Verification of optimization algorithms: a case study of a quadratic assignment problem solver

Tsong Yueh Chen[*], Huimin Lin[†], Robert Merkel & Daoming Wang

Abstract

It is often difficult to verify the solutions of computationally intensive mathematical optimization problems. Metamorphic testing is a technique to verify software test output even when a complete testing oracle is not present. We apply metamorphic testing to a classic optimization problem, the quadratic assignment problem (QAP). A number of metamorphic relations for the QAP are described in detail, and their effectiveness in "killing" mutated versions of an exact QAP solver is compared. We show that metamorphic testing can be effectively applied to the QAP in the absence of an oracle, and discuss the implications for the testing of solvers for other hard optimization problems.

1 Introduction

In software testing, an "oracle" is a means by which it is determined whether the output of a test case meets the software's specification. In some cases, a convenient oracle is readily available; however, there are many situations where the oracle's evaluation is manually conducted, a slow and error-prone process. For some applications - a notable example is scientific simulation - even this may not be possible.

Metamorphic testing is a testing technique designed to allow the checking of test output where there is no oracle, or it is too expensive to verify against the oracle [1]. In this paper, we investigate the use of metamorphic testing on a class of problems where oracles are difficult to find - mathematical optimization problems. A classic example of this class, the quadratic assignment problem, serves as the subject of our case study.

1.1 The Quadratic Assignment Problem

Mathematical optimization problems, in general, involve finding the maxima or minima of functions, usually taking into account constraints of the function parameters. Many practically important classes of optimization problems are NP-hard[1], or even harder. In practice, given that we need an actual solution rather than simply knowing the existence of one, this means that verifying that a solution is indeed optimal can be very difficult.

The Quadratic Assignment Problem (QAP) is a classic example of this kind of difficult operations research problem. Informally, the problem can be viewed as "the assignment of facilities to locations" [2]. For example, consider a company which intends to build k factories denoted as $f_1, f_2, \ldots, f_k$. The factories can be constructed in k distinct locations $l_1, l_2, \ldots, l_k$. The output of some factories is used as the input to others. The amount of goods that will move between factories is quantified as the "weight". This weight can be represented as a $k \times k$ matrix w, where $w(i, j)$ represents the weight of the items moving from factory f_i to factory f_j.

The relative expense of moving items around between locations can be represented in another $k \times k$ matrix c, where $c(i, j)$ represents the cost of moving a unit of weight from location l_i to location l_j. In many real-world applications, the cost corresponds to the distance between two locations. Therefore, it is common for $c(i, j)$ and $c(j, i)$ to be the same for all i and j, and thus for the matrix to be symmetric, but this is not compulsory according to the formal definition of the problem. The quadratic assignment problem is to assign factories to locations such that the total transport cost is minimised. More formally [3], the goal is to find a bijection $g : \{f_1 \ldots f_k\} \rightarrow \{l_1 \ldots l_k\}$ such that the following summation is minimised:

$$\sum_{i=1}^{k} \sum_{j=1}^{k} w(i, j)\, c(g(i), g(j)) \qquad (1)$$

Like many other optimization problems, the solution g may not be unique (as will be illustrated in section 2.1).

Given that the problem is NP-hard, and the obvious practical applications, there has been a great deal of research to find efficient approximation algorithms for the QAP (see [4] for a survey).

In this paper, however, our interest in the QAP was in demonstrating the detection of faults in an exact QAP solver.

[*]T.Y. Chen and Robert Merkel (corresponding author) are at at Swinburne University of Technology, John St. Hawthorn 3122, Email: {tchen,rmerkel}@swin.edu.au

[†]Huimin Lin and Daoming Wang are at the Institute of Software, Chinese Academy of Sciences, Beijing, China, Email: {lhm,dmwang}@ios.ac.cn

[1] in their decision problem form

Lau [5] provides such a solver, implemented in Java. Primarily intended for educational purposes, this solver meets our needs as a relatively straightforward implementation that could be easily incorporated into a test harness. Lau's code base contains a large number of other methods that implement other optimization algorithms, that are not executed when using the QAP solver. To avoid irrelevant and time-consuming processing of this code in our experimental setup, methods unrelated to the QAP (and thus never executed) were removed from the source code.

1.2 Metamorphic Testing

Metamorphic testing [1] is a method for verifying test output in the absence of an oracle. It is based on the idea of taking pre-existing *source* test cases T, and systematically generating a set of *follow-up* test cases T' based on T. While it may not be known directly whether the program output on individual members of T or T' is correct, T' is constructed in such a way so that certain relations between T, T', and their corresponding outputs, must hold if the software is functioning according to its specification.

To take a simple example, consider that the software under test is designed to compute the sine of an angle θ. Elementary trigonometry tells us that $\sin(-\theta) = -sin(\theta) \forall \theta$. We can use this property as a *metamorphic relation* to derive follow-up test cases and relationships between the outputs for the original and follow-up test cases. Therefore, if we have executed a test case θ_i with output s_{θ_i}, we can derive a follow-up test $-\theta_i$. The output from the execution of the follow-up test case $-\theta_i$ must be equal to $-s_{\theta_i}$ for the metamorphic relationship to hold. If it does not, then a failure in the software under test is revealed.

This metamorphic relation is a simple one-to-one correspondence between a single source test case and a single follow-up test case, but metamorphic relations can be defined between multiple tests. For instance, consider the trigonometric tangent function. The tangent addition formula states that for angles θ and ϕ, $\tan(\theta + \phi) = \frac{\tan\theta + \tan\phi}{1 - \tan\theta\tan\phi}$. To use this metamorphic relation with two source test cases, θ and ϕ, a follow-up test case $(\theta + \phi)$ is constructed and executed, and the metamorphic relation is then checked. Furthermore, while these examples have made use of an equality relationship, this is not obligatory; other relationships can be defined, such as inequalities, greater-than or less-than relationships, or non-numeric relationships such as subsets.

Metamorphic testing has been examined in a number of contexts, including COTS component testing [6], context-sensitive middleware [7], and programs involving symmetries [8]. It has been shown to provide effective testing in the absence of an oracle.

In this paper, we seek to demonstrate the application of metamorphic testing for a solver of a well-known optimization problem.

2 Some Metamorphic Relations for the QAP

For most software under test, there are many valid metamorphic relations that might be used to check software correctness. However, not all of them will be effective in revealing software failures; as a trivial example, if we have a program P that takes some input I (assuming there is no internal state), and returns P(I), clearly P(I) = P(I) for all I. However, this will probably reveal only a very few failures.

We devised a number of metamorphic relations that we hoped would be effective for evaluating the correctness of a QAP solver. The three types of metamorphic relations (six relations in total), were identified quickly, over a few hours. None of the authors had any previous experience in the area of QAP solvers. All the metamorphic relations we constructed, were evaluated experimentally. We did not cherry-pick a few satisfactory metamorphic relations from a much larger, mostly unsatisfactory bunch!

2.1 Relabelling

The labelling of factories and locations in the QAP does not affect the nature of the solutions, only their names. For instance, assume that we have a QAP of size 3 with an optimal assignment of f_1 to l_2 f_2 to l_1 and f_3 to l_3. If we relabel factories f_1, f_2, and f_3 as $f_2\prime$, $f_3\prime$, and $f_1\prime$ respectively, without changing the weights between the relabelled factories, we know that assigning factory $f_2\prime$ to l_1, $f_3\prime$ to l_2, and $f_1\prime$ to l_3 will be optimal for the relabelled QAP.

However, for any given QAP, there may be more than one optimal solution - for a trivial example, consider a QAP where all weights are zero and hence every possible assignment is an optimal solution. However, the QAP solver does not guarantee to return any particular optimal assignment. Therefore, we cannot assume that the optimal solution found to a relabelled problem will be the corresponding relabelling of the optimal assignment found for the original problem. However, all optimal solutions by definition have the same total cost; therefore, any optimal solution to the relabelled problem will have the same total cost as the optimal solution to the original problem.

Consider the QAP denoted by Q_a with weight and distance matrices defined as follows:

$$w_a = \begin{matrix} 0 & 5 & 8 \\ 0 & 0 & 0 \\ 0 & 4 & 0 \end{matrix}, c_a = \begin{matrix} 0 & 7 & 2 \\ 7 & 0 & 10 \\ 2 & 10 & 0 \end{matrix} \quad (2)$$

If we relabel the factories such that $f_1\prime = f_3$, $f_2\prime = f_1$, and $f_3\prime = f_2$, we can obtain a weight matrix $w_a\prime$:

$$w_a\prime = \begin{matrix} 0 & 0 & 4 \\ 8 & 0 & 5 \\ 0 & 0 & 0 \end{matrix} \quad (3)$$

Similarly, if we relabel the locations $l_1\prime = l_2, l_2\prime = l_3, l_3\prime = l_1$, we get a distance matrix $c_a\prime$:

$$c_a\prime = \begin{matrix} 0 & 10 & 7 \\ 10 & 0 & 2 \\ 7 & 2 & 0 \end{matrix} \quad (4)$$

The QAP instance $Q_a\prime$, with weight matrix $w_a\prime$ and distance matrix c_a has an optimal solution with exactly the same cost as Q_a. The QAP instance $Q_a\prime\prime$, with weight matrix w_a and distance matrix $c_a\prime$ will also have an optimal solution with the same cost, as would (though we did not use this) $Q_a\prime\prime\prime$ with $w_a\prime$ and $c_a\prime$.

We implemented this metamorphic relation by developing a tool that randomly permutes the weight of any given problem instance, and then separately permutes the distance of the problem instance, resulting in two follow-up problem instances. The QAP solver is then used to solve both the source problem instance Q_a and the two follow-up problem instances $Q_a\prime$ and $Q_a\prime\prime$. The total cost of all three must be equal.

2.2 Adding a new factory

There is no straightforward way to predict the effect on the optimal solution after an arbitrary addition of a factory and location to an existing QAP. However, if it is possible to ensure that any optimal solution must have the new factory being placed in the new location, calculating the cost of the optimal solution to the expanded problem, is then quite straightforward.

One way to ensure this is to make the new location a very long distance M from all the existing locations (much greater than distances between existing factories), and assign zero weights between the new factory and all existing factories. As discussed earlier, there is the possibility that there are multiple optimal solutions to the original QAP, and therefore there are multiple optimal solutions for the expanded QAP. If this is the case, we cannot be sure that the optimal solution found by the solver for the expanded QAP will simply be the optimal solution found for the original one, with the new factory assigned to the new location. Regardless, any optimal solution must have the new factory in the new location, and hence the total cost must not change.

One potential flaw in such a scheme is that there are still certain obvious types of faults that such a relation will not detect. For instance, consider a bug in the cost calculator such that it always finds a total cost of 0, regardless of the assignment of factories to locations. Therefore, a modified version of this metamorphic relation was sought. Instead of zero weights between the new factory and all existing factories, one existing factory f_i is chosen randomly, and a weight smaller than all existing non-zero weights is assigned between the new factory f_n and f_i, with weights between the new and existing factories other than f_i set to 0. In this case, the new factory will still be assigned to the new location, but the cost of an optimal solution will be increased. If the distance between f_n and f_i is d, and the weight is ϵ, the total cost will increase by $d \cdot \epsilon$. For this relationship to hold, there must be no "isolated" factories (that is, factories for whom all weights are 0) in the existing problem.

For instance, consider again Q_a from the previous section. The distance matrix for the follow-up test cases, $Q_a\prime$ and $Q_a\prime\prime$, $c_a\prime$, is defined as follows:

$$c_a\prime = \begin{matrix} 0 & 7 & 2 & M \\ 7 & 0 & 10 & M \\ 2 & 10 & 0 & M \\ M & M & M & 0 \end{matrix} \quad (5)$$

We can define the corresponding weight matrices, $w_a\prime$ for the zero-weight case, and $w_a\prime\prime$ for the non-zero weight case:

$$w_a\prime = \begin{matrix} 0 & 5 & 8 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{matrix}, w_a\prime\prime = \begin{matrix} 0 & 5 & 8 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{matrix} \quad (6)$$

The QAP problem $Q_a\prime$ made up of $w_a\prime$ and $c_a\prime$ will have the same total cost as Q_a. The QAP problem $Q_a\prime\prime$ defined by $w_a\prime\prime$ and $c_a\prime$ will have a total cost $M > Q_a$.

Both versions of this metamorphic relation, linking Q_a with $Q_a\prime$, and Q_a with $Q_a\prime\prime$, were implemented as described.

2.3 Merging

The previous metamorphic relations are straightforward one-to-one correspondences. As explained in section 1.2, it is possible to define metamorphic relations for the QAP involving multiple test cases. Here, we define a metamorphic relation based on the merger of two existing problem instances.

Such a merged problem Q_m can be constructed in a fairly straightforward manner from two smaller problems Q_a and Q_b. The factories for Q_m are the union of the factories from Q_a and Q_b. If $l_{a1}, l_{a2} \ldots, l_{aj}$ designate the locations in Q_a, let $l_{ma1}, l_{ma2}, \ldots, l_{maj}$ designate all the locations in Q_m taken from Q_a. For all possible pairs of $l_{ma\alpha}$ and $l_{ma\beta}$, let the distance between them be the same as the distance between the corresponding locations in Q_a, $c_{a\alpha a\beta}$. Similarly, for $l_{mb1}, l_{mb2}, \ldots l_{mbk}$, the distances between the pairs should be the same as the corresponding locations in Q_b. The weights for the merged problem should be constructed in the same manner.

The key insight enabling the optimal solution to the merged problem to be predicted is the distances and the weights between pairs, where one member of the pair is from Q_a and one from Q_b. This can be achieved by making the distance between the pairs of locations of this type some very large value M - much larger than the distances between all pairs where both are from Q_a, or both from Q_b. The weights between pairs of factories where one is from Q_a and one from Q_b are zero. To minimise the total cost, any optimal solution will ensure that the very long distances correspond with the

zero weights, and thus ensures that the factories corresponding to Q_a are in one "cluster" of locations, with the factories corresponding to Q_b at the other cluster.

If the "clusters" of factories end up at the locations corresponding to their corresponding problems, the total cost of the merged problem will be the sum of the two original problems, thus giving us a metamorphic relation. But we must guarantee that the factories are located in this manner. The simplest way to guarantee this is to ensure that the original problems are of different sizes - any attempt by the solver to place the factories in the "wrong" cluster will result in at least one very large cost penalty, and will thus not be an optimal solution.

A second metamorphic relation can be created by a technique similar to that for adding nodes, by ensuring that there is one small weight between a pair of factories, one in each cluster. One factory originally from Q_a, f_{maj}, and a factory from Q_b, f_{mbk} are randomly selected, and rather than a zero weight between them a very small weight ϵ, smaller than all existing weights, is added between them, such that $w(aj, bk) = \epsilon$. The total cost of the optimal solution to the merged problem is then the sum of the total costs of the original problems and $\epsilon \times \Delta$.

To illustrate this, we consider Q_a from the previous examples, and Q_b defined as follows:

$$w_b = \begin{matrix} 0 & 20 & 0 & 0 \\ 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 30 \\ 5 & 0 & 0 & 0 \end{matrix}, c_b = \begin{matrix} 0 & 25 & 15 & 22 \\ 25 & 0 & 12 & 44 \\ 15 & 12 & 0 & 68 \\ 22 & 44 & 68 & 0 \end{matrix} \quad (7)$$

We can now define follow-up test cases for the two variants of the metamorphic relations, Q_m for the zero-weight case and $Q_m\prime$ for the weight case. In both cases, the distance matrix c_m is defined as follows:

$$c_m = \begin{matrix} 0 & 7 & 2 & M & M & M & M \\ 7 & 0 & 10 & M & M & M & M \\ 2 & 10 & 0 & M & M & M & M \\ M & M & M & 0 & 25 & 15 & 22 \\ M & M & M & 25 & 0 & 12 & 44 \\ M & M & M & 15 & 12 & 0 & 68 \\ M & M & M & 22 & 44 & 68 & 0 \end{matrix} \quad (8)$$

The weight matrix for the zero-weight case, w_m and for the weight case, $w_m\prime$, are as follows:

$$w_m = \begin{matrix} 0 & 5 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 30 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 \end{matrix} \quad (9)$$

$$w_m\prime = \begin{matrix} 0 & 5 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 30 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 \end{matrix} \quad (10)$$

Q_m has a total cost equal to the sums of the costs of Q_a and Q_b, and $Q_m\prime$ has a total cost equal to $Q_m + M$.

Both of these metamorphic relations were implemented as described, with a small program written to transform two existing problem instances into a single merged one, with zero or non-zero weight between a pair of members of different original problems.

3 Experimental evaluation

As we did not have access to a set of faulty QAP solvers, we instead used the technique of mutation testing to assess the effectiveness of our metamorphic relations.

In our experiment, we compared the proportion of mutants killed by the various metamorphic relations we have devised, to the proportion killed by checking against a pre-computed correct solution - that is, with a test oracle.

3.1 Mutations - Jumble

Mutation testing is a well-established technique in which faults are deliberately inserted into software to form "mutants" to determine whether a set of test cases can detect them. It can be used for a number of different purposes, including making an assessment of the quality of test cases in terms of mutation scores, denoting the proportion of mutants killed. Jumble [9] is a mutation testing tool designed for assessing the quality of unit tests written in the JUnit unit testing framework for Java software. Jumble supports the evaluation of a test case by the following procedure:

- identifying all locations in a Java class where its supported mutation operations can be performed, producing a set of mutants.
- For each mutant, executing the specified JUnit test, and recording whether the test detects a failure.
- Reporting the proportion of mutants detected.

Jumble allows the user to specify a number of different mutation methods, but by default two types of mutations are enabled. The first involves *conditionals*, where a condition in an `if`, `while`, `do`, and `for` statements, is replaced by its negation. The second default mutation type is for arithmetic operators, where arithmetic operator is replaced by another according to a predefined table - for instance, the "+" operator is replaced with "-". Only the default mutation types were used in our experiments.

3.2 Sample cases as source test data

For our experiment, we required some QAP instances that could serve as the basis for a suitable source test set. Rather than generating our own QAP instances, we started with some sample instances from the QAPLIB library of quadratic assignment problem instances, which is widely used in the QAP research community as a testbed to determine the performance of new exact and approximate solvers. [10].

The nature of our experiment meant that even the smallest problems in QAPLIB were too slow to be practical. More tractable QAP instances were created by selecting a contiguous, randomly selected subset of the factories and locations in a problem instance in QAPLIB. The use of contiguous subsets, rather than just selecting random factories and locations, was on the basis that many of the problem instances in QAPLIB seemed to have the shortest distances to numerically contiguous factories, and many had quite sparse weight matrices, with the vast majority of non-zero weights to near neighbours. By this measure, we hoped to retain the "flavour" of the original problems.

Even so, the limited computing resources available made it impractical to apply Jumble mutations to many different test cases. Only a few test cases were therefore tried with each metamorphic relation, but using a large number of mutants.

In this study, for most problems only three problem instances were used as the source test cases (referred to as t_1, t_2 and t_3 in Table 1). 768 mutants were tried with each test case. For the "merge" relations, two source test cases are required, and that the two source test cases must be of different sizes. For these relations, three additional QAP instances, slightly smaller than t_1, t_2, and t_3, were created by the same shrinking procedure. These new QAP instances were paired with the corresponding t_i to provide the necessary second source test for the merge relation.

3.3 Testing procedure

For each selected QAP instance and each metamorphic relation (or pairs of problem instances in the case of the "merge" metamorphic relation), a JUnit test case was created. The JUnit test executes the source test case, the follow-up test case, and checks whether the specified metamorphic relation holds, and succeeds or fails accordingly. Jumble was then used to repeatedly apply the JUnit test to all the different mutated versions of the QAP solver. The proportion of mutants that were detected was recorded.

4 Results

Table 1 shows the proportion of mutants killed by running the three test cases and the various metamorphic relations. For the merge relations, as noted previously, two source test cases of different sizes were created to apply this relation. For these relations, therefore, the column t_i should be read as "t_i and another, smaller, source test case".

Metamorphic Relation	Mutants killed (%)		
	t_1	t_2	t_3
Oracle	71	73	75
Relabelling	13	13	14
Add (zero weight)	25	25	25
Add (weight)	75	74	74
Merge (zero weight)	21	22	24
Merge (weight)	77	74	76

Table 1. Proportion of mutants killed by metamorphic relations

It is not surprising that the oracle case did not kill all mutants, simply because a single test case is not necessarily a failure-revealing input for any specific mutant. What may be slightly more surprising to the reader is that the performance of two of the metamorphic relations, "Add (weight)" and "Merge (weight)" is higher than that of the oracle. This can also be explained in terms of the number of test cases executed. Effectively, "Add (weight)" executes two test cases for each source test case, and "Merge (weight)" executes three test cases for each pair of source test cases, while testing using the oracle only involves the execution of one test case. An error revealed by any of the source or follow-up test cases may violate the metamorphic relation and hence reveal the failure.

The results show dramatic differences in the effectiveness of different metamorphic relations - the "Relabelling" relation detected around 13% of mutants, whereas the "Merge (weight)" relation detected around 75% of mutants. This shows that selection of metamorphic relations is vital for the effective application of metamorphic testing

Furthermore, we observe that a seemingly minor difference in a metamorphic relation - the difference between "weight" and "zero weight" for both the "Add" and "Merge" metamorphic relations - results in dramatic differences in failure-revealing effectiveness. It is interesting to consider why this small difference contributed to such contrasting failure-revealing effectiveness. There are a number of possible explanations. One explanation, discussed in [11], is that the "weight" versions of the metamorphic relations led the program execution patterns in the follow-up test cases and the source test cases to be more divergent, giving more chance for a bug to be exposed. If so, this indicates that testers should choose relations that encourage divergent execution patterns.

There are, of course, a wide variety of other metamorphic relations that could be tried for this problem. One property of the QAP, (and many other optimization problems), is that a problem instance may have multiple optimal solutions. This makes it difficult to devise metamorphic relations based on the mapping of factories to locations, rather than the total cost. However, making metamorphic relations

conditional may make this easier. For instance, if problem instance Q has assignments a and total cost t, we create a follow-up problem instance $Q\prime$ identical to Q except that one randomly selected weight value $w_m\prime(i,j) = w_m(i,j) + \epsilon$, where ϵ is a positive constant. We can unconditionally say that $t \leq t\prime \leq t + \epsilon * c_m(i,j)$. However, two other conditional metamorphic relations exist. If the assignment for the optimal solution for $Q\prime$ is still a, the total cost c must increase by precisely $\epsilon \cdot c_m(i,j)$. Furthermore, if the cost increase is strictly smaller than $\epsilon \cdot c_m(i,j)$, $a\prime$ the assignment for the optimal solution to $Q\prime$, must be different to a.

In this experiment, we have used an exact solver that guarantees to find a globally optimal solution to the QAP. In practice, heuristic approximation algorithms are typically used; under such conditions, the metamorphic relations above would not be suitable. Some work has already been conducted in applying metamorphic testing to heuristic algorithms [12].

The dramatic difference in effectiveness of different metamorphic relations suggests it would be desirable to have some method to screen a set of metamorphic relations to find an effective subset. This study suggests that mutation analysis may prove useful for this purpose.

5 Conclusion

In this study, we have shown that metamorphic testing can be effectively used to find faults in an exact QAP solver.

To our pleasant surprise, we found that it was relatively easy, even for non-experts in the problem domain, to come up with effective metamorphic relations. Even given the relatively small scope of this study, we feel that this is a strong indication of the likely practical utility of this approach.

Our results suggest that the metamorphic testing approach is very likely to be useful for ensuring the quality of other solvers of hard optimization problems, given the difficulty of verifying the correctness of their solutions.

The simplicity of metamorphic testing suggests that relatively inexperienced practitioners, or even end users, can perform useful verification using this technique. The tester can specifically target the properties that are important for their own use of the software. Furthermore, the technique is very straightforward to describe, and requires only a small amount of domain knowledge to apply.

Acknowledgements

This project was supported by the Natural Science Foundation of China (Grant No.60421001) and Australian Research Council (ARC LX0776490).

References

[1] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: a new approach for generating next test cases," Hong Kong University of Science and Technology, Tech. Rep. HKUST-CS98-01, 1998.

[2] J. W. Gavett and N. V. Plyter, "The optimal assignment of facilities to locations by branch and bound," *Operations Research*, vol. 14, no. 2, pp. 210–232, March 1966.

[3] Wikipedia, "Quadratic assignment problem — wikipedia, the free encyclopedia," 2007, [Online; accessed 12-October-2007]. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Quadratic_assignment_problem&oldid=139133382

[4] P. Pardalos, F. Rendl, and H. Wolkowicz, "The quadratic assignment problem: a survey and recent developments," in *Quadratic assignment and related problems (New Brunswick, NJ, 1993)*, P. Pardalos and H. Wolkowicz, Eds. Providence, RI: Amer. Math. Soc., 1994, pp. 1–42.

[5] H. T. Lau, *A Java Library of Graph Algorithms and Optimization*. Chapman and Hall/CRC, 2007.

[6] S. Beydeda, "Self-metamorphic-testing components," in *Computer Software and Applications Conference, 2006. COMPSAC '06. 30th Annual International*, Chicago, Illinois, USA, September 2006, pp. 265–272.

[7] W. K. Chan, T. Y. Chen, H. Lu, T. H. Tse, and S. S. Yau, "Integration testing of context-sensitive middleware-based applications: a metamorphic approach," *International Journal of Software Engineering and Knowledge Engineering*, vol. 6, no. 5, pp. 677–703, 2006.

[8] A. Gotlieb and B. Botella, "Automated metamorphic testing," in *Proceedings of the 27th International Computer Software and Applications Conference (COMPSAC 2003)*, 2003, pp. 34–40.

[9] Reel Two, "Jumble website." [Online]. Available: http://jumble.sourceforge.net/

[10] R. E. Burkard, S. E. Karisch, and F. Rendl, "QAPLIB - a quadratic assignment problem library," *Journal of Global Optimization*, vol. 10, no. 4, pp. 391–403, 1997.

[11] T. Y. Chen, D. Huang, T. H. Tse, and Z. Q. Zhou, "Case studies on the selection of useful relations in metamorphic testing," in *Proceedings of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC 2004)*, 2004, pp. 569–583, Madrid, Spain.

[12] A. C. Barus, T. Y. Chen, D. D. Grant, F.-C. Kuo, and M. F. Lau, "Testing of heuristic methods: a case study of a greedy algorithm," in preparation.