefore, our permutation neighbourhood is exponential in the number of vertices of the graph.

### *Measures of Distance Between Colourings*

The the distance between colourings $\boldsymbol{\kappa}$ and $\boldsymbol{\kappa}'$ is commonly measured in the literature by the Hamming Number, namely the number of vertices whose colour differ between the two colourings, i.e. $\sum_{i \in V} [\kappa_i \neq \kappa'_i]$. However, for pairs of solutions which are equivalent up to colour permutation, the Hamming Number is large, thus implying that two colourings are a long way apart. We offer an alternative measure which overcomes this drawback, by working with permutation colour classes, that is sets of colourings which are isomorphic up to colour permutation. A distance measure on colour permutation classes is given by the function $\Delta$, where

$$\Delta(\boldsymbol{\kappa}, \boldsymbol{\kappa}') = \min_{\pi} \sum_{i \in V} [\pi(\kappa_i) \neq \kappa'_i],$$

for any pair of colourings $\boldsymbol{\kappa}$ and $\boldsymbol{\kappa}'$. This distance measure may be calculated, similarly to our subgraph colour permutation move described above, by applying the Hungarian Algorithm to the $|V| \times |V|$ matrix $M(\boldsymbol{\kappa}, \boldsymbol{\kappa}')$ with elements

$$M_{\mu\nu}(\boldsymbol{\kappa}, \boldsymbol{\kappa}') = |V| - \sum_{i \in V} [\kappa_i = \mu][\kappa'{}_i = \nu].$$

## Performance of Permutation Descent

In order to evaluate the effectiveness of our new permutation descent move, we tested it on benchmark problems, for which comparison may be made with the performance of other algorithms. In the absence of benchmark problems with edge weights, we concentrated on unweighted benchmark graph colouring problems. For simplicity we shall confine our discussion to graphs from class $\mathcal{G}_{1000,1/2}$ with a fixed number of colours, $K$, as no further insight was gained from the computational experiments on other problem classes.

Our algorithm **PermImprove** works on a colouring of the graph by constructing a no-conflict subgraph $U$, as described in the section on colour permutation move, and returning a new colouring in which the colours in $U$ have been permuted to minimise the interface cost, using the Hungarian Algorithm. Our implementation of **PermImprove** employed no-conflict clusters of random size, between 20 and 100, until it was down to 50 colour conflicts. When there are few conflicts, a cluster is constructed from a cut which contains as many colour conflicts as is possible, usually all.

On its own **PermImprove** was far too slow to be competitive with vertex descent. As the two neighbourhood structures are complementary, we combine **PermImprove** with a vertex descent algorithm, **VertImprove**. The **VertImprove** algorithm performs 1000 so called *sweeps* of the graph. A sweep involves checking each vertex in the graph in turn and choosing, randomly, a colour for that vertex which gives the lowest cost. Thus, neutral moves are made, but not deteriorating ones. We apply **PermImprove** to solutions which appeared to be locally optimised with respect to vertex descent, i.e. had been subjected to **VertImprove**.

The heuristic is initialised with a random colouring and restarted at the best colouring found, when 5 consecutive runs of **PermImprove** have failed to produce an improvement. Then, when this combined algorithm appears to get stuck we perturb the solution before proceeding. The algorithm employed to do this, called **KickstartP**, performs a single disruptive permutation move (see appendix for details). This seemed to be sufficient not to be undone quickly by the rather powerful vertex descent algorithm.

The result of 1000 second runs on instances of $\mathcal{G}_{1000,1/2}$ are depicted in figure 4. We also tested our heuristic on the standard instance[8] known as DSJC1000.5, and achieved the same results. The results show that the addition of **PermImprove** has a deteriorating, and not an improving, effect on **VertImprove** with a kickstart. However, the kickstart component plays an important role, providing significant improvement especially when the number of available colours is restricted.

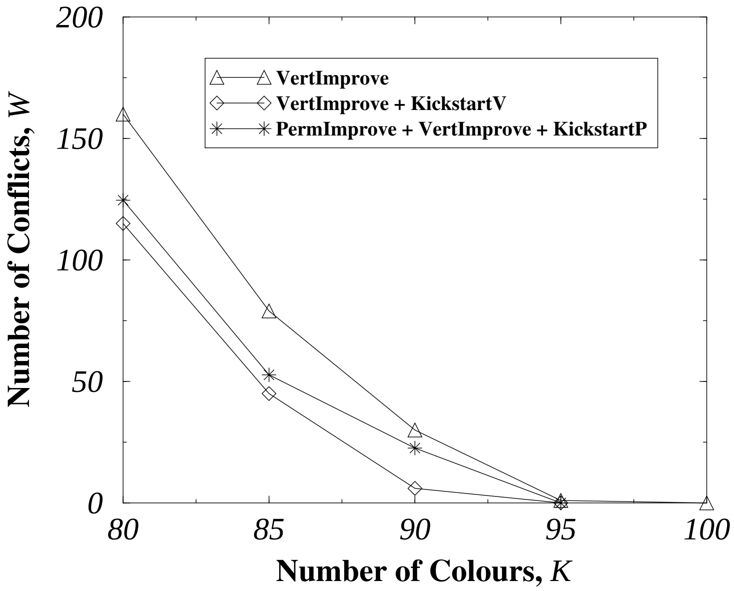


Figure 4: Comparative performance of hybrid algorithms, constructed from **VertImprove**, **PermImprove** and a kickstart component, for some typical instances of $\mathcal{G}_{1000,1/2}$.

One would expect **PermImprove** to make changes to the colouring only rarely, and then most frequently to make neutral moves. From experimentation we discovered that only around 4% of the vertices can be recoloured to one other colour without causing a new colour conflict, even in the initial colouring presented to **PermImprove**, after 1000 sweeps of **VertImprove**. However, neutral moves may be very valuable as they are complementary to vertex descent and therefore encourage a more extensive search of the solution space. There are, thus, two explanations which we may speculate for our findings. One is that the solution space has many well spread (local) optima, obviating the need for extensive exploration of the space. The other is that the benefit of the time spent on many non-improving moves using **PermImprove** is outweighed by the consequent reduction in vertex descent search, which is very quick.

It is an open question whether **PermImprove** would be competitive with a more sophisticated algorithm for constructing the partition. The authors have failed to find such an algorithm, at least, for $\mathcal{G}_{1000,1/2}$. This may in part be due to the class of problems we have investigated. Random graphs of the type we have studied have no local structure which we can exploit in constructing a subgraph. It may be easier to find good partitions for a sparse graph or one with some geometric structure. However, the efficiency of our search also depends on the number of colours. In this respect, the random graphs we used appeared an ideal testing ground for our exponential neighbourhood.

As the vertex descent and permutation kickstart components proved to be so successful a few implementation details are given in the appendix.

# Conclusions

This paper contributes to recent research into polynomially searchable exponential neighbourhoods for combinatorial optimisation. It offers a new type of search, which is applicable to a very broad range of graph colouring problems. At its heart, our **Permutation Descent** employs the Hungarian Algorithm to obtain an optimal (re)colouring of a subgraph in polynomial time. The neighbourhood move is suitable for inclusion in hybrid local search algorithms. It has an impressive theoretical efficiency, searching $K!$ colourings in order $K^3$ time, which for random graphs with $N$ vertices, gives a computational efficiency of $\mathrm{e}^{\Theta(N)}$ A by-product of our approach is a colouring distance measure which takes account of permutation isomorphism, and is offered as a replacement for the Hamming Distance.

Computational experiments on the classic benchmark Chromatic Number Problem on random graphs $\mathcal{G}_{1000,1/2}$, are reported. Our implementation incorporated permutation descent moves in a vertex descent colouring algorithm with multiple kickstarts (restarts performed after slightly perturbing the solution). We discovered that, contrary to expectation, the heuristic performed better without the permutation descent moves, though colour permutation kickstart proved to be valuable.

One lesson to be learnt from our investigations is that even very promising polynomially searchable exponential neighbourhoods may in practice be difficult to exploit. Only computational experimentation can ascertain their competitiveness. It is therefore important to report such negative results.

The failure of permutation descent, within the vertex descent algorithm, for graph colouring, may mainly be due to the relative slowness of a permutation descent move, which is $O(K^3)$. Moreover, the similarity of the solution landscape, may also contribute by making neutral moves of exploration of limited value. The performance of our implementation of vertex descent with permutation kickstart is remarkably good. We achieved a solution with 89 colours for $\mathcal{G}_{1000,1/2}$, which is as good as tabu search by itself.

Despite these initial disappointing results we believe that our Permutation Descent algorithm may yet prove to be useful, as it has more flexibility than these initial experiments test. For example, Permutation Descent is applicable to weighted graphs, for which efficient algorithms, such as those using independent sets, fail. The most promising direction for future research using subgraph permutation is within a Genetic Algorithm.

*Acknowledgements*

We would like to thank Alain Hertz for his initial encouragement and interest in our approach, and two anonymous referees for constructive suggestions which have improved the presentation in this article.

### *Appendix: Implementation details of vertex descent and permutation kick-starts*

We implemented vertex descent efficiently by first computing a table giving the relative cost of colouring each vertex, $i$, to a particular colour, $\mu$, (with all other vertices left the same)

$$c(i, \mu) = \sum_{j \neq i} w_{i\,j}[\mu = \kappa_j].$$

The 'best colours' for a vertex are those for which $c(i, \mu) \leq c(i, \nu)$ for all $\nu$. From our table we compute a *colour list*, $L(i)$s of best colours for each vertex. To perform vertex descent for a vertex $i$ we choose one of the colours in the colour list $L(i)$. When a vertex changes colour we update the colour table for all its adjacent vertices and if necessary their colour lists. Keeping this information is efficient because, for any reasonable colouring, the vast majority of vertices have only one element in its list and so do not need to be updated. The overhead of computing the original cost table is reduced by performing many sweeps at a time.

The disruptive permutation move used in **KickstartP** is performed as follows. A subgraph is chosen randomly and the corresponding permutation matrix on colour classes constructed as for **PermImprove**. The minimum non-zero element in the matrix is then added to the (zero) elements of the diagonal. If this is not sufficient to ensure that the Hungarian algorithm finds a different solution, then the number along the diagonal is increased until it does. In the absence of permutation descent, kickstart was performed differently. **KickstartV** randomly selects a fraction, 5%, of the vertices and randomly recolours them.

# References

[1] Glass C. A. Bag rationalisation by near-optimal colouring. *Journal of the Operational Research Society*, 53:544–551, 2002.

[2] D. Brélaz. New methods to color vertices of a graph. *Communications of the ACM*, 22:251–256, 1979.

[3] J. C. Culberson and Feng Luo. Exploring the $k$–colorable landscape with iterated greedy. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, 1993*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 245–284. American Mathematical Society, 1996.

[4] A. Hertz and D. De Werra. Using tabu search techniques for graph coloring. *Computing*, 39:345–351, 1987.

[5] C. Fleurent and J. A. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63:437–461, 1996.

[6] Daniel Costa, Alain Hertz, and Olivier Dubuis. Embedding a sequential procedure within an evolutionary algorithm for coloring problems. *Journal of Heuristics*, 1:105–128, 1995.

[7] P. Galinier and J. K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3(4):379–397, 1999.

[8] D. S. David S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part II graph coloring and number partitioning. *Operations Research*, 39(3):378–406, May-June 1991.

[9] Craig Morgenstern. Distributed coloration neighborhood search. In David S. Johnson and Michael A. Trick, editors, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, 1993*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 335–357. American Mathematical Society, 1996.

[10] Sarvanov V. I. and Doroshko N. N. Approximate solution of the traveling salesman problem by a local algorithm with scanning neighbourhood of factorial cardinality in cubic time. *Software: algorithms and programs, Mathematical Institute of the Belorussian Academy of Science*, 31:11–13, 1981.

[11] Gutin G. Exponential neighbourhood local search for the traveling salesman problem. *Computers and Operations Research*, 26, 1999.

[12] Punnen A. P. Traveling salesman problem: new polynomial approximation algorithms and domination analysis. Research report, University of New Brunswick, Canada, 1996.

[13] Deineko V. and Woeginger G. J. A study of exponential neighborhoods for the travelling salesman problem and for the quadratic assignment problem. *Mathematical Programming*, 87:519–542, 2000.

[14] Congram R., Potts C. N., and van de Velde S. L. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *Informs Journal of Computing*, 14:52–67, 2002.

[15] Carpaneto G. and Toth P. Algorithm 548: Solution of the assignment problem. *ACM Transactions of Mathematical Software*, 6(1):104–111, 1980.

[16] C. A. Glass and A. Prügel-Bennett. A polynomially searchable exponential neighbourhood for graph colouring. Technical report, DASS, City University, London, UK., 2000.

[17] Bollobás B. The chromatic number of random graphs. *Combinatorica*, 8(1):49–55, 1988.