12-7-2010

Modifications and additions to ant colony optimisation to solve the set partitioning problem

Marcus Randall
Bond University, marcus_randall@bond.edu.au

Andrew Lewis
Griffith University

2010 Sixth IEEE International Conference on e–Science Workshops

Modifications and Additions to Ant Colony Optimisation to Solve the Set Partitioning Problem

Marcus Randall
School of Information Technology
Bond University
Australia
mrandall@bond.edu.au

Andrew Lewis
Institute for Integrated and Intelligent Systems
Griffith University
Australia
a.lewis@griffith.edu.au

Abstract— **Ant colony optimisation has traditionally been used to solve problems that have few/light constraints or no constraints at all. Algorithms to maintain and restore feasibility have been successfully applied to such problems. Set partitioning is a very constrained combinatorial optimisation problem, for which even feasible solutions are difficult to construct. In this paper a binary ant colony optimisation framework is applied to this problem. To increase its effectiveness, feasibility restoration, solution improvement algorithms and candidate set strategies are added. These algorithms can be applied to complete solution vectors and as such can be used by any solver. Moreover, the principles of the support algorithms may be applied to other constrained problems. The overall results indicate that the ant colony optimisation algorithm can efficiently solve small to medium sized problems. It is envisaged that in future research parallel computation could be used to simultaneouly reduce solver time while increasing solution quality.**

I. Introduction

The processing of constraints under meta-heuristic search algorithms is problematic and often done in an ad-hoc way. A variety of methods exist ranging from penalty functions, moving between feasible points using specialised operators and restoring feasibility (see Coello Coello [9] for a survey of these). Constructive approaches such as Ant Colony Optimisation (ACO) (see Dorigo and Di Caro [15] for an overview) present their own unique problems. The largest of which is that during the construction it is not possible to determine whether the final solution will be feasible. For other meta-heuristics that operate on full solutions (such as simulated annealing, tabu search and genetic algorithms), optimisation can be guided more easily through feasible space as feasibility can be determined after each move.

Highly constrained optimisation problems such as the set partitioning problem (SPP) offer greater challenges to solve effectively. Highly constrained problems may be thought of as those having a high proportion of equality constraints. These problems have traditionally been in the domain of specialised heuristics and integer linear programming techniques rather than of meta-heuristics [1]. This is because meta-heuristics find it difficult to obtain and maintain feasible solutions to these problems. The most effective implementation to date has been Chu and Beasley's genetic algorithm (GA) [8]. Specialised genetic operators were used to generate solutions. In this paper ACO is used as a medium to create solutions from which the feasibility restoration and improvement can be applied.

The paper is organised as follows. Section II gives a description of a typical highly constrained optimisation problem, namely the SPP as well as a summary of the meta-heuristic approaches used to solve it. Section III describes the modifications and additions (e.g., feasibility restoration and solution improvement) necessary for ACO to effectively solve the SPP. Sections IV and V give the computational results and conclusions respectively.

II. The SPP and Meta-heuristics

The SPP is an extremely practical combinatorial optimisation problem for which there exists a variety of applications (see Balas and Padberg [2] for an extensive overview). The most well known of these is the aircraft crew scheduling problem [20]. The SPP is traditionally expressed as a 0-1 integer linear programme [8] given in Equations 1-3.

$$\text{Minimise } \sum_{i=1}^{N} c_i x_i \tag{1}$$

s.t.

$$\sum_{i=1}^{N} a_{ij} x_i = 1 \qquad 1 \leq j \leq M \tag{2}$$

$$x_i = \{0, 1\} \qquad 1 \leq i \leq N \tag{3}$$

Where:

N is the number of columns,
c_i is the cost of the i^{th} column,
x_i is 1 if column i is in the solution, 0 otherwise,
a_{ij} is 1 if the j^{th} row is covered in the i^{th} column and
M is the number of rows.

There have been relatively few attempts at using standard meta-heuristics to solve highly constrained problems such as the SPP. Abramson, Dang and Krishnamoorthy [1] are able to solve relatively small SPP instances to optimality using two variations of simulated annealing. Crawford and Castro [13] present an ACO algorithm that incorporates constraint programming lookahead functions that again is able to solve small problems. Due to the large amount of computational resources required, Levine [21] and Czech [14] have created parallel genetic algorithms and simulated annealing solvers respectively.

978-0-7695-4295-9/10 $26.00 © 2010 IEEE
DOI 10.1109/eScienceW.2010.27

Chu and Beasley's [8] GA approach uses an augmented penalty method in order to produce feasible solutions. Rather than applying a simple penalty term to the fitness (objective) function, they separate this into two distinct functions, namely fitness and unfitness. The fitness is the original objective function while the unfitness function is the cumulative measure of the number of times each row is over or undercovered. Their GA consists of tailored genetic operators, parent selection techniques and a heuristic to improve the feasibility of child solutions (though not necessarily guarantee feasibility). The results showed that not only could feasible solutions be obtained, but many of these were optimal (even to large size problems).

According to Maniezzo and Milandri [24], the traditional ACO framework is ineffectual for solving the SPP. Hence they use a modified approach in which some elements of ACO are combined with tree search procedures. This becomes a form of branch and bound. At each step of their modified algorithm, the partial solution is expanded with up to k nodes (columns) per ant. The k nodes are produced with the tree search algorithm. Pheromone is applied to the coupling of how well column i covers constraint/row j. The results, while not as effective as Chu and Beasley's [8] genetic algorithm, show that such an approach is viable.

III. ACO FOR HIGHLY CONSTRAINED PROBLEMS

Much research and development effort into ACO has been concentrated on problems whose solutions can easily be represented with natural encodings (i.e., non 0-1 values). This ensures that it is relatively easy to obtain feasible solutions. As examples of this consider problems such as the TSP (e.g., [16], [17], [29]) QAP (e.g., [11], [19], [23], [28]) and job and machine scheduling (e.g., [3], [10], [30]). In comparison, even relatively lightly constrained problems such as the vehicle routing problem (e.g., [7], [18], [27]) and the generalised assignment problem (GAP) (e.g., [22], [26]) have received less attention. For highly constrained problems, Maniezzo and Milandri [24] have presented a hybrid ACO/tree search solver for the SPP. As outlined in Section II, the majority of this approach is based on tree search, rather than pure ACO strategy.

As for some other ACO implementations (for example Blum, Roli and Dorigo [5] and Bu, Yu and Guan [6]) it is sensible to encode the SPP solution as a binary vector. Each solution component is a column that is allowed to take on either of two values, 0 or 1. Pheromone (τ) in this case is given as $\tau(i,k)$ where i and k are integers and $1 \leq i \leq N$ and $k \in \{0,1\}$. Note that this is an $O(N)$ memory requirement whereas Maniezzo and Milandri's [24] is $O(NM)$. One of the most important questions for a constructive meta-heuristic such as ACO is how each ant at each step will decide which value to assign to the current component. For a highly constrained problem such as the SPP, a balance between optimising the objective measures and satisfying constraints (i.e., each row is covered once) is needed. To this end, Equations 4 and 5 represent these two possibilities. Equation 4 rewards good combinations of low cost columns that cover a large number of rows while Equation 5 is concerned with columns that have a greater number of rows.

$$\eta(i) = d \times \frac{\big(p - c(i)\big) \times r(i)}{p \times s} \tag{4}$$

$$\eta(i) = d \times \frac{r(i)}{s} \tag{5}$$

Where:

i is the column ($1 \leq i \leq N$),
d is the conflict status. If incorporating this column into the solution causes a row or rows to be over covered, it becomes 0, else it is 1,
p is the maximum cost of all columns,
$c(i)$ is the cost of column i,
$r(i)$ is the number of rows covered by column i and
s is the maximum number of rows covered by a column.

Both heuristics can be used in a manner outlined by Randall [26] (which is referred to as adaptive) for the GAP. This technique switches between the two forms of the heuristic based on the solver's ability to produce feasible solutions. For instance, if a colony of ants (at a particular iteration) fails to find a feasible solution amongst them, the next iteration can use a heuristic such as Equation 5. Similarly, if feasible solutions are frequently found, Equation 4 should be used instead.

Another important issue is the order in which the solution components (columns) are processed by the ants. This is known as the assignment order [12] and can make a significant impact on solution quality [12], [26]. Three general purpose structures (though discussed here in terms of the SPP) are as follows. Note that each variation is given a code to which it is referred in Section IV.

1) *Normal Order* – That is to assign a 0-1 value to each column in sequential order, i.e., $1 \ldots N$. This is referred to as `ao1`.
2) *Weighted Order* – Weight can be thought of as the column cost or how many rows it covers. The first simply means that it is more desirable to begin a solution using columns with smaller costs. For the second, the columns can be ordered by the number of rows they cover (whether in ascending or descending order). These are referred to as `ao2`, `ao2asc` and `a02dec` respectively.
3) *Random Order* – Components are ordered in a random way. Unlike approaches 1) and 2), the order can be reassigned at each iteration, or for each ant at each iteration. This is referred to as `ao3`.

Beyond these essential parts of an ACO implementation, special feasibility, improvement and candidate set strategies are described next.

A. Restoring Feasibility

As SPP is a highly constrained problem, an ACO solver (as described above) will find it difficult to produce feasible solutions. In fact, initial testing of the solver showed that it could only produce up to 2% feasible solutions across a range of problems. For larger problems, it was often the case that no feasible solutions could be constructed.

To increase the number of feasible solutions, it is desirable to have an algorithm which can transform an infeasible solution into a feasible one. This type of algorithm is generally referred to as a feasibility restoration algorithm [25]. It has been traditionally considered too difficult to devise an algorithm for SPP that will generate feasible solutions [8]. However, a simple $O(N^2)$ strategy is to take an existing (infeasible) solution and search for columns (not presently in the solution) that best cover its uncovered rows. If the newly added column conflicts with any other column(s) currently in the solution, these are removed. Once again (in the worst case) an undercovered solution is produced. By repeating this procedure for a number of iterations, a feasible solution may be produced. To determine which column to add next, a heuristic scoring system is used. This takes the two factors into account, namely the number of extra rows covered by the column and the number of rows that are overcovered. The latter is a more important consideration as the goal is to obtain a feasible solution. In the implementation, the values of 1 and 10 respectively were found to be suitable for this purpose.

Termination of the algorithm can be either when a feasible solution is found or a maximum number of iterations has elapsed (the number of columns N is a sensible value). Columns are allowed to be changed only once in the execution of the algorithm. The pseudocode for feasibility restoration is given in Algorithm 1.

Initial tests of the algorithm indicate that feasible solutions are produced in nearly all cases. In fact, without it, the solver is unable to create feasible solutions to most of the test problems (as used in Section IV).

Algorithm 1 The feasibility restoration algorithm as applied to the entire colony of ants.

```
1:  for each of the infeasible ant solutions do
2:    $feasible$ = false
3:    $iterations = 0$
4:    while $feasible$ = false AND $iterations < N$ do
5:      Determine which rows are not covered by the
        solution
6:      if there are uncovered rows then
7:        $best\_column$ = Determine the best column to
          cover the rows
8:        Place the $best\_column$ in the solution and re-
          move those that conflict with it (i.e., have rows
          in common)
9:      else
10:       $feasible$ = true
11:     end if
12:     Increment $iteration$
13:   end while
14:   if $feasible$ = true then
15:     Recalculate the objective cost
16:   end if
17: end for
18: end

19: Determine the best column to cover rows
20: $best\_score = 0$
21: for all columns that are not currently in the solution or
    have been changed by the feasibility restoration algo-
    rithm do
22:   $score$ = (number extra rows covered
      $\times$ $score\_extra\_rows$) $-$ (number of rows overcovered
      $\times$ $overcover\_score$)
23:   if $score > best\_score$ then
24:     $best\_score = score$
25:     $best\_column = column$
26:   end if
27: end for
28: return $best\_column$
29: end Determine the best column to cover rows
```

1) Improving Solution Costs: While the feasibility restoration algorithm is able to create a feasible solution from an infeasible one, it does this without regard to solution cost. It should be noted that local search/gradient search techniques are not effective for this problem [8]. Therefore two different ways may be used to improve the quality of the solutions produced by this process. These may be used either separately or in conjunction and are as follows, where v is the biasing factor:

1) *Bias towards cheaper columns*: Referred to as "IS1". As the algorithm changes part of an ant's solution, a reasonable strategy is to bias the heuristic so that it incorporates columns of lower cost. In this fashion, line 22 can be changed to the following, where v is the biasing factor:

   $score$ = (number extra rows covered $\times$ $score\_extra\_rows$) $-$ (number of rows overcovered $\times$ $overcover\_score$) $+$ $cost\_score \times v(column)$

   Where:

   $v(column) = 1 - \frac{c(column)}{p}$ and
   p is the maximum cost of all columns.

   The value of $cost\_score$ varies dynamically throughout a run of a solver in a manner similar to selecting the component selection heuristic. This means that it increases the value of the parameter if the solver is able to find feasible solutions. If, however, using lower cost columns makes it difficult for the restoration process to produce feasible solutions, the value can be decreased. This is another example of an adaptive setting.

2) *Removing high cost columns*: Referred to as "IS2". Further improvement can be made after a feasible solution is obtained. To do this, the solution can be analysed to determine a cluster of high cost columns.

These columns can be removed from the solution (which of course then becomes undercovered) and the feasibility restoration process can be applied. To prevent the algorithm from reinstating the cluster of columns (or part thereof), they are given a tabu status. Algorithm 2 shows how this may be applied to a single solution.

Algorithm 2 The improvement algorithm extension that incorporates feasibility restoration.

```
1: $t$ = number of columns set as 1
2: $max\_diff = 0$
3: $b$ = Sort columns from biggest to smallest
4: for $col = 1$ to $t - 1$ do
5:     $diff = b(col) - (col + 1)$
6:     if $diff > max\_diff$ then
7:         $max\_diff = diff$
8:         $break\_point = col$
9:     end if
10: end for
11: Change $b(1 \ldots break\_point)$ to 0 and make them tabu
    in the restoration process
```

It is possible that feasibility cannot be restored or a solution of higher cost is obtained. In both cases, the original solution is reinstated.

B. Candidate Set Strategies

Reasonable and practical size set partitioning problems tend to have many thousands of columns. Given that the ACO strategy will for each ant at each step evaluates each column and then (most likely) invoke feasibility restoration, computational time will be excessive. With sufficient parallel computing resources, this issue is mitigated by the fact that evaluation for each ant can be computed independently, and concurrently. In the absence of parallel computational facilities, and particularly for the computational experiments described in this paper for which only a uniprocessor-based computer was available, a standard technique, known as candidate sets, can be used. Candidate sets sensibly limit the number of evaluations that are done at each step, by, in this case, reducing the number of columns. Herein, three strategies are developed:

1) **Random** – For each iteration of the algorithm, each ant chooses a subset of columns (with which to work) at random.
2) **Column Compatibility** – Due to the constraintedness of these problems, a sensible strategy is to confine attention to a subset of columns that have a greater chance of producing feasible solutions. Each column's compatibility is calculated as the inverse of the aggregated number of rows it has in common with the rest of the columns. The columns with the least number of clashes receive higher rankings.
3) **Column Compatibility (probabilistic)** – This is a combination of the above two. The rank of the column is used to compute a probability of it being chosen as part of the candidate set. The better ranked columns receive higher probabilities than lesser ranked ones.

The size of the candidate sets needs to be set for all techniques. This is specified as a percentage. Techniques 1 and 3 can be recomputed at each iteration of the search.

IV. Computational Experiments

The computational platform used is a uniprocessor Pentium 4 running at 3 GHz. Ant Colony System (ACS) [17] is used as the base search technique. The standard ACS parameters[1] are chosen as $\{\beta = 2, \gamma = \rho = 0.1, m = 10\}$. These are typical values that are considered robust across problems [17]. Initial experimentation revealed that the setting $q_0 = 0.1$ (i.e., a less greedy to choice of the inclusion of columns) was suitable for the SPP. Each test instance will be run across ten random seeds for one thousand iterations. The set of test problems is given in Table I and is divided into small and large instances. These are a cross-section of problems from the World OR-Library [4].

The aim of these experiments is to, in as reasonable amount of computational time possible, test the support heuristics and strategies for ACO. Specifically, these are the use of the visibility heuristic (η), solution improvement, assignment orders and candidate set strategies.

The experiments are subsequently divided into three phases. These phases all use the feasibility restoration heuristic, however, Phases 1 and 2 are run on the small test problems to determine suitable parameter and algorithmic settings.

Phase 1: The combination of assignment order and visibility heuristic is tested. The settings for each are: assignment order {`ao1`, `ao2`, `a02asc`, `a02dec`, `a03`} and visibility heuristic {Equation 4, Equation 5, adaptive}. The Cartesian product of these sets gives fifteen possibilities. The Kruskal Wallis statistical procedure will be used to test for any significant differences. If these are detected, post-hoc testing is then used to rank the combinations. The best of these will be used as the basis for Phase 2. One thousand ant colony iterations per test instance will be used in this exploratory phase.

Phase 2: The aim is to the solution cost improvement technique IS2 described in Section III-A. IS1 is used as default as it is able to bias the process towards cheaper columns without any additional computational expense. Results are generated with IS2 turned off and on. Further to this, a simple multistart search has been implemented. This algorithm produces random solutions that are each subjected to the feasibility restoration and improvement processes. To allow a fair comparison, the multistart algorithm produces the same number of solutions per run as ACO.

Phase 3: Using the best combination of settings as determined from Phases 1 and 2, the solver is run on the

[1]β is the weighting factor of the heuristic within the probabilistic selection equations, ρ is the local pheromone decay value, γ is the global pheromone updating value and m is the number of ants.

largest problems in the test set. In terms of the candidate sets, initial investigation revealed that of the three types, Compatibility (probabilistic) produced the best results, even with small set sizes. Therefore this will be used with a set size that for each problem instance allows up to one thousand columns.

A. Results

The Phase 1 results were generated in order to detect if there were any significant differences due to assignment order and visibility heuristics. This was gauged using the best solution value produced in each run. On both counts this proved to be the case (using a significance level of 0.05). Post-hoc testing revealed that it is best to use a random assignment order combined with either the adaptive or greedy visibility heuristic (Equation 4).

In Phase 2, IS2 was tested. Table II reveals that improved solutions could be found using it rather than not. This was statistically significant at the 0.05 level. The question then becomes, does this improvement technique require distinctly more computational time? For problems in which the optimal solution could not be found in the one thousand iterations, IS2 typically required around a third more computational time. This was for the extra feasibility restoration attempts that form part of the technique. On the problem instances for which the optimal cost could be obtained, the median runtime result was always lower for IS2. This suggest that it generally requires fewer iterations to receive the same cost value. The multistart algorithm was in general very poor compared to ACO. This is despite the fact that it uses the same support heuristics. It suggests that ACO is capable of driving the search towards better solutions.

The Phase 3 results are presented in Table III. A property of the ACO solver that emerged after much initial computational experimentation was that it dealt very poorly with problems having over one thousand columns. Runtimes became exceedingly long and the returned solutions were relatively poor. Hence the use of candidate set strategies became necessary for solving larger instances. However, while they are useful for reducing the amount of columns and subsequently the overall runtime, an unpleasant side effect is, of course, that the ability of the solver to construct quality solutions is reduced. Indeed with the setting of very small values of the candidate set percentage parameter, it was found the solver could not construct a feasible solution. Allowing the solver to access up to one thousand columns of the problem typically produced solutions within ten percent of the optimal cost (though NW11 went close to 45%) taking between 3000 and 10000 CPU seconds (largely dependent on the number of rows). These results are certainly not yet as effective as the specialised meta-heuristics of Chu and Beasley [8] and Maniezzo and Milandri [24]. However, it must be kept in mind, that the general support algorithms are very useful for highly constrained problems.

Given that the feasibility restoration algorithm is effective when the full set of columns is used, it would seem that for ACO, parallel processing strategies for this and similar problems will need to be explored. This should help in regard to producing better solutions in substantially less wall-clock time. More investigation also needs to be carried out on the creation of suitable subsets of columns to constitute candidate sets, particularly for cases where limited computational resources must be used.

TABLE III
THE PHASE 3 RESULTS FOR THE LARGE PROBLEM INSTANCES EXPRESSED AS RPDS.

Problem	min	med	max
NW11	36.02	43.64	44.51
KL01	5.16	8.01	9.76
NW07	7.84	9.34	9.76
NW09	11.8	15.6	15.6
NW19	9.51	12.08	12.24
NW29	6.69	6.69	6.69
NW30	4.21	4.21	4.21
NW31	0.35	0.7	0.7
NW33	4.52	5.81	7.61
NW35	0	1.39	1.39
NW36	0.11	0.14	0.14

V. CONCLUSIONS

Highly constrained combinatorial optimisation problems have proved to be a challenge for meta-heuristic solvers (particularly for those that construct solutions). Often, the incorporation of specialised heuristics is necessary in order to produce practical implementations. In this paper however, algorithms have been developed so that they (in principle) may be applied to a range of constrained problems. The feasibility restoration procedure is able to move components in and out of a solution so as to satisfy a set of equality constraints. This could be easily adapted to support inequality constraints. The improvement technique builds on feasibility restoration to encourage solutions of better cost to be produced. This is especially important when effective local search algorithms are not available.

For this implementation, the ACO algorithm and its supporting components, clearly are effective on the smaller sized problems. This is despite the assertion of Maniezzo and Milandri [24] that the standard ACO framework is not suited to the SPP. Due to the many thousands of columns in the larger instances, ACO slowed substantially (to a point where it was not practicable to run). This aspect of scalability requires significant attention. The candidate set techniques used to reduce the number of columns, allowed the algorithm to process these problem instances and produce feasible solutions. While this is a positive outcome, the aim, of course, is to improve the quality of these results. Therefore, the next stage of the research is to modify the algorithm to suit parallel computation. By lessening the reliance on candidate sets it is hoped that the feasibility restoration process will have sufficient columns to operate more effectively and deliver improved reults. Ultimately, this will enable the effective solving of SPP and similar problems by modified ACO. It would appear that for this class of problems, the use of parallel computing is not just desirable but a practical necessity.

References

[1] D. Abramson, H. Dang, and M. Krishnamoorthy. A comparison of methods for solving 0-1 integer programs using a general purpose simulated annealing algorithm. *Annals of Operations Research*, 63:129–150, 1996.

[2] E. Balas and M. Padberg. Set Partitioning - A Survey. In N. Christophides, A. Mingozzi, P. Toth, and C. Sandi, editors, *Combinatorial Optimization*, pages 151–210. John Wiley, 1979.

[3] A. Bauer, B. Bullnheimer, R. Hartl, and C. Strauß. Minimizing total tardiness on a single machine using ant colony optimization. *Central European Journal of Operations Research*, 8:125–141, 2000.

[4] J. Beasley. OR-Library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.

[5] C. Blum, A. Roli, and M. Dorigo. HCACO: The hypercube framework for ant colony optimization. In *Proceedings of the* 4^{th} *Metaheuristics International Conference*, 2001.

[6] T. Bu, S. Yu, and H. Guan. Binary-coding-based ant colony optimization and its convergence. *Journal of Computer Science and Technology*, 19(4):472–478, 2004.

[7] B. Bullnheimer, R. Hartl, and C. Strauß. An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research*, 89:319–328, 1999.

[8] P. Chu and J. Beasley. Constraint handling in genetic algorithms: the set partitioning problem. *Journal of Heuristics*, 4:323–357, 1998.

[9] C. Coello Coello. A survey of constraint handling techniques used with evolutionary algorithms. Technical Report Lania-RI-9904, Laboratorio Nacional de Informtica Avanzada, 1999.

[10] A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian. Ant system for job-shop scheduling. *JORBEL - Belgian Journal of Operations Research, Statistics and Computer Science*, 34(1):39–53, 1994.

[11] O. Cordón, I. de Viana, and F. Herra. Analysis of the best-worse ant system and its variants on the QAP. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Third International Workshop on Ant Algorithms, ANTS 2002*, volume 2463 of *Lecture Notes in Computer Science*, pages 228–234. Springer-Verlag, Brussels, Belgium, 2002.

[12] D. Costa and A. Hertz. Ants can colour graphs. *Journal of the Operational Research Society*, 48:295–305, 1997.

[13] B. Crawford and C. Castro. ACO with lookahead procedures for solving set partitioning and covering problems. Workshop proceedings of "Combination of Metaheuristic and Local Search with Constraint Programming Techniques", 2005.

[14] Z. Czech. A parallel genetic algorithm for the set partitioning problem. In 8^{th} *Euromicro Workshop on Parallel and Distributed Processing*, pages 343–350, 2000.

[15] M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, London, 1999.

[16] M. Dorigo and L. Gambardella. Ant colonies for the traveling salesman problem. *BioSystems*, 43:73–81, 1997.

[17] M. Dorigo and L. Gambardella. Ant Colony System: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[18] L. Gambardella, E. Taillard, and G. Agazzi. MACS-VRPTW - A multiple ant colony system for vehicle routing problems with time windows. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 63–76. McGraw-Hill, London, 1999.

[19] L. Gambardella, E. Taillard, and M. Dorigo. Ant colonies for the quadratic assignment problem. *Journal of the Operational Research Society*, 50:167–176, 1999.

[20] K. Hoffman and M. Padberg. Solving airline crew-scheduling problems by branch-and-cut. *Management Science*, 39:657–682, 1993.

[21] D. Levine. A parallel genetic algorithm for the set partitioning problem. In I. Osman and J. Kelly, editors, *Meta-Heuristics: Theory and Application*, pages 23–35. Kluwer Academic Publishers, Norwell, MA, 1996.

[22] H. Lourenco and D. Serra. Adapative search heuristics for the generalized assignment problem. *Mathware and Soft Computing*, 9:209–234, 2002.

[23] V. Maniezzo and A. Colorni. The ant system applied to the quadratic assignment problem. *IEEE Transactions on Knowledge and Data Engineering*, 11(5):769–778, 1999.

[24] V. Maniezzo and M. Milandri. An ant-based framework for very strongly constrained problems. In M. Dorigo, G. Di Caro, and M. Sampels, editors, *Third International Workshop on Ant Algorithms, ANTS 2002*, volume 2463 of *Lecture Notes in Computer Science*, pages 222–227, Brussels, Belgium, 2002. Springer-Verlag.

[25] M. Randall. Feasibility restoration for iterative meta-heuristic search algorithms. In *Lecture Notes in Artificial Intelligence*, volume 2358, pages 168–178. Springer-Verlag, Berlin, 2002.

[26] M. Randall. Heuristics for ant colony optimisation using the generalised assignment problem. In *Proceedings of the Congress on Evolutionary Computing 2004*, pages 1916–1923, Portland, Oregon, 2004.

[27] M. Reimann, K. Dörner, and R. Hartl. D-Ants: Savings based ants divide and conquer the vehicle routing problem. *Computers and Operations Research*, 31:563–591, 2004.

[28] T. Stützle and M. Dorigo. ACO algorithms for the quadratic assignment problem. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 33–50. McGraw-Hill, London, 1999.

[29] T. Stützle and M. Dorigo. ACO algorithms for the traveling salesman problem. In K. Miettinen, M. Makela, P. Neittaanmaki, and J. Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science*. Wiley, 1999.

[30] S. van der Zwaan and C. Marques. Ant colony optimisation for job shop scheduling. In *Third Workshop on Genetic Algorithms and Artificial Life (GAAL 99)*, 1999.

TABLE I

THE PROBLEM INSTANCES USED IN THIS STUDY. NOTE THAT THE LINEAR PREPROCESSING MEASURES OR REDUCTION OPERATIONS (CALLED PREP) DESCRIBED BY CHU AND BEASLEY [8] HAVE BEEN APPLIED. THE AIM OF PREP IS TO TRANSFORM THE PROBLEM INTO AN EQUIVALENT, BUT SMALLER SIZED PROBLEMS (IN TERMS OF THE NUMBER OF ROWS AND COLUMNS). SMALL PROBLEMS ARE CONSIDERED AS THOSE THAT HAVE UNDER ONE THOUSAND PREP COLUMNS WHEREAS LARGE HAVE GREATER THAN THIS NUMBER.

		Original		After PREP		Optimal
Size	Name	Rows	Columns	Rows	Columns	Cost
Small	NW08	24	434	21	352	35894
	NW12	27	626	25	451	14118
	NW15	31	647	29	405	67743
	NW20	22	685	22	536	16812
	NW21	25	577	25	421	7408
	NW22	23	619	23	520	6984
	NW23	19	711	18	423	12534
	NW24	19	1366	19	926	6314
	NW25	20	1217	20	844	5960
	NW26	23	771	21	464	6796
	NW27	22	1355	22	817	9933
	NW28	18	1210	18	580	8298
	NW32	19	294	18	251	14877
	NW34	20	889	20	718	10488
	NW37	19	770	19	639	10068
	NW38	23	1220	21	690	5558
	NW39	25	667	25	565	10080
	NW40	19	404	19	336	10809
	NW41	17	197	17	177	11307
	NW42	23	1079	23	795	7656
	NW43	18	1072	17	982	8904
Large	KL01	55	7479	47	5915	1086
	NW07	36	5172	34	3105	5476
	NW09	40	3103	40	2301	67760
	NW11	39	8820	34	6482	116256
	NW18	124	10757	110	8439	340160
	NW19	40	2879	32	2134	10898
	NW29	18	2540	18	2034	4274
	NW31	26	2662	26	1728	8038
	NW33	23	3068	23	2308	6678
	NW35	23	1709	23	1132	7216
	NW36	20	1783	20	1204	7314

TABLE II

THE FIRST TWO COLUMNS GIVE THE RESULTS OF SEARCH WITHOUT AND WITH IS2 RESPECTIVELY WHILE THE THIRD IS OF THE MULTISTART ALGORITHM. "MIN", "MED" AND "MAX" DENOTE MINIMUM, MEDIAN AND MAXIMUM RESPECTIVELY. RESULTS ARE EXPRESSED AS RELATIVE PERCENTAGE DIFFERENCES (RPDS). THIS IS CALCULATED AS $\frac{a-b}{b} \times 100$ WHERE a IS THE BEST COST PRODUCED BY A RUN AND b IS THE OPTIMAL COST.

	Without IS2			With IS2			Multistart		
	min	med	max	min	med	max	min	med	max
NW08	1.1	1.59	1.91	0.48	1.59	1.91	23.83	23.88	25.28
NW12	0	1.03	1.66	0.47	1.89	3.27	13.29	16.93	19.21
NW15	0	0	0	0	0	0	0	0	0
NW20	0	0	0.91	0	0	0.91	1.46	5.15	8.92
NW21	0	0	0	0	0	0	8.02	11.66	27.35
NW22	0	1.09	2.49	0	0.54	1.09	27.21	35.08	49.46
NW23	0	0	1.01	0	0	1.01	2.09	10.2	19.74
NW24	0.16	0.73	3.8	0	0	0	23.34	27.94	28.03
NW25	4.8	4.8	8.22	4.8	4.8	4.8	10.6	49.31	63.52
NW26	0	0	0	0	0	0	4.33	18.63	35.23
NW27	0	0	0	0	0	0	9.27	13.79	35.03
NW28	0	0	0	0	0	0	0	6.58	15.18
NW32	1.67	4.42	7.02	0	0	1.63	6.82	31.49	42.51
NW34	0	2.03	2.15	0	2.15	2.15	7.64	18.98	34.95
NW37	1.64	3.07	6.79	1.64	3.07	3.07	1.64	3.07	3.07
NW38	0	0	0.61	0	0	0.61	2.77	4.32	14.32
NW39	4.26	6.73	7.86	4.29	6.37	6.73	6.37	32.75	47.53
NW40	0	0.36	0.36	0	0.36	0.36	0.36	3.72	6.66
NW41	0	0.54	11.99	0	1.09	1.09	1.09	10.32	10.32
NW42	0	0	0	0	0	0	0	3.64	15.52
NW43	2.63	6.1	7.35	1.21	1.21	6.27	13.3	34.95	41.8