



Local Search Heuristics for Multi-Index Assignment Problems with Decomposable Costs

H. -J. Bandelt; A. Maas; F. C. R. Spieksma

The Journal of the Operational Research Society, Vol. 55, No. 7, Part Special Issue: Local Search. (Jul., 2004), pp. 694-704.

Stable URL:

<http://links.jstor.org/sici?sici=0160-5682%28200407%2955%3A7%3C694%3ALSHFMA%3E2.0.CO%3B2-P>

The Journal of the Operational Research Society is currently published by Operational Research Society.

Your use of the JSTOR archive indicates your acceptance of JSTOR's Terms and Conditions of Use, available at <http://www.jstor.org/about/terms.html>. JSTOR's Terms and Conditions of Use provides, in part, that unless you have obtained prior permission, you may not download an entire issue of a journal or multiple copies of articles, and you may use content in the JSTOR archive only for your personal, non-commercial use.

Please contact the publisher regarding any further use of this work. Publisher contact information may be obtained at <http://www.jstor.org/journals/ors.html>.

Each copy of any part of a JSTOR transmission must contain the same copyright notice that appears on the screen or printed page of such transmission.

The JSTOR Archive is a trusted digital repository providing for long-term preservation and access to leading academic journals and scholarly literature from around the world. The Archive is supported by libraries, scholarly societies, publishers, and foundations. It is an initiative of JSTOR, a not-for-profit organization with a mission to help the scholarly community take advantage of advances in technology. For more information regarding JSTOR, please contact support@jstor.org.



Local search heuristics for multi-index assignment problems with decomposable costs

H-J Bandelt¹, A Maas² and FCR Spieksma^{*3}

¹Universität Hamburg, Hamburg, Germany; ²Maastricht University, The Netherlands; and ³Katholieke Universiteit Leuven, Leuven, Belgium

The multi-index assignment problem (MIAP) with decomposable costs is a natural generalization of the well-known assignment problem. Applications of the MIAP arise, for instance, in the field of multi-target multi-sensor tracking. We describe an (exponentially sized) neighbourhood for a solution of the MIAP with decomposable costs, and show that one can find a best solution in this neighbourhood in polynomial time. Based on this neighbourhood, we propose a local search algorithm. We empirically test the performance of published constructive heuristics and the local search algorithm on random instances; a straightforward iterated local search algorithm is also tested. Finally, we compute lower bounds to our problem, which enable us to assess the quality of the solutions found.

Journal of the Operational Research Society (2004) 55, 694–704. doi:10.1057/palgrave.jors.2601723

Keywords: heuristics; local search; computational analysis; multi-index assignment

Introduction

In the well-known assignment problem, one is given two (disjoint) n -sets I and J , and a cost c_{ij} for each pair $(i, j) \in I \times J$. The problem is to select n pairs such that every element of $I \cup J$ occurs once in a selected pair, while the sum of the costs of the selected pairs is minimal.

This paper addresses a generalization of this two-dimensional assignment problem that involves $k \geq 3$ (instead of two) n -sets. Different types of generalizations are possible, for example, when $k=4$, and given a cost for each quadruple, one could ask for

- (1) n quadruples such that each element of every n -set occurs once in a quadruple, or
- (2) n^2 quadruples such that each pair of elements occurs once in a quadruple, or
- (3) n^3 quadruples such that each triple of elements occurs once in a quadruple,

while minimizing in all cases the sum of the costs of the selected quadruples. We focus here on the generalization described under (1); thus, we consider the problem of finding n (different) k -tuples such that each element occurs once in a k -tuple. This variant is called the *axial* k -index assignment problem. The goal of this paper is, for the special case of decomposable costs,

- to assess experimentally the quality of two types of heuristics proposed in Bandelt *et al.*¹

- to describe an (exponentially sized) neighbourhood, and to show that a best solution within this neighbourhood can be found in polynomial time,
- to experiment with local search heuristics based on this neighbourhood,
- to describe a Lagrangian relaxation scheme that yields lower bounds for the optimum value of instances of our problem.

The paper is organized as follows. First, we give a precise description of the problem and we mention applications and related literature. Next, we propose the neighbourhood, and describe two local search algorithms based on it. Then, we describe the set-up of our computational experiments, and the performance of the algorithms on the instances generated. Finally, we deal with lower bounds based on Lagrangian relaxation, and state the conclusions.

Problem description

A more formal description of our problem is as follows. Given an n -set A ($n \geq 2$) and an index set $K = \{1, \dots, k\}$ with $k \geq 3$ elements (sometimes referred to as the ‘colours’), we denote by

$$A_r = \{r\} \times A \quad \text{for } 1 \leq r \leq k$$

k disjoint copies of A . Then the Cartesian power A^K constitutes the set of all transversals (‘clusters’) of the sets A_1, A_2, \dots, A_k , that is, for each element $a = (a(1), \dots, a(k))$ of A^K exactly one element, viz $(r, a(r))$, of A_r ($1 \leq r \leq k$) is captured. To each cluster $a \in A^K$, a cost c_a is associated. The problem is now to find n clusters such that each element of $\cup_{r \in K} A_r$

*Correspondence: FCR Spieksma, Faculty of Economic and Applied Economic Sciences, Katholieke Universiteit Leuven, Naamsestraat 69, Leuven, B-3000, Belgium.

E-mail: frits.spieksma@econ.kuleuven.ac.be

occurs once in a cluster, while the sum of the costs of the selected clusters is minimal. This can be formulated as an integer programming problem using binary variables x_a that equal 1 if and only if cluster $a \in A^K$ is selected:

$$(MIAP) \quad \text{minimize} \quad c(x) = \sum_{a \in A^K} c_a x_a \quad (1)$$

$$\text{such that} \quad \sum_{a \in A^K : a(r)=i} x_a = 1 \quad \text{for } r \in K, i \in A \quad (2)$$

$$x_a \in \{0, 1\} \quad \text{for } a \in A^K \quad (3)$$

We will assume in this paper that the costs c_a are not completely arbitrary, but depend on given pairwise distances $d(u, v)$ between elements u, v from different colour sets. Specifically, the costs c_a are assumed to satisfy the following equality:

$$c_a = \sum_{r,s \in K, r < s} d((r, a(r)), (s, a(s))) \quad \text{for all } a \in A^K \quad (4)$$

Thus, the cost of a cluster equals the sum of the $\binom{k}{2}$ distances involved. Our motivation for considering this specific type of cost function is that in many applications (see later) the cost coefficients exhibit a certain structure that conforms to (4).

To illustrate the problem consider the following small instance. Take $K = A = \{1, 2, 3, 4\}$. For $r \neq s$ from K and for i, j from A , let the distance $d((r, i), (s, j))$ equal 2 if $i, j > 1$ and $(r + j) \text{ MOD } 4 \neq (s + i) \text{ MOD } 4$, equal 1 if exactly one of i, j is 1, and equal 0 otherwise. Thus, the distances can be read off from the multi-labelled 4-star of Figure 1. Then the solution comprising the four clusters $(1, 1, 1, 1), (2, 3, 4, 4), (3, 4, 3, 2)$, and $(4, 2, 2, 3)$ has costs $0 + 6 + 6 + 6 = 18$, whereas the solution $(1, 2, 3, 4), (2, 3, 4, 1), (3, 4, 1, 2)$, and $(4, 1, 2, 3)$ is optimal with costs $3 + 3 + 3 + 3 = 12$. Notice that both solutions contain the optimal binary assignments between A_1 and each A_j ($j = 2, 3, 4$), each having cost 2.

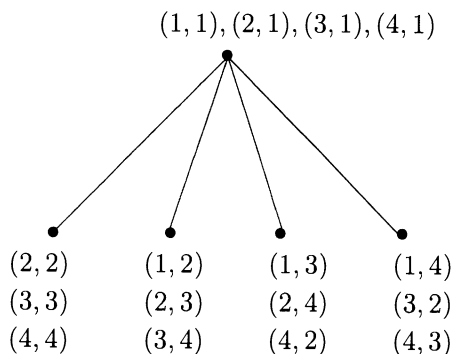


Figure 1 An axial MIAP instance with decomposable (graph) costs.

Literature and applications

The axial multi-index assignment problem (MIAP) has been introduced by Pierskalla² in 1968. For the case that k is fixed, many applications of the k -index assignment problem have been described in the literature. In particular, the case $k = 3$ has received quite some attention: we mention applications in the field of production planning and rostering (see eg Spieksma³ for an overview). Applications for larger values of k seem to be less abundant: we mention Fortin and Tusera⁴ (routing in meshes) and Pusztaszeri *et al*⁵ (tracking elementary particles).

In the case that the value of k is not prespecified, one prominent field of applications of the MIAP is the so-called *data-association* problem in multi-target tracking problems. A short description of this problem is as follows: given is a single radar (or sensor) that surveys its vicinity in a circular fashion. After each revolution of the radar (called a *scan*), a set of measurements induced by objects in the vicinity of the radar has become available. Such a measurement can consist of the location, speed, height, etc of the object involved. Of course, a single object may induce a measurement in different scans. The problem is now to partition the set of all measurements into subsets, such that all measurements from the same subset are induced by a single object. This subset of measurements is called a *track*. One can verify that this problem can be formulated as a multi-index assignment problem by letting A_r be the set of measurements found in scan $r, r = 1, 2, \dots$ (see Morefield,⁶ Pattipati *et al*⁷ and Poore and Rijavec⁸). It should be pointed out that although the cost coefficients arising in this setting possess a certain structure, they are not decomposable in the sense of (4) (see Robertson⁹). Solution methods based on Lagrangian relaxation for these problems have been described by, among others, Chummun *et al*,¹⁰ Pattipati *et al*⁷ and Poore.¹¹ Storms and Spieksma¹² describe a method based on solving the linear programming relaxation, followed by a rounding procedure. Veenman *et al*¹³ describe a problem similar to the data-association problem, which they call the *motion correspondence* problem. Finally, Robertson⁹ describes a greedy, randomized adaptive search procedure.

A variant of this problem arises when we consider a single scan of multiple radars that survey the same area. Then, again, one is interested in determining which measurements come from the same object. Letting A_r be the measurements of radar r yields a multi-index assignment problem. In this case, however, we argue that a straightforward model of this variant admits decomposable cost coefficients. Let us first consider an (idealized) setting where the position of each of n stationary objects is measured by k radars simultaneously, giving a potential total of nk measurements. Due to measurement errors, atmospheric condition, etc, the location of an object measured by radar r will in general differ from the location measured by another radar s . How to decide which set of measurements, one from each radar, originates

from a same object? It is very natural to take the Euclidean distance between any two measured locations from different radars as the cost of deciding that these two measurements originate from a same object. Next, by choosing as the cost of a k -tuple of measured locations the sum of the $\binom{k}{2}$ distances, we have cost coefficients satisfying (4). Of course, in general, a radar measures more than location only. For instance, speed, direction, elevation may be measured as well. Thus, measuring an object yields a vector of values. Again, however, one can summarize the difference between two vectors in a single number to indicate the difference between the two vectors. In this way, one can capture the costs of deciding that two measurements originate from the same object.

The 3-index assignment problem with decomposable costs is \mathcal{NP} -hard. This follows directly from a result in Spieksma and Woeginger¹⁴ where it is shown that even when the points of $A_1 \cup A_2 \cup A_3$ lie in the plane, the problem of partitioning the point set into triangles from $A_1 \times A_2 \times A_3$ such that the sum of the circumferences of the triangles is minimal \mathcal{NP} -hard. Thus, since it is apparently unlikely that a polynomial time algorithm giving optimal solutions exists, heuristics are the designated way to get hold of good solutions within reasonable computing times.

Bandelt *et al*¹ describe approximation algorithms for special cases of the multi-index assignment problem, including the case where the cost coefficients satisfy (4) (see later). More specifically, they consider the case where the distance function d satisfies a (relaxed) form of the triangle inequality, and prove approximation results under this assumption. Computational results for these algorithms in the cases $k = 3$ and 4 are reported in Bierlein¹⁵ and Crama and Spieksma.¹⁶ A recent polyhedral study of the MIAP can be found in Magos *et al*.¹⁷ Further work on the MIAP can be found in Rüschemdorf¹⁸ and Gilbert and Hofstra.¹⁹ Aiex *et al*²⁰ describe the application of greedy randomized local search procedures to the 3-index assignment problem; we refer to Aarts and Lenstra²¹ and to Ahuja *et al*²² for an overview of local search methods.

Local search

We now propose a neighbourhood of a solution to the MIAP, and we describe two constructive heuristics for the MIAP.

The neighbourhood

Every bipartition of the index set K into a (proper, nonempty) subset L and its complement $K \setminus L$ induces a bipartition of any cluster $a \in A^K$ into the corresponding two restrictions ('partial clusters') $a_L \in A^L$ and $a_{K \setminus L} \in A^{K \setminus L}$, so that a can be identified with the pair $(a_L, a_{K \setminus L})$. Accordingly, every feasible solution x to the MIAP, which consists of n

disjoint clusters $a^1, \dots, a^n \in A^K$, splits into feasible solutions x_L (consisting of a^1_L, \dots, a^n_L) and $x_{K \setminus L}$ (consisting of $a^1_{K \setminus L}, \dots, a^n_{K \setminus L}$) of the MIAP relative to L and $K \setminus L$, respectively. Since the union of any two partial clusters from A^L and $A^{K \setminus L}$ (such as $a^i_L \cup a^j_{K \setminus L}$) yields a cluster from A^K , there are $n!$ ways to recombine the two feasible partial solutions x_L and $x_{K \setminus L}$ into a new feasible solution $y \in A^K$. Thus, the *neighbourhood* of x relative to L ($\emptyset \neq L \subset K$) is defined as the $n!$ -set $N^L(x)$ of all solutions $y = y(\pi)$ that consist of the n disjoint clusters

$$a^1_L \cup a^{\pi(1)}_{K \setminus L}, \dots, a^n_L \cup a^{\pi(n)}_{K \setminus L}$$

where π is any permutation of $\{1, \dots, n\}$.

Since the costs $c(x)$ of a solution x are decomposable, the costs $c_L(x_L)$ are well defined by restricting the summation in (4) over all colours $r < s$ from L . Now define the costs $c_{L, K \setminus L}(x; \pi)$ of the recombination of x_L with $x_{K \setminus L}$ determined by the permutation π of $\{1, \dots, n\}$ as follows:

$$c_{L, K \setminus L}(x; \pi) = \sum_{r \in L, s \in K \setminus L} \sum_{i=1}^n d((r, a^i(r)), (s, a^{\pi(i)}(s))) \quad (5)$$

Then, in particular, with id denoting the identity permutation,

$$c(x) = c_L(x_L) + c_{K \setminus L}(x_{K \setminus L}) + c_{L, K \setminus L}(x; id)$$

Therefore, a best neighbour $y \in N^L(x)$, that is, a best recombination of x_L and $x_{K \setminus L}$, is found by solving an ordinary binary assignment problem between the two n -sets $\{a^1_L, \dots, a^n_L\}$ and $\{a^1_{K \setminus L}, \dots, a^n_{K \setminus L}\}$ with respect to the cost function $\pi \mapsto c_{L, K \setminus L}(x; \pi)$. Other examples of methods that use network flow techniques to search a (large) neighbourhood are described in Ahuja *et al*.²²

Whereas a single neighbourhood can thus be searched for a best solution in polynomial time, it would need exponential time to visit all neighbourhoods $N^L(x)$, $\emptyset \neq L \subset K$. In the local search heuristics proposed here, we therefore cycle only through a limited number of subsets L , namely either all singletons (heuristic LS1) or all doubletons (heuristic LS2), until no improvement has been achieved after a full round of k singletons or $\binom{k}{2}$ doubletons visited.

Algorithm LS1

Input: Distance function d determining c via (4), start solution x^0 .

Iteration ($t \geq 1$): Given x^{t-1} , put $h = 1 + (t-1)$ modulo k and compute a best solution x^t in $N^{(h)}(x^{t-1})$.

Stop: $c(x^t) = c(x^{t-k})$ for $t \geq k$.

Output: Solution x that is locally optimal with respect to $N^L(x)$ for each singleton L .

Algorithm LS2

Input: Distance function d determining c via (4), start solution x^0 , lexicographic ordering $L_1 = \{1, 2\}$, $L_2 = \{1, 3\}, \dots, L_{\binom{k}{2}} = \{k-1, k\}$ of all doubletons.

Iteration ($t \geq 1$): Given x^{t-1} , put $h = 1 + (t-1)$ modulo $\binom{k}{2}$ and compute a best solution x^t in $N^{L_h}(x^{t-1})$.

Stop: $c(x^t) = c(x^{t-\binom{k}{2}})$ for $t \geq \binom{k}{2}$.

Output: Solution x that is locally optimal with respect to $N^L(x)$ for each doubleton L .

Starting solution

In our experiments, we used three ways of getting hold of a starting solution. One way is simply constructing a random solution, the other two ways are proposed in Bandelt *et al*¹ and are described hereunder.

Hub heuristics

The single-hub heuristic first chooses a so-called *hub* colour h , $h \in K$. Then it solves $k-1$ assignment problems between the points of A_h and A_r , $r \in K \setminus h$ (using the costs $d(u, v)$, $u \in A_h$, $v \in A_r$, $r \in K \setminus h$). For instance, both solutions mentioned in the example presented earlier are obtainable by the single-hub heuristic with respect to hub colour $h = 1$.

Algorithm SINGLE-HUB

Input: Distance function d , colour $h \in K$.

Iteration ($r \in K \setminus h$): Compute a solution z_{hr} to the minimum cost assignment problem between A_h and A_r with respect to d .

Output: The unique MIAP solution x that extends all binary assignment solutions z_{hr} , that is $x_{\{h, r\}} = z_{hr}$ for $r \in K \setminus h$.

Algorithm MULTI-HUB

Iteration ($h \in K$): Run SINGLE-HUB for each hub colour $h \in K$.

Output: The best solution of the k solutions found.

Notice that the complexity of MULTI-HUB equals the complexity of solving $O(k^2)$ assignment problems.

Recursive heuristics

An obvious drawback of the hub heuristic is that it does not consider all distances within a cluster when computing a solution. The recursive heuristic avoids this drawback as follows. It specifies a sequence of the k colours, say a permutation σ of $\{1, \dots, k\}$, and then constructs a chain of partial MIAP solutions $y^1, y^2, \dots, y^k = x$, relative to the sets $L_1 = \{\sigma(1)\}$, $L_2 = \{\sigma(1), \sigma(2)\}, \dots, L_k = K$, so that the final MIAP solution x extends the partial solutions in this sequence, viz, $x_{L_r} = y^r$ for $2 \leq r \leq k-1$. The costs of an assignment between the clusters in a partial MIAP solution y^{r-1} and the points of $A_{\sigma(r)}$ is defined analogously as in (5): for each partial cluster and assigned point it sums over the distances between the $r-1$ points of a cluster and the new point from $A_{\sigma(r)}$.

Algorithm SINGLE-RECURSIVE

Input: Distance function d , permutation σ of $\{1, \dots, k\}$.

Iteration ($r = 2, \dots, k$): Compute a minimum cost assignment between the partial MIAP solution y^{r-1} relative to $L_{r-1} = \{\sigma(1), \dots, \sigma(r-1)\}$ and the points of $A_{\sigma(r)}$. Then, via this assignment, y^{r-1} is extended to the partial MIAP solution y^r relative to L_r .

Output: The MIAP solution $x = y^k$ obtained in the last step.

Notice that, in principle, there are $k!$ different permutations, each giving rise to a possibly different solution returned by SINGLE-RECURSIVE. In our implementation, we run the following two algorithms, which generate only k or $\binom{k}{2}$, respectively, many permutations, so that every singleton or doubleton of colours would come first in the processing order exactly once.

Algorithm RECUR1

Iteration ($1 \leq h \leq k$): Run SINGLE-RECURSIVE for a randomly generated permutation σ with the proviso $\sigma(1) = h$.

Output: The best solution of the k solutions found.

Algorithm RECUR2

Iteration ($1 \leq i < j \leq k$): Run SINGLE-RECURSIVE for a randomly generated permutation σ with the proviso $\sigma(1) = i$ and $\sigma(2) = j$.

Output: The best solution of the $\binom{k}{2}$ solutions found.

The complexity of SINGLE-RECURSIVE is equal to the complexity of solving k assignment problems, plus $O(n^2k^2)$ (for updating the costs of the partial clusters formed). This implies that, apart from the complexity caused by solving $O(k^2)$ ($O(k^3)$) assignment problems, there is an $O(n^2k^3)$ ($O(n^2k^4)$) term in the complexity bound of RECUR1 (RECUR2).

Experiments

We have generated in total 80 instances, each with $nk = 128$ points. There are four types of instances. For each type we generated:

- five instances with $k = 4$, $n = 32$,
- five instances with $k = 8$, $n = 16$,
- five instances with $k = 16$, $n = 8$,
- five instances with $k = 32$, $n = 4$.

The points in the instances of type 1 are randomly generated in the plane. More specifically, for each of the 128 points an integral x -coordinate and an integral y -coordinate is randomly drawn from $[0, 999]$. The distance between a pair of points is the Euclidean distance rounded down to the nearest integer. An instance of type 2 is generated such that each distance between a pair of points is randomly drawn from $[0, 9]$. An instance of type 3 is generated such that each distance between a pair of points is randomly drawn from $[0, 4]$. Finally, an instance of type 4 is again a geometric instance of the following kind. We partition the square $(0, 999) \times$

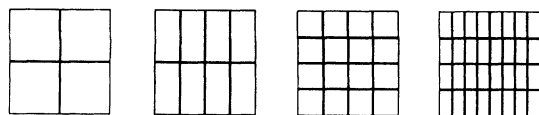


Figure 2 Type 4 instances for, from left to right, $k = 4, 8, 16,$ and 32 .

$(0, 999)$ into k rectangles. Each of the sets $A_r (r = 1, \dots, k)$ resides in a rectangle (see Figure 2 for a picture). We will refer to the instances of types 1 and 4 as Euclidean instances.

To solve the assignment instances in our experiments, we used the code of Jonker and Volgenant²³ (see Dell’Amico and Toth²⁴ for a computational study of various implementations of algorithms that solve the assignment problem). All our algorithms were coded in Java (version 1.3), and we used a Pentium 3 with a 500 MHz processor and 64 Mb internal memory.

Results

Starting heuristics

We first compared the quality of the solutions found by the heuristic MULTI-HUB and the two recursive heuristics RECUR1 and RECUR2. Both MULTI-HUB and RECUR1 need to solve $O(k^2)$ assignment problems; RECUR1 needs an additional updating step, and is at least theoretically

computationally more expensive. RECUR2 needs to solve $O(k^3)$ assignment problems. The outcomes are given in Table 1. The numbers reported are averages over five instances. Columns 2, 5, and 8 (called ‘value’) contain the average values found by MULTI-HUB, RECUR1, and RECUR2, respectively. The percentages given in columns 5 and 8 indicate the amount of improvement over the average solution value found by MULTI-HUB. Columns 3, 6, and 9 (called ‘best’) report how many times (out of the five instances) the corresponding heuristic found a solution with a value that was the best among the three solution values found by MULTI-HUB, RECUR1, and RECUR2. Finally, columns 4, 7, and 10 (called ‘time’) describe the computing times needed in milliseconds.

It turns out that the solutions of the recursive heuristics are better than the solutions found by MULTI-HUB. This is, in particular, true for the case of non-Euclidean instances with small values of k . For each type of instances, the improvement decreases with increasing k . From the Euclidean instances, the improvement of the recursive heuristics over MULTI-HUB is twice as large for type 1 instances compared to type 4 instances. Summarizing, the less structure is present in the instances, the more improvement there is of the recursive heuristics over MULTI-HUB. RECUR2 is slightly better than RECUR1; especially with larger k , it finds more often a better solution than RECUR1, although the improvement in value is not large.

Table 1 Starting heuristics

	Heuristics								
	MULTI-HUB			RECUR1			RECUR2		
	Value	Best	Time	Value	Best	Time	Value	Best	Time
<i>Type 1</i>									
$k = 4$	342.4	0	11.2	315.2 (7.9%)	3	274.2	314.8 (8.1%)	4	244.2
$k = 8$	1063.4	0	6.2	981 (7.7%)	1	132.4	979 (7.9%)	4	175.2
$k = 16$	3009.4	0	3.6	2894.4 (3.8%)	0	85.2	2872.6 (4.5%)	5	214
$k = 32$	7562.6	0	3	7433.6 (1.7%)	2	64.4	7429.8 (1.8%)	4	479.4
<i>Type 2</i>									
$k = 4$	624.2	0	12.8	437.4 (29.9%)	2	262.8	430 (31.1%)	3	262.6
$k = 8$	1973	0	6	1568.2 (20.5%)	1	122.6	1555.4 (21.2%)	4	181.4
$k = 16$	4757.8	0	3	4195.8 (11.8%)	1	80	4173 (12.3%)	4	205.4
$k = 32$	10452.4	0	3.4	9816.2 (6.1%)	0	74.8	9722.4 (7.0%)	5	482.8
<i>Type 3</i>									
$k = 4$	371.6	0	3	283.8 (23.6%)	2	301.8	281 (24.4%)	4	233.8
$k = 8$	1111	0	5.4	922.4 (17.0%)	1	121.6	915.2 (17.6%)	5	181.8
$k = 16$	2634.2	0	3	2370.2 (10.0%)	0	72	2356 (10.6%)	5	121.8
$k = 32$	5710.4	0	2.6	5402.8 (5.4%)	0	114	5377.2 (5.8%)	5	193.8
<i>Type 4</i>									
$k = 4$	1101	0	12.2	1064.4 (3.3%)	2	255.2	1064.4 (3.3%)	4	226.6
$k = 8$	2467.4	0	6	2399.4 (2.8%)	1	114.8	2395.6 (2.9%)	5	136.2
$k = 16$	5174.8	0	3	5096.8 (1.5%)	1	71	5094.6 (1.5%)	5	120.8
$k = 32$	10585	0	2.2	10511.2 (0.7%)	0	51.4	10501.2 (0.8%)	5	192

All computation times are within 0.5s. Specifically, MULTI-HUB is extremely fast. Computation times of RECUR1 and RECUR2 are an order of magnitude larger. Comparing computation times of RECUR1 and RECUR2, it turns out that for $k=32$ RECUR2 can take up to 8 times longer as RECUR1.

Local search

We now investigate the quality of the neighbourhood we propose, as measured by the outcome of the local search algorithms LS1 and LS2. Each of these two algorithms was run using three starting solutions: the solution found by MULTI-HUB, the best solution found by RECUR1 and RECUR2, and a random solution. Thus, we can distinguish six local search algorithms. We report the outcome of these six variants in Tables 2 and 3.

First of all, the local search algorithms LS1 and LS2 almost always improve on the best solution found so far. In particular, the improvement for the non-Euclidean instances is percentage-wise larger than for the Euclidean instances, both for LS1 and for LS2.

Second, let us comment on the choice of the starting solution. At least for the non-Euclidean instances, it seems that a better starting solution yields a better final outcome, both for LS1 and LS2. In the case of Euclidean instances, however, the local search algorithms with either hub starting solution or recursive starting solution only marginally

improve on the solutions found using a random starting solution. For $k=32$, starting with a recursive solution results in arriving faster at a local optimum; for other values of k , computation times are comparable.

Finally, a perhaps somewhat surprising outcome is that LS1 seems better than LS2, that is, the neighbourhood that results from ‘splitting off’ one colour gives better solutions than the neighbourhood that splits off two colors. Also, computation times of LS2 are much more increasing with k than the computation times needed by LS1.

Overall, the best strategy for a local search algorithm seems to use LS1 applied to the best solution found by RECUR1 and RECUR2.

Iterated local search

How good are the local optima found by the local search algorithms? Can we find better local optima by ‘disturbing’ the current local optimum found, spend some more computation time, and end up in a new local optimum? To answer this question, we implemented an iterated local search algorithm. When a local optimum is found, we disturb it by either randomly selecting two points of a same colour and interchange their assignment (variant 1, denoted by ILS₁), or randomly select four points of the same colour, and interchange their assignment (variant 2, denoted by ILS₂). Next, we continue this procedure by using either LS1 or LS2 to arrive at a possibly different local optimum. We

Table 2 LS1

	<i>LS1</i>								
	<i>Hub starting solution</i>			<i>Recursive starting solution</i>			<i>Random starting solution</i>		
	<i>Value</i>	<i>Best</i>	<i>Time</i>	<i>Value</i>	<i>Best</i>	<i>Time</i>	<i>Value</i>	<i>Best</i>	<i>Time</i>
<i>Type 1</i>									
$k=4$	309.8	5	734	310.8	2	604	310.4	3	630
$k=8$	976	2	662	972.4	1	580	975.2	3	648
$k=16$	2871	3	1032	2865.8	5	844	2888.8	1	920
$k=32$	7439.2	2	1614	7427	5	1306	7431.8	2	1924
<i>Type 2</i>									
$k=4$	431.2	0	780	421	3	614	426.4	2	600
$k=8$	1529.4	4	770	1529	1	584	1549.4	0	706
$k=16$	4141.6	1	1032	4119.8	3	924	4159.2	1	1018
$k=32$	9733	1	2316	9677.2	4	1484	9787.4	0	2020
<i>Type 3</i>									
$k=4$	282.4	1	650	279.4	2	570	279.8	2	636
$k=8$	909.2	0	866	902.6	5	668	913.6	0	682
$k=16$	2342.4	0	1010	2331	4	910	2342.4	1	1068
$k=32$	5372.6	1	1956	5358.8	4	1634	5388	0	2010
<i>Type 4</i>									
$k=4$	1061.8	3	790	1062.2	2	580	1061.2	1	582
$k=8$	2395.4	0	694	2391.4	5	626	2395.4	1	670
$k=16$	5091.2	1	1198	5088	3	856	5091	1	1010
$k=32$	10 501.8	0	1718	10 496.4	4	1454	10 508.2	1	1848

Table 3 LS2

LS2									
Hub starting solution			Recursive starting solution			Random starting solution			
Value	Best	Time	Value	Best	Time	Value	Best	Time	
<i>Type 1</i>									
<i>k</i> = 4	324	1	612	313.4	4	606	333.4	0	670
<i>k</i> = 8	990.2	0	912	973.2	5	724	997.2	0	1066
<i>k</i> = 16	2897.8	0	2132	2868.8	5	1426	2889.8	0	2482
<i>k</i> = 32	7435.8	2	6548	7427	5	3846	7439.8	1	8588
<i>Type 2</i>									
<i>k</i> = 4	493.4	0	626	429.2	5	586	500	0	638
<i>k</i> = 8	1604.4	0	1088	1553	5	658	1617.2	0	1046
<i>k</i> = 16	4199.4	0	2670	4154.2	3	1650	4190.6	2	2244
<i>k</i> = 32	9739	0	15080	9677.8	5	7952	9761.6	0	11 744
<i>Type 3</i>									
<i>k</i> = 4	316.8	0	724	280.6	5	582	311.8	0	658
<i>k</i> = 8	947.6	0	1110	914.4	5	670	940.6	0	1054
<i>k</i> = 16	2353.2	1	2572	2345.2	5	1712	2374.2	0	3230
<i>k</i> = 32	5379.2	0	10 282	5361.6	4	5988	5383.8	1	11 984
<i>Type 4</i>									
<i>k</i> = 4	1068.2	3	624	1063.2	2	598	1068.6	0	606
<i>k</i> = 8	2404.4	0	1052	2395.4	5	670	2400.4	0	970
<i>k</i> = 16	5094.8	1	2240	5090.6	4	1540	5097.8	0	2108
<i>k</i> = 32	10 505.8	0	9110	10 494.8	4	6812	10 500.8	1	9460

run ILS₁ for 60 s, and ILS₂ for 150 s. Thus, in total, we have four iterated local search variants denoted by ILS₁(LS1) and ILS₂(LS1) (see Table 4), and ILS₁(LS2) and ILS₂(LS2) (see Table 5). Column 2 of either table reports the average values found by LS1, LS2 using as a start solution the best solution found by RECUR1 and RECUR2. Column 3 lists the average value found by ILS₁(LS1) and ILS₁(LS2). Column 4 indicates for how many instances the corresponding iterated local search variant found a better solution than LS1, LS2. Column 5 reports how many times the solution found by ILS₁(LS1), ILS₁(LS2) coincides with the best solution found in these experiments. Column 6 denotes the average value found by ILS₂(LS1), ILS₂(LS2). Column 7 indicates for how many instances the corresponding iterated local search variant found a better solution than LS1, LS2. Finally, column 8 reports how many times the solution found by ILS₂(LS1), ILS₂(LS2) coincides with the best solution found in these experiments.

Each of the iterated local search variants finds in most cases a better solution than the original local optimum. The instances of type 1 seem less susceptible for improvement than the instances of the other types. In particular, the iterated local search algorithms improve significantly the solutions found for the non-Euclidean instances when *k* = 8, 16, 32. ILS₂ is better than ILS₁ for *k* = 16, 32; for smaller values of *k*, there is no clear advantage for ILS₂.

Lower bounds

Here, we describe two different ways of computing a lower bound on the optimal value of an instance of the problem. Both lower bounds are based on applying a subgradient procedure to a Lagrangian relaxation of an integer programming formulation of the problem. In either case, the bound obtained would be dominated by solving the corresponding LP-relaxation.

A first lower bound

First, we describe how we can apply Lagrangian relaxation to formulations (1)–(3). Then we sketch the use of an iterative subgradient procedure to find the lower bounds.

Let us fix two indices, say *h*₁, *h*₂ ∈ *K*, and let us apply Lagrangian relaxation to all constraints using multipliers λ_{*r**i*}, *r* ∈ *K* \ {*h*₁, *h*₂}, *i* = 1, ..., *n*. It follows that we can write the resulting objective function (called the Lagrangian) as follows:

$$\min \sum_{a \in A^k} c_a x_a - \sum_{r \in K \setminus \{h_1, h_2\}} \sum_{i=1}^n \sum_{a \in A^k: a(r)=i} \lambda_{ri} x_a + \sum_{r \in K \setminus \{h_1, h_2\}} \sum_{i=1}^n \lambda_{ri}$$

Table 4 Iterated local search with LS1

	<i>LS1 value</i>	<i>Iterated local search</i>					
		<i>ILS₁(LS1)</i>			<i>ILS₂(LS1)</i>		
		<i>Value</i>	<i>Better</i>	<i>Best</i>	<i>Value</i>	<i>Better</i>	<i>Best</i>
<i>Type 1</i>							
<i>k</i> = 4	310.8	309.2	3	5	309.2	3	5
<i>k</i> = 8	972.4	967.8	5	5	967.8	5	5
<i>k</i> = 16	2865.8	2865.6	1	5	2865.6	2	5
<i>k</i> = 32	7427	7427	0	5	7427	0	5
<i>Type 2</i>							
<i>k</i> = 4	421	394.6	5	1	395.6	5	1
<i>k</i> = 8	1529	1456.4	5	1	1455.4	5	1
<i>k</i> = 16	4119.8	4046.8	5	0	4020.8	5	1
<i>k</i> = 32	9677.2	9641.8	4	0	9621.4	5	1
<i>Type 3</i>							
<i>k</i> = 4	279.4	262.8	5	1	263.8	5	0
<i>k</i> = 8	902.6	867	5	2	866.6	5	3
<i>k</i> = 16	2331	2293	5	1	2287.6	5	3
<i>k</i> = 32	5358.8	5337	5	0	5337.6	5	1
<i>Type 4</i>							
<i>k</i> = 4	1062.2	1055	5	3	1055	5	3
<i>k</i> = 8	2391.4	2379.6	5	2	2379.2	5	1
<i>k</i> = 16	5088	5075.6	5	1	5074.4	5	1
<i>k</i> = 32	10496.4	10487.4	5	1	10487	5	2

Table 5 Iterated local search with LS2

	<i>LS2 value</i>	<i>Iterated local search</i>					
		<i>ILS₁(LS2)</i>			<i>ILS₂(LR2)</i>		
		<i>Value</i>	<i>Better</i>	<i>Best</i>	<i>Value</i>	<i>Better</i>	<i>Best</i>
<i>Type 1</i>							
<i>k</i> = 4	313.4	313.4	1	0	313.6	0	0
<i>k</i> = 8	973.2	970.8	2	1	972	1	1
<i>k</i> = 16	2868.8	2866.6	3	3	2867	2	2
<i>k</i> = 32	7427	7427.6	0	4	7427.6	0	4
<i>Type 2</i>							
<i>k</i> = 4	429.2	430	0	0	430	0	0
<i>k</i> = 8	1553	1537.2	4	0	1535.8	5	0
<i>k</i> = 16	4154.2	4072.6	5	0	4053.6	5	0
<i>k</i> = 32	9677.8	9665	4	0	9636.8	5	0
<i>Type 3</i>							
<i>k</i> = 4	280.6	280.6	0	0	280.6	0	0
<i>k</i> = 8	914.4	908.6	4	0	905.6	4	0
<i>k</i> = 16	2345.2	2312.8	5	0	2309.2	5	0
<i>k</i> = 32	5361.6	5349	5	0	5340.2	5	1
<i>Type 4</i>							
<i>k</i> = 4	1063.2	1061.6	4	0	1061.6	5	0
<i>k</i> = 8	2395.4	2388.6	5	0	2389	5	0
<i>k</i> = 16	5090.6	5081	5	0	5078.4	5	1
<i>k</i> = 32	10494.8	10489.6	5	0	10489.4	5	1

Table 6 Lower bounds

	First lower bound	Second lower bound
<i>Type 1</i>		
$k = 4$	1.069	1.025
$k = 8$	—	1.057
$k = 16$	—	1.064
$k = 32$	—	1.071
<i>Type 2</i>		
$k = 4$	1.088	1.510
$k = 8$	—	1.820
$k = 16$	—	1.734
$k = 32$	—	1.437
<i>Type 3</i>		
$k = 4$	1.089	1.312
$k = 8$	—	1.574
$k = 16$	—	1.524
$k = 32$	—	1.347
<i>Type 4</i>		
$k = 4$	1.039	1.010
$k = 8$	—	1.018
$k = 16$	—	1.021
$k = 32$	—	1.018

Rearranging terms yields for the Lagrangian

$$\min \sum_{a \in A^K} (c_a - \sum_{r \in K \setminus \{h_1, h_2\}} \lambda_{r,a(r)})x_a + \sum_{r \in K \setminus \{h_1, h_2\}} \sum_{i=1}^n \lambda_{ri} \quad (6)$$

Observe that the remaining constraints in model (1)–(3) are nothing else but the familiar constraints of an ordinary assignment problem. Thus, we are only interested in assigning the points of A_{h_1} to the points of A_{h_2} . This implies that the only ‘interesting’ cost coefficients, say $g(i, j)$, are those for which

$$g(i, j) = \min \left\{ c_a - \sum_{r \in K \setminus \{h_1, h_2\}} \lambda_{r,a(r)} : a \in A^K, \right. \\ \left. a(h_1) = i, a(h_2) = j \right\}, \quad i, j = 1, \dots, n$$

The problem with cost coefficients $g(i, j)$ (which take $O(n^{k-2})$ operations to compute) is an ordinary assignment problem. Thus, given Lagrange multipliers $\lambda_{ri}, r \in K, 1 \leq i \leq n$, we can solve the resulting problem yielding a solution x_a^* . Obviously, this gives us a lower bound. Moreover, we use the solution x_a^* to compute a new set of values for the Lagrange multipliers, thereby employing a subgradient procedure as follows. Let the derivative μ_{ri} of the Lagrangian (6) with respect to the multipliers λ_{ri} be equal to

$$\mu_{r,i} = 1 - \sum_{a \in A^K: a(r)=i} x_a^* \quad \text{for } r \in K, 1 \leq i \leq n$$

We then iteratively set, for each $r \in K, 1 \leq i \leq n$

$$\lambda_{r,i} := \max(\lambda_{r,i} - \rho \mu_{r,i}, 0)$$

where ρ is a step length. In our implementations, we used the quotient of the difference between the current relaxation value (CRV) and the value of the best-known solution (BKS), and the norm of the vector λ as a value for ρ , that is, $\rho = (BKS - CRV) / \|\lambda\|^2$. For each pair of indices $h_1, h_2 \in K$, we ran this procedure for a 1000 iterations, and stored the best lower bound found.

Computationally, the bottleneck in this subgradient procedure is computing the costs $g(i, j)$. In fact, computation times for cases with $k > 4$ become prohibitive. Therefore, we have only computed this lower bound for the instances with $k = 4$.

This approach is a generalization of the approach employed by Frieze and Yadegar²⁵ for the 3-index assignment problem. Finally, notice that this approach is valid for any instance of MIAP, that is, we do not use here the fact that the costs are decomposable.

A second lower bound

To describe our second lower bound, we first propose a formulation that explicitly makes use of the fact that the costs c_a are decomposable. For each $u \in A_r, v \in A_s (r \neq s)$, we introduce binary variables z_{uv} indicating whether u and v are joined in a single cluster ($z_{uv} = 1$) or not ($z_{uv} = 0$). There is a formulation for each $h \in K$, so we assume some $h \in K$ is prespecified:

$$\text{minimize} \quad \sum_{r,s \in K} \sum_{u \in A_r} \sum_{v \in A_s} d(u, v) z_{uv} \quad (7)$$

$$\text{such that} \quad \sum_{v \in A_r} z_{uv} = 1 \quad \text{for } u \in A_h, r \in K \setminus h \quad (8)$$

$$\sum_{u \in A_h} z_{uv} = 1 \quad \text{for } v \in A_r, r \in K \setminus h \quad (9)$$

$$z_{uv} + z_{vw} - z_{vw} \leq 1 \\ \text{for } u \in A_h, v \in A_r, w \in A_s, r, s \in K \setminus h \quad (10)$$

$$z_{uv} \in \{0, 1\} \quad \text{for } u \in A_r, v \in A_s, r, s \in K \quad (11)$$

Equalities (8) imply that each point of A_h is assigned to a point from each other colour set; equalities (9) imply that each point not in A_h is assigned to a point from A_h . Inequalities (10) imply that if point $u \in A_h$ is assigned to $v \in A_r$ (ie, $z_{uv} = 1$) and if point $u \in A_h$ is assigned to $w \in A_s$ (ie, $z_{uw} = 1$), then the points v and w are assigned to each other (ie, $z_{vw} = 1$).

When we apply Lagrangian relaxation to constraints (10) using multipliers $\lambda_{u,v,w}$, a model results that amounts to solving $k-1$ independent assignment problems, namely for each $r \in K \setminus h$ an assignment problem between A_h and A_r . Observe that the number of multipliers equals $O(k^2 n^3)$; hence, we can compute the cost coefficients in the resulting Lagrangian relatively fast. Similarly, as for the first Lagrangian relaxation, we compute the derivative $\mu_{u,v,w}$ of $\lambda_{u,v,w}$, and we set

$$\lambda_{u,v,w} := \lambda_{u,v,w} + \rho \mu_{u,v,w}$$

for all $u \in A_h, v \in A_r, w \in A_s, r, s \in K \setminus h$

We used a value of $\rho = 0.1$ throughout the procedure and, for each choice of $h \in K$, we ran the procedure for a 1000 iterations.

Performance of the lower bounds

The performance of the lower bounds described above is reported in Table 6. In this table, we report in columns 2 and 3 the average value (over five instances) of the quotient of the best solution found and the average value of the first and second lower bound, respectively.

The first lower bound shows that, at least for the instances with $k=4$, a solution is found within 10% of the optimum. The results are better for the Euclidean instances. The second lower bound turns out to be quite strong for the Euclidean instances, dominating the first lower bound ($k=4$). As an example, the best solutions found to the instances of type 4 with $k=4$ are (on average) within 1% of the optimum. Also, the performance on type 1 instances is quite satisfactory, and only moderately increases with k . However, the second lower bound is quite weak on the non-Euclidean instances, in particular, for type 2 instances.

Conclusion

This paper deals with the MIAP with decomposable costs. We proposed a local search algorithm based on a new neighbourhood. We tested this algorithm along with constructive heuristics. Our conclusions can be summarized as follows:

- the hub heuristic MULTI-HUB needs very little computing time,
- the recursive heuristics RECUR1 and RECUR2 give better solutions than MULTI-HUB, especially for non-Euclidean instances,
- local search improves the solutions found by the constructive heuristics; in particular, the neighbourhood based on 'splitting off' one colour set significantly improves the solution values for the non-Euclidean instances,

- an iterated local search method is able to improve on the local optima found so far; for the Euclidean instances, we can prove, using lower bounds computed by a subgradient procedure, that the solutions found are on average within 8% of the optimum value (and are often much better).

Summarizing, local search is a viable, attractive method for computing solutions to MIAPS with decomposable costs.

References

- 1 Bandelt HJ, Crama Y and Spieksma FCR (1994). Approximation algorithms for multidimensional assignment problems with decomposable costs. *Discrete Appl Math* **49**: 25–50.
- 2 Pierskalla WP (1968). The multidimensional assignment problem. *Opns Res* **16**: 422–431.
- 3 Spieksma FCR (2000). Multi index assignment problems: complexity, approximation, applications. In: Pitsoulis L and Pardalos P (eds). *Nonlinear Assignment Problems, Algorithms and Applications*. Kluwer, Dordrecht, pp 1–12.
- 4 Fortin D and Tusera A (1994). Routing in meshes using linear assignment problem. In: Bachem A, Derigs U, Jünger M and Schrader R (eds). *Operations Research '93*. Physica-Verlag, Heidelberg, pp 169–171.
- 5 Pusztaszeri JF, Rensing PE and Liebling TM (1996). Tracking elementary particles near their primary vertex: a combinatorial approach. *J Global Optim* **9**: 41–64.
- 6 Morefield CL (1977). Applications of 0–1 integer programming to multitarget tracking problems. *IEEE Trans Automat Control* **22**: 302–312.
- 7 Pattipati KR, Deb S, Bar-Shalom Y and Washburn R (1992). A new relaxation algorithm and passive sensor data association. *IEEE Trans Automat Control* **37**: 197–213.
- 8 Poore AB and Rijavec N (1993). A Lagrangian relaxation algorithm for multi-dimensional assignment problems arising from multitarget tracking. *SIAM J Optim* **3**: 545–563.
- 9 Robertson AJ (2001). A set of greedy randomized adaptive local search procedure (GRASP) implementations for the multidimensional assignment problem. *Comput Optim Appl* **19**: 145–164.
- 10 Chummun MR, Kirubarajan T, Pattipati KR and Bar-Shalom Y (2001). Fast data association using multidimensional assignment with clustering. *IEEE Trans Aerospace Electron Systems* **37**: 898–913.
- 11 Poore AB (1994). Multidimensional assignment formulations of data-association problems arising from multitarget and multi-sensor tracking. *Comput Optim Appl* **3**: 27–57.
- 12 Storms P and Spieksma FCR (2003). An LP-based algorithm for the data association problem in multitarget tracking. *Comput Opns Res* **30**: 1067–1085.
- 13 Veenman CJ, Reinders MJT and Backer E (2003). Establishing motion correspondence using extended temporal scope. *Artif Intell* **145**: 227–243.
- 14 Spieksma FCR and Woeginger GJ (1996). Geometric three-dimensional assignment problems. *Eur J Opl Res* **91**: 611–618.
- 15 Bierlein R. (1993). *Mehrdimensionale Zuordnungsprobleme: Algorithmen und Anwendungen*. Master's thesis, University of Passau, Germany.
- 16 Crama Y and Spieksma FCR (1992). Approximation algorithms for three-dimensional assignment problems with triangle inequalities. *Eur J Opl Res* **60**: 273–279.

- 17 Magos D, Mourtos I and Appa G (2002). *Polyhedral results for assignment problems*. Research Report LSE-CDAM-2002-01, London School of Economics, UK.
- 18 Ruschendorf L (1983). On the multidimensional assignment problem. *Methods Opns Res* **47**: 107–113.
- 19 Gilbert KC and Hofstra RB (1988). Multidimensional assignment problems. *Decision Sci* **19**: 306–321.
- 20 Aiex RM, Resende MGC, Pardalos PM and Toraldo G (2000). *GRASP with path relinking for the three-index assignment problem*. Technical Report, AT&T Labs, US.
- 21 Aarts EHL and Lenstra JK (eds) (1997). *Local Search in Combinatorial Optimization*. Wiley: Chichester.
- 22 Ahuja RK, Ergun , Orlin JB and Punnen AP (2002). A survey of very large-scale neighborhood search techniques. *Discrete Appl Math* **123**: 75–102.
- 23 Jonker R and Volgenant A (1987). A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing* **38**: 325–340.
- 24 Dell’Amico M and Toth P (2000). Algorithms and codes for dense assignment problems: the state of the art. *Discrete Appl Math* **100**: 17–48.
- 25 Frieze AM and Yadegar J (1981). An algorithm for solving 3-dimensional assignment problems with application to scheduling a teaching practice. *J Opl Res Soc* **32**: 989–995.

*Received September 2002;
accepted January 2004 after one revision*