

Joint MAP Bias Estimation and Data Association: Simulations

Scott Danford, Bret Kragel, and Aubrey Poore
Numerica Corporation
P.O. Box 271246
Fort Collins, CO 80527-1246

ABSTRACT

The problem of joint maximum a posteriori (MAP) bias estimation and data association belongs to a class of nonconvex mixed integer nonlinear programming problems. These problems are difficult to solve due to both the combinatorial nature of the problem and the nonconvexity of the objective function or constraints. Algorithms for this class of problems have been developed in a companion paper of the authors. This paper presents simulations that compare the “all-pairs” heuristic, the k-best heuristic, and a partial A*-based branch and bound algorithm. The combination of the latter two algorithms is an excellent candidate for use in a realtime system. For an optimal algorithm that also computes the k-best solutions of the joint MAP bias estimation problem and data association problem, we investigate a branch and bound framework that employs either a depth-first algorithm or an A*-search procedure. In addition, we demonstrate the improvements due to a new gating procedure.

Keywords: MAP Bias Estimation, Data Association, Heuristics, A*-Search, Branch and Bound, K-Best Solutions, Nonconvex MINLP

1. INTRODUCTION

This work is a continuation of a companion paper by the authors¹ on the joint MAP bias estimation and association problem as formulated in Section 2. An introduction to the problem, references to the literature, and algorithms can be found there. The goal of this work then is to present some simulations to demonstrate the performance of some of the algorithms.

The problem of joint bias estimation and data association is a mixed integer nonlinear programming problem. The only known methods for solving this problem optimally use exhaustive search procedures, for example, branch and bound or A*-search. Such procedures are not amenable to realtime needs and so one frequently resorts to heuristics to construct a good solution. Our approach¹ has been to develop good heuristics followed by a branch and bound or A*-search algorithm that can improve the heuristic as time allows.

In this paper we show the numerical performances of two heuristics. The first is called an “all-pairs” algorithm in that it computes a translation bias for each pair of assignments followed by a local search to improve the quality of the solution. The “all-pairs” heuristic produces good quality solutions; however, its deficiency is its runtime performance. We have developed an alternate algorithm that appears to retain the quality of the solutions with superior runtime. This algorithm is based on covariance inflation and adaptive cost shifting and displacements computed from arcs in the k-best solutions.

While heuristics can perform well on one set of problems and not the next, we augment the heuristic with a branch and bound or A*-search procedure that can iteratively improve the heuristic and can converge to optimality given sufficient time. The branch and bound framework consists of many variations depending on the construction of an upper and lower bound, the search procedure, and the partitioning algorithm. In this work, we concentrate on the search procedure by investigating two such procedures, namely, best first search based on lower bound, i.e., an A*-search algorithm, and a depth-first search.

The paper is organized as follows. The formulation of the joint MAP bias estimation and data association problem investigated here is presented in the Section 2. The heuristics are reviewed in Section 3. The branch and bound algorithm is outlined in Section 4. The scenarios used for the simulations are presented in Section 6 and performances of the different algorithms are compared in Section 7.

2. THE MAP BIAS ESTIMATION-ASSOCIATION PROBLEM

Consider an unknown number N of objects moving independently in space, some of which are being tracked by two sensors, with sensor 1 having tracks on $m \leq N$ objects, while sensor 2 is tracking $n \leq N$ objects. The sensor tracks for each object consist of an estimate of the true, but unknown, kinematic state of the object, along with a covariance matrix describing the stochastic uncertainty in the estimate. In general, the number of objects tracked by each sensor will be different, i.e., $m \neq n$, and it will often be the case that each sensor has tracks on some objects not being tracked by the other sensor. The formulation we use for algorithm testing is

$$\begin{aligned}
 & \text{Minimize}_{(x,d)} c_r(d) + \sum_{(i,j) \in A} c_{ij}(d)x_{ij} \\
 & \text{Subject To: } \sum_{j \in A(0)} x_{0j} = n, \\
 & \sum_{j \in A(i)} x_{ij} = 1 \quad (i = 1, \dots, m), \\
 & \sum_{i \in B(0)} x_{i0} = m, \\
 & \sum_{i \in B(j)} x_{ij} = 1 \quad (j = 1, \dots, n), \\
 & x_{ij} \in \begin{cases} \{0, 1\} & \text{for } i \neq 0 \text{ or } j \neq 0, \\ \{0, 1, \dots, \text{Min}\{m, n\}\} & \text{for } i = 0 \text{ and } j = 0, \end{cases}
 \end{aligned} \tag{1}$$

where A is the set of feasible arcs, $A(i) = \{j \mid (i, j) \in A\}$ and $B(j) = \{i \mid (i, j) \in A\}$, such that $A(0) = J$, $B(0) = I$, and

$$\begin{aligned}
 c_r(d) &= \frac{1}{2}d^T(R_{xx} - R_{xy} - R_{yx} + R_{yy})^{-1}d \\
 c_{ij}(d) &= \frac{1}{2}(\hat{x}_i + d - \hat{y}_j)^T Z_{ij}^{-1}(\hat{x}_i + d - \hat{y}_j)
 \end{aligned} \tag{2}$$

$$+ \frac{1}{2} \ln [\beta_T^2 |2\pi Z_{ij}|] + \gamma_{ij} \text{ for } i \neq 0 \text{ and } j \neq 0, \tag{3}$$

$$Z_{ij} = P_{ii} - S_{ij} - S_{ji} + Q_{jj} \tag{4}$$

where β_T denotes the target density, $\gamma_{ij} = \ln((1 - P_D^1)(1 - P_D^2))$, and P_D^k , $k \in \{1, 2\}$ is the probability of sensor k observing a particular target.

Similar to the book by Blackman and Popoli² and publications by Levedahl,^{3,4} the terms involving the registration covariance matrix R serve to include a prior estimate of the bias in the objective function. Note also that the singleton costs $c_{i0} = c_{0j} = 0$ are independent of d .

3. LOCAL SEARCH AND HEURISTICS

In this section, we briefly review the local search algorithm and two heuristics, namely, the ‘‘all-pairs’’ heuristic and the k-best heuristic, used in our simulations.

3.1 Local Search Algorithm

The local search algorithm starts with a given d , then iteratively solves for the assignment and updates the bias d until the assignments no longer change. Here is a more formal description.

- (a) Given a bias estimate d , compute a corresponding assignment $S = \{(i, j) \mid x_{ij} = 1\}$ as a solution of the assignment problem (1), but with d fixed.

(b) Given an assignment S , compute the bias by solving

$$\text{Minimize}_d \quad c_r(d) + \sum_{(i,j) \in S} c_{ij}(d).$$

(c) Repeat a) and b) until the assignment no longer changes.

This is generally an expensive algorithm, but its use for a few instances is warranted.

3.2 All-Pairs Heuristic

For each feasible arc $(i, j) \in A$,

(a) Compute the solution d from $D_d c_{ij}(d) = 0$.

(b) Given d from (a), apply the local search algorithm (3.1).

We have added a time improvement to this algorithm by

(c) If during (b) an assignment S or displacement d has been computed in an earlier local search, then stop the local search.

The addition of part (c) has significantly reduced algorithm runtime in many cases.

3.3 Heuristics Based on K-Best Solutions

This class of heuristics starts with two initial preprocessing components: covariance inflation and cost shifting as explained in the work of Danford, Kragel, and Poore.¹ Assuming the covariance matrices have been inflated and the costs have been shifted, the k-best solutions are computed with $d = 0$. The arcs in the k^{th} best solution are denoted by S_k . Given this notation, one can generate several heuristics explained as follows.

K-Best Intersection Heuristic. For $\ell = 1, \dots, k$ solve the least squares problem

$$\text{Minimize}_d \quad c_r(d) + \sum_{(i,j) \in \cap_{p=1}^{\ell} S_p} c_{ij}(d)$$

for a d_{ℓ} . Note that the number of d 's generated is k . Next one should use the local search algorithm in Subsection 3.1 to improve the bias and assignment starting from each d_{ℓ} ($\ell = 1, \dots, k$).

K-Best All-Pairs Heuristic. For each arc $(i, j) \in \cup_{\ell=1}^k S_{\ell}$ solve $D_d c_{ij}(d) = 0$. The number of d 's generated is provably $O(K \text{Minimum}\{m, n\})$, but is most often approximately $\text{Minimum}\{m, n\} + K$. The local improvement algorithm is employed to improve each of these d 's.

K-Best Heuristic. An algorithm that is competitive with the All-Pairs Heuristic in quality but superior in runtime is to first inflate covariances and use adaptive cost shifting to generate a good initial assignment. Then, we combine the K-Best Intersection Heuristic and K-Best All-Pairs Heuristic.

4. A BRANCH AND BOUND FRAMEWORK

The branch and bound procedure described above will be used to solve the optimization problem

$$\begin{aligned} & \text{Minimize } f(d, x) \\ & \text{Subject to } x \in F \\ & \quad d \in \mathfrak{R}^n \end{aligned}$$

where F is a finite set. The variable d is a continuous variable. The branch and bound procedure uses an acyclic graph known as a branch and bound tree that forms progressively finer partitions of the finite set F defined by the constraints and integer variables in equation (1). The nodes in the tree correspond to a collection \mathcal{F} of subsets of F . Also, the algorithm maintains a node list called ‘‘OPEN’’ and an upper bound ‘‘UPPER,’’ which is the minimal cost of all feasible solutions found so far. OPEN is initialized with the node F along with corresponding information such as the lower bound, while UPPER is set equal to the cost of a feasible solution (\bar{d}, \bar{x}) found by a heuristic such as one of those noted in Section 3.

Salient Features of the Branch and Bound Algorithm

Step 1. Initialization: Compute a global lower bound $LB(F)$, feasible solution (\bar{d}, \bar{x}) , and an upper bound $UB(F) = f(\bar{d}, \bar{x})$. If $LB(F) = UB(F)$, stop for the optimal solution (\bar{d}, \bar{x}) has been found; otherwise, continue. $OPEN = \{(F, LB(F))\}$ and $UPPER = UB(F)$.

Step 2. Remove a node N from OPEN.

Step 3. If $LB(N) \geq UPPER$, the node N is said to be fathomed and can be deleted from further consideration; otherwise, find a global solution (\bar{d}, \bar{x}) for the node N or partition N into $\{N_1, \dots, N_k\}$ nodes. In the former case, if $f(\bar{d}, \bar{x}) < UPPER$, set $UPPER = f(\bar{d}, \bar{x})$, mark (\bar{d}, \bar{x}) as the best solution so far, and go to Step 4. In the latter case, for each child N_i of N , do the following

3a. Compute a lower bound $LB(N_i)$. If $LB(N_i) < UPPER$, place the node N_i in OPEN:

$$OPEN = OPEN \cup \{(N_i, LB(N_i))\}; \tag{5}$$

Otherwise, N_i is said to be fathomed.

3b. Compute a feasible solution (\bar{d}, \bar{x}) and an upper bound $UB(N_i) = f(\bar{d}, \bar{x})$. If $UB(N_i) = f(\bar{d}, \bar{x}) < UPPER$, set $UPPER = f(\bar{d}, \bar{x})$ and mark (\bar{d}, \bar{x}) as the best solution so far.

Step 4. Termination Test: If OPEN is nonempty, go to Step 2; otherwise, terminate as the best solution found thus far is optimal.

In Step 1 above, the case $LB(F) = UB(F)$ will almost never occur due to the sensor and track state uncertainty. In the following, we address each of these steps in the above generic algorithm.

4.1 The Root Node

The root node in the branch and bound tree is defined by the original problem (1). For this node, one must supply a global lower bound, an initial feasible solution and its corresponding upper bound, along with a partition of the problem into disjoint components.

4.1.1 A Global Lower Bound for the Root Node

The global lower bound required in the branch and bound algorithm is obtained by replacing the costs $c_{ij}(d)$ by

$$\begin{aligned} b_{ij} &= \frac{1}{2} \ln[\beta_T^2 |2\pi Z_{ij}|] + \gamma_{ij} \text{ for } i, j \neq 0, \\ b_{ij} &= 0 \text{ for } i = 0 \text{ or } j = 0, \end{aligned} \tag{6}$$

which are independent of the displacement d . Furthermore, $b_{ij} \leq c_{ij}(d)$ for all i, j , thereby providing a basis for the global lower bound needed in the branch and bound procedure.

4.1.2 The Initial Feasible Solution

The initial feasible solution that serves as an upper bound on the optimal solution can be obtained by any of the heuristics in Section 3. Such a solution produces a bias estimate \bar{d} and an assignment $S = \{(i, j) \in \mathcal{A} \mid \bar{x}_{ij} = 1\}$.

4.2 A Descendant Node

In this section, a descendant node in the branch and bound tree, the optimization problem, a global lower bound for the node, and the construction of a feasible solution are formulated.

A node in the branch and bound tree is defined by arcs $(k, \ell) \in A$ that are in the assignment, $S = \{(k, \ell) \neq (0, 0) \mid x_{k\ell} = 1\}$, arcs declared not to be in any assignment for that node, $D = \{(k, \ell) \neq (0, 0) \mid x_{k\ell} = 0\}$, and arcs that are free to be assigned, $A - S - D$. For the root node, $S = D = \emptyset$ and for a leaf node, $A = S \cup D \cup \{(0, 0)\}$.

4.2.1 The Optimization Problem at a Descendant Node

Define

$$\begin{aligned}\tilde{A}(i) &= \{j \mid (i, j) \in A - S - D \text{ for some } j\}, \\ \tilde{B}(j) &= \{i \mid (i, j) \in A - S - D \text{ for some } i\}, \\ I &= \{i \mid \tilde{A}(i) \text{ contains at least one nonzero index}\}, \\ J &= \{j \mid \tilde{B}(j) \text{ contains at least one nonzero index}\}.\end{aligned}$$

Then, the optimization problem for each node has the form

$$\begin{aligned}\text{Minimize}_{(d,x)} & \left[c_r(d) + \sum_{(i,j) \in S} c_{ij}(d) + \sum_{(i,j) \in A-D-S} c_{ij}(d)x_{ij} \right] \\ \text{Subject To:} & \sum_{j \in \tilde{A}(0)} x_{0j} = |J|, \\ & \sum_{j \in \tilde{A}(i)} x_{ij} = 1 \quad (i \in I, i \neq 0), \\ & \sum_{i \in \tilde{B}(0)} x_{i0} = |I|, \\ & \sum_{i \in \tilde{B}(j)} x_{ij} = 1 \quad (j \in J, j \neq 0), \\ & x_{ij} \in \begin{cases} \{0, 1\} & \text{for } i \neq 0 \text{ or } j \neq 0, \\ \{0, 1, \dots, \text{Min}\{|I|, |J|\}\} & \text{for } i = 0 \text{ and } j = 0, \end{cases}\end{aligned} \tag{7}$$

4.2.2 A Lower Bound at a Descendant Node

A lower bound is obtained by replacing $c_{ij}(d)$ in the final term in the objective function in equation (7) by b_{ij} from equation (6) for each $(i, j) \in A - D - S$. The objective function in (7) then decomposes into two components:

$$\text{Minimize}_d \left[c_r(d) + \sum_{(i,j) \in S} c_{ij}(d) \right] + \text{Minimize}_x \left[\sum_{(i,j) \in A-D-S} b_{ij}x_{ij} \right]$$

subject to the constraints in (7). The two optimization problems in this equation can be solved efficiently by starting with the corresponding solutions of the parent node.

4.2.3 A Feasible Solution and Upper Bound for the Descendant Node

While there are several methods for computing an upper bound for a descendant node, here are a couple that we have found to work well. The first is to start with the solution d of

$$\text{Minimize}_d \left[c_r(d) + \sum_{(i,j) \in S} c_{ij}(d) \right],$$

which is computed in the lower bound. The second would be to take the d from the parent feasible solution. For either case, the resulting d is used to obtain an assignment $P = \{(i, j) \in A - S - D \mid x_{ij} = 1\}$. This yields a (\bar{d}, \bar{x}) with $\bar{d} = d$ and $\bar{x}_{ij} = 1$ if $(i, j) \in S \cup P$. The corresponding upper bound is given by $c_r(\bar{d}) + \sum_{(i,j) \in S \cup P} c_{ij}(\bar{d})$. This upper bound and corresponding solution (\bar{d}, \bar{x}) can then be improved by using the local search algorithm discussed in Section 3.1 adapted to the problem (7).

4.3 Partitioning a Node

Having explained the optimization problem, the lower bound, and the construction of a feasible solution that serves as an upper bound for the node, we now turn to the partitioning algorithm. One could use the partition developed by Murty⁵ modified to the current inequality constrained problem. A second method of partitioning and the one used in this work is to partition a node based on the different ways a constraint can be satisfied. For example, given a constraint $\sum_{j \in \tilde{B}(i)} x_{ij} = 1$ for some $i \in I$ we could enumerate the $j \in \tilde{B}(i)$ by $\{j_0, j_1, \dots, j_r\}$ with $j_0 = 0$. For each of these values j_p we create a child node, in which $x_{ij_p} = 1$, $x_{ij} = 0$ for $j \neq j_p$, and $x_{kj_p} = 0$ for $k \neq i$. Since $\sum_{j \in \tilde{B}(i)} x_{ij} = 1$, the union of the corresponding nodes is the parent, and the intersection of any two of these nodes is the empty set. Therefore, the solution spaces of these nodes form a partition of the solution space of the parent node.

4.4 Search Procedure for Node Selection

In the branch and bound procedure, the selection of the next node from the list OPEN is called a search procedure. We pursue two such algorithms here, namely A* search, which is a best-first search based on the lower bound, and depth-first search with node ordering, i.e., A* per level.

4.4.1 A*-Search

Given a node N defined by the arc sets S and D , the lower bound is defined by

$$\text{Minimize}_d \left[c_r(d) + \sum_{(i,j) \in S} c_{ij}(d) \right] + \text{Minimize}_x \left[\sum_{(i,j) \in A - D - S} b_{ij} x_{ij} \right],$$

wherein, one can identify $\text{Minimize}_d \left[c_r(d) + \sum_{(i,j) \in S} c_{ij}(d) \right]$ as the backward cost from the current node to the root node and $\text{Minimize}_x \left[\sum_{(i,j) \in A - D - S} b_{ij} x_{ij} \right]$ as the underestimate of the forward cost to any leaf node from node N . Thus, this could be used for A*-search.

The drawback of A*-search is that it requires memory that grows exponentially with the problem size. This also affects the speed due to the overhead of maintaining a priority queue.

4.4.2 Depth-First Search

In a depth-first branch and bound algorithm, the node with the most arcs fixed, i.e., $(i, j) \in S \cup D$, is chosen from those in OPEN to explore next. Since many nodes may have the same number of constraints at the same level in the branch and bound tree, ties are broken by selecting the node with the lowest lower bound. This tie breaking rule, which is essentially A* per level, is called *node ordering*. A depth-first search requires memory that grows linearly with the problem size. Thus, its memory requirements are small compared to A*-search and it can be used on much larger problems without running out of space or spending a great deal of time maintaining a queue.

4.5 Gating Arcs in a Node

It is possible to gate the arcs in the assignment problem for each node N by first observing that costs in an optimal solution must satisfy $c_{ij}(d) \leq 0$. Recall the optimization problem has the objective function $\text{Minimize}_{(d,x)} \left[c_r(d) + \sum_{(i,j) \in S} c_{ij}(d) + \sum_{(i,j) \in A-D-S} c_{ij}(d)x_{ij} \right]$. The costs $c_{ij}(d)$ ($(i,j) \in S$) define a region in the bias space, say R_S in which d must lie. Thus, any arc $(i,j) \in A-D-S$ for which $\{d \mid c_{ij}(d) \leq 0\} \cap R_S = \emptyset$ can be ruled out from further consideration in the optimization problem. While this region is too complex to implement, there is a less stringent gate that can be efficiently implemented. The idea is to replace each $\{d \mid c_{ij}(d) \leq 0\}$ with the smallest cuboid containing this set. To explain, let $C \in \mathbb{R}^{p \times p}$ be positive definite. Then

$$\{\epsilon \mid \epsilon^T C^{-1} \epsilon \leq G\} \subset \{\epsilon \mid |\epsilon_k| \leq \sqrt{GC_{kk}} \text{ for } k = 1, \dots, p\}$$

where C_{kk} is the k^{th} diagonal element in C . This fact motivates the definition of the cuboid

$$B_{ij}(d) = \left\{ d \mid |(x_i - (d + y_j))_k| \leq \sqrt{[-\ln[\beta_T^2 |2\pi Z_{ij}|] - 2\gamma_{ij}](Z_{ij})_{kk}} \right\},$$

where $Z_{ij} = P_{ii} + Q_{jj} - S_{ij} - S_{ji}$. Then any arc (i,j) for which $B_{ij}(d) \cap B_S = \emptyset$ where $B_S = \cap_{(i,j) \in S} B_{ij}(d)$ cannot be in the optimal solution with those already in S .

5. STAND-ALONE A*-SEARCH

The A*-search procedure embedded in the the above branch and bound framework can stand alone and yields an optimal solution. Of course, this best first search based on the lower bound still requires the partitioning algorithm in Section 4.3 to create a tree structure, but it does not require that one compute an upper bound at each node. The GNPL algorithm due to Levedahl^{3,4} is an A*-search in this sense, since the costs are shifted to be nonnegative in which case his search procedure is also based on a lower bound. A noteworthy observation for A*-search is that the first k-leaf nodes explored in the A*-search are the k-best solutions. What is more, this search can be made even more efficient through the use of gating in Section 4.5. Additionally, the upper bound obtained from the heuristics in Section 3 can be used to fathom nodes in the A* tree.

6. TWO SCENARIOS

For purposes of algorithm comparisons, we make use of two different scenarios as described below. The first is from a high-fidelity simulation. The second is a low-fidelity simulation for which we know truth and can generate problems similar to those found in the work of Levedahl.³

6.1 High-Fidelity Scenario

This scenario takes track states generated by a radar simulator in which biases such as range and angle offsets, and atmospheric refraction were modeled. Multiple closely spaced targets are simulated. Two radar trackers periodically send their track state estimates to a centralized fusion platform where tracks are propagated to the same time. These time-aligned track state estimates are the inputs for the joint bias estimation and assignment algorithm. Note that the algorithm still assumes that the biases take the form of constant offsets, and that the estimates are actually Gaussian estimates of the true target location. These assumptions are violated in this environment. The sensors may not have perfect measurement data association, meaning each track state estimate could be blending information from different physical targets, and the biases include terms that are not constant offsets in the kinematic state; thus, there is a bias model mismatch.

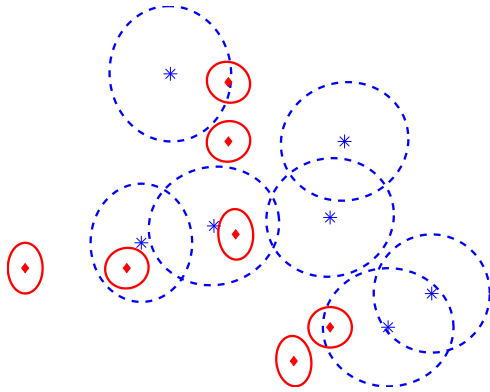


Figure 1. The effect of bias on data association.

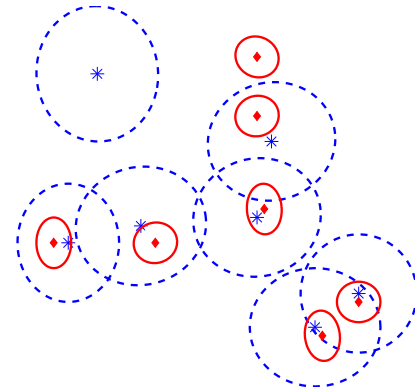


Figure 2. Data association after bias correction.

6.2 Artificial Scenario

This scenario uses a simple simulator which provides a variety of different practice problems on which to test the algorithms. This simulator begins by producing true target positions. For each Monte Carlo run, we generate thirty targets. The targets’ true states are drawn from a Gaussian distribution. The mean of this distribution is a point on a feasible ballistic trajectory. The standard deviation is spherical and set to produce appropriate separation between the targets. A Gaussian distribution was chosen over a uniform in order to represent the starburst shape for a family of separating ballistic objects.

Next, two sensor platforms each make measurements of the targets. First, independent Bernoulli trials determine which targets each sensor observes. One of the sensors has a very high chance of observing targets, while the other often misses. This is meant to emulate a situation in which one observing platform has high resolution, and can observe even small targets, while another requires a high radar cross section in order to report a track. Each sensor also generates a bias by drawing a state vector offset from a Gaussian distribution. The sensor reports its believed covariance for this offset for use in the prior information term of the optimization problem, but the covariance used to generate it was multiplied by a uniformly distributed bias multiple. This multiple represents the amount by which the sensor is misrepresenting its own biases.

Finally, for each target observed, a tracking error covariance matrix is constructed nondeterministically to represent the inherent uncertainty in sensor readings and dynamics models. The final sensor measurement is the true target position, plus the independently drawn tracking error, plus the bias offset for that sensor.

As an illustration of the problem, Figure 1 shows the impact of a bias on data association. Without addressing biases, one would incorrectly associate four of the objects with overlapping covariances. The six correct matches are illustrated in Figure 2 in which the tracks from one sensor are translated to the other by an offset d .

7. SIMULATIONS

In the following graphs, we present the results of Monte Carlo testing. We begin by analyzing the performance of various “all-pairs” heuristics, followed by a comparison with the k-best heuristic. Next, we study the different search procedures used in the branch and bound algorithm. The “anytime” algorithm is composed of the k-best heuristic and an A* partial branch and bound procedure. The performance gains in combining a heuristic with an exact algorithm demonstrate the value of such a combination. The improvements due to gating are also presented. Finally, we compare the association quality in the presence of biases when nothing is done, when inflation is added, and when the joint MAP bias estimation and association algorithm is used.

7.1 Heuristics

In this section, we first demonstrate some of the performance characteristics of the “all-pairs” heuristic and its enhancements, and then compare with the k-best heuristic.

7.1.1 “All-Pairs” Heuristic

There are three parts of the “all-pairs” algorithm, namely, the use of the initial d arising as a solution of $D_d c_{ij}(d) = 0$, the addition of the local search for each d computed, and the enhancement due to memory management.

Figure 3 demonstrates the necessity of using the local search when employing this algorithm. These tests were taken from the high-fidelity simulation environment.

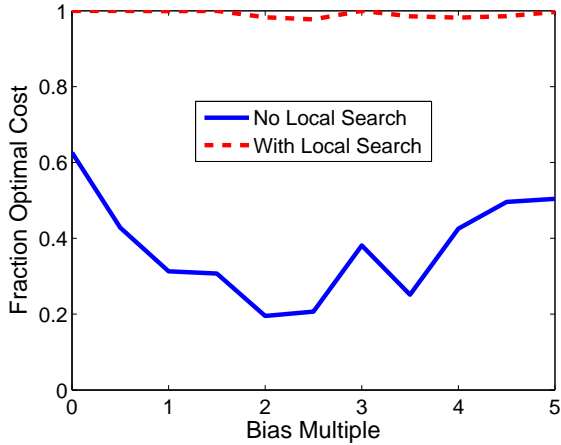


Figure 3. The percentage of the optimal cost obtained.

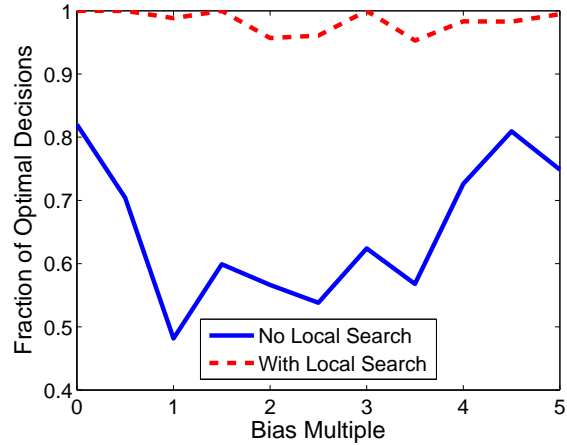


Figure 4. The percentage of optimal decisions made.

Unfortunately, the time required to perform the local search at each location can be excessive. If the previous assignments obtained are stored, considerable time savings can be achieved by not duplicating computations. Figure 5 shows the time required by the three “all-pairs” variations using data from the high-fidelity simulation environment. The best results for the “all-pairs” algorithm came from using local search and memory management. The disadvantage is that the time required to solve larger problems is too large; the “all-pairs” with no local search is much faster, but the quality of the solution is unacceptable. In the remainder of this paper, we use the term “all-pairs” to refer to the heuristic with local search and memory management.

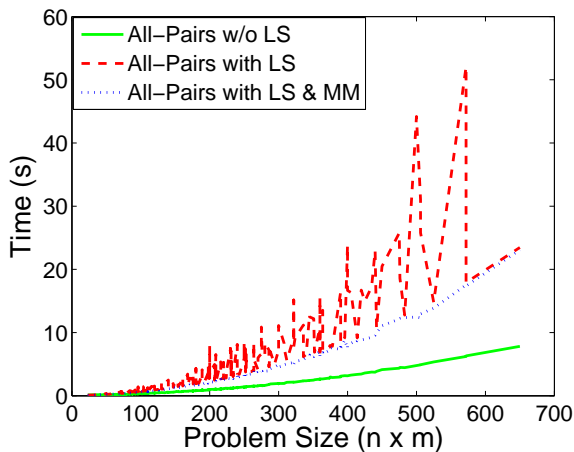


Figure 5. The time required by the three variations of the “all pairs” algorithm.

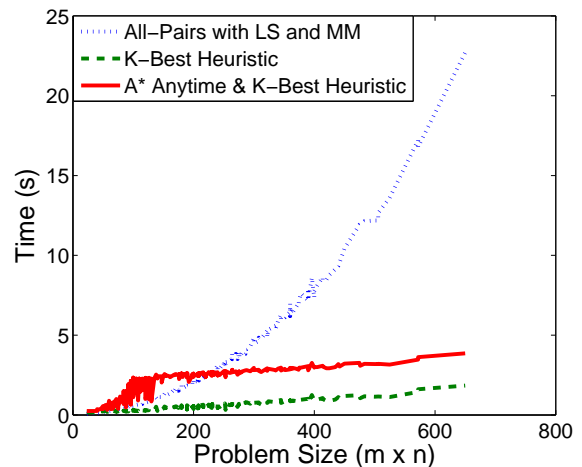


Figure 6. The time required by the heuristic and anytime algorithms.

7.1.2 Comparison of Heuristics

In this section, we compare an “anytime” or partial branch and bound algorithm that utilizes A*-search with the k-best heuristic and “all-pairs” heuristic. The “anytime” algorithm is allowed to run an additional 2 seconds after being passed the solution found by the k-best heuristic. The results using the high-fidelity simulation environment are found in Figures 6, and 7. Figure 7 is plotted with respect to the bias level in order to stress the uncertainty in the bias covariance as reported by the sensor. We make this comparison because bias statistics are often wrong and the algorithm must be robust to the model mismatch. A bias level of one indicates perfect agreement with the statistics reported by the sensor while a level of five indicates that the bias statistics are actually five times that reported by the sensor.

Figure 7 shows the better performance of the k-best heuristic compared to that of the “all-pairs” heuristic. On the other hand, adding an A* “anytime” algorithm to the k-best heuristic is better still.

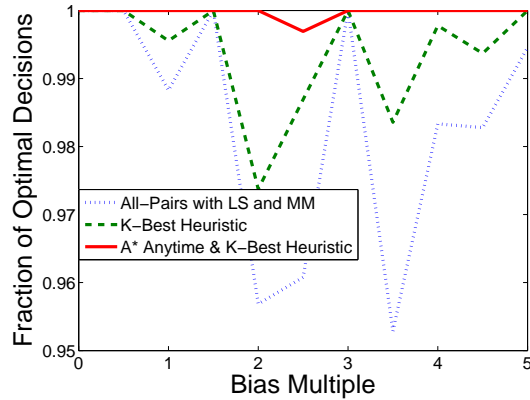


Figure 7. The decisions made by the “all-pairs”, combined heuristic and anytime algorithms.

In Figure 6, we demonstrate the superior runtime performance of k-best and A* anytime algorithm compared to that of the “all-pairs” heuristic. The “all-pairs” algorithm does have good runtime performance on problems of size up to $mn = 100$, e.g., ten by ten, but even then, the k-best heuristic is faster.

Now one may conjecture about the performance difference between A*-search and DFS in the “anytime” algorithm. To demonstrate the difference, we set the initial upper bound to 75% of optimal to see the difference. The result is illustrated in Figure 8. These results support the choice of an informed search (A*) rather than an uninformed search (DFS) as part of the partial branch and bound.

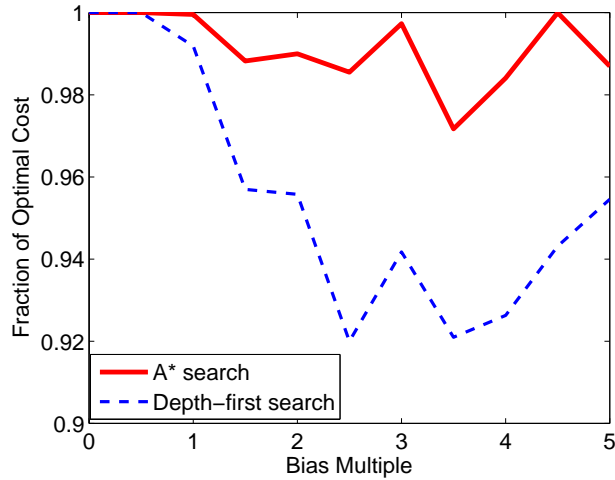


Figure 8. The amount of improvement from 75% of the optimal by A* and depth-first searches.

In the artificial scenario the k-best heuristic found the optimal solution in 997 out of 1000 runs. The “anytime” algorithm improved this to 1000 out of 1000. The “all-pairs” algorithm with local search and memory management missed the optimal solution only once.

Our conclusion then is that the k-best heuristic is superior to the “all-pairs” algorithm. It is competitive in quality and superior in runtime. The best of these algorithms is, however, the combination of k-best heuristic with a limited A*-search.

7.2 Branch and Bound Search Procedures

A*-search has exceptional optimality characteristics, but the memory required to solve problems to optimality can grow exponentially in the size of the problem. This is further explained in the book by Russell and Norvig [6, Chapter 4].

Figure 9 compares the memory requirements of DFS and A*-search plotted on a log scale versus problem size. The trend is exponential for the A*-search, but linear for the DFS; thus, if memory is a concern, DFS appears to be the clear choice provided that the k-best heuristic is used to generate the initial upper bound.

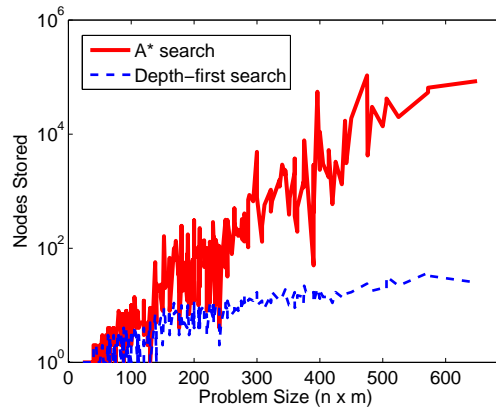


Figure 9. The memory requirements of A*-search compared to depth-first search.

Figures 10-11 compare the number of nodes that were expanded by the different search procedures. The DFS algorithm with the initial k-best heuristic expands slightly more nodes than A*-search. We add that this feature is due to the capability of the k-best heuristic to provide an initial near-optimal upper bound. The results when not using this heuristic are shown in Figure 11. With the heuristic, the DFS is slightly more efficient since it does not need to store and maintain a very large priority queue.

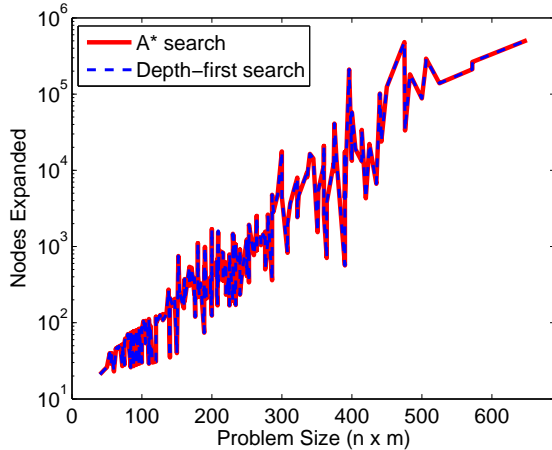


Figure 10. The number of nodes evaluated by the two search procedures when starting with a good heuristic solution.

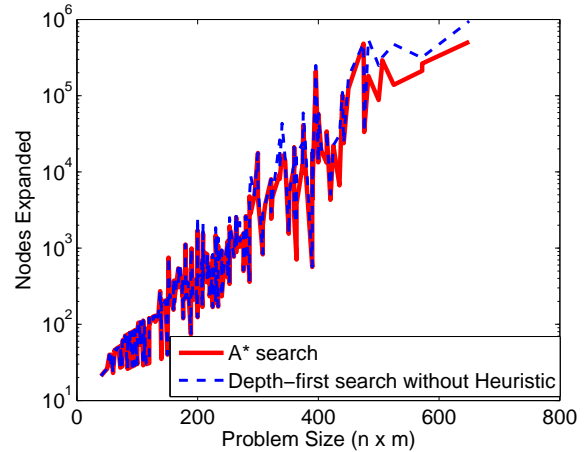


Figure 11. The number of nodes evaluated by the two search procedures when starting with no upper bound.

7.3 Gating

When running a branch and bound to optimality the gating procedure described in Section 4.5 can reduce the number of node evaluations. Figure 12 shows this for the 1000 runs from the artificial test scenario. While the difference may appear superficial, the small nearly constant shift on the log scale actually represents savings of around 40-50% of the lower bound computations. The time required to solve a problem was reduced by a similar amount. Also, a partial branch and bound or A*-search algorithm would benefit from such gating procedures. The gating procedure could be implemented in a more efficient manner than used for our initial testing in order to produce more savings.

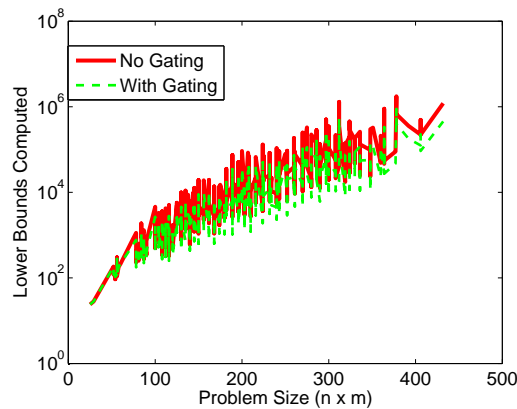


Figure 12. Lower bound evaluations with and without the gating procedure.

7.4 A Comparison of Bias Estimation and No Bias Estimation

When biases are significant compared to the target spacing, it is very difficult to get correct associations without estimating the bias. While the optimal solution to the joint MAP bias estimation and data association formulation may not perform perfectly due to model mismatches and unreliable prior information, it has shown drastic improvements compared to methods that do not use bias estimation. Figure 13 shows the fraction of truth assignments averaged over the 1000 runs in the artificial scenario. The combination of the k-best heuristic and the “anytime” A*-branch found the optimal solution of the joint MAP bias estimation and data association problem 100% of the time; however, this agreed with truth 98.8% of the time. The reason for the difference is model mismatch as indicated above.

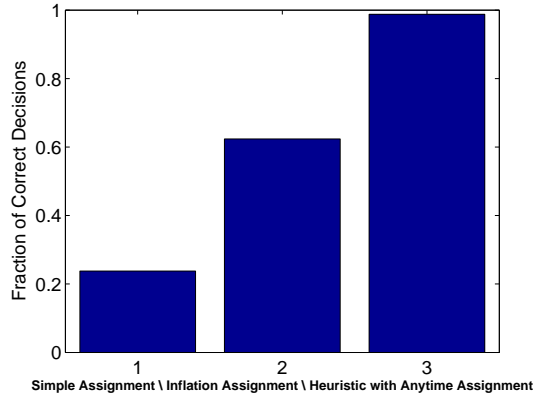


Figure 13. Fraction of algorithm assignments that match truth.

8. CONCLUSIONS

The philosophy behind the development of a class of algorithms for the joint MAP estimation of biases and data association in this work has been to combine a good initial heuristic with local search followed by a partial or full branch and bound as time allows. If the branch and bound procedure is stopped prior to completion, one can use the current best solution until time is available to complete the process.

We have explored two different heuristics in Section 3, namely, the “all-pairs” and the k-best heuristics. Computational experience shows that the “all-pairs” algorithm produces very good or near-optimal solutions, but its runtime is unsatisfactory for larger problems. The k-best heuristic on the other hand has been designed to be competitive with the quality of the “all-pairs” algorithm, but with a superior runtime performance. This heuristic also provides any branch and bound or A*-search algorithm with an exceptional initial upper bound that assists in the early pruning of nodes in the search tree that might otherwise need to be explored.

For the search procedure in the branch and bound algorithm, we have pursued A* and depth-first search procedures. A*-search is optimal in that it explores the smallest number of nodes among any search procedure that uses the same lower bound function. The very first leaf node that A* explores is an optimal solution. If A* is allowed to continue, then it will compute a guaranteed k-best bias-assignment pairs. The deficiency of A* is that the memory required to store nodes can grow exponentially in the size of the problem. Depth-first search has memory that grows linearly in the size of the problem; however, it can explore substantially more nodes than A*. If the initial upper bound is optimal, the two algorithms explore exactly same number of nodes. One would conjecture that if the initial upper bound is near optimal, then the number of nodes visited when using DFS with node ordering would be near that of A*-search. Indeed, the simulations in this paper support this conjecture. We have also demonstrated the enhancement provided by a gating procedure.

Based on the data and scenarios explored thus far, we can make the following recommendations on algorithms.

- With respect to heuristics, the k-best heuristic or “all-pairs” algorithm with local search and memory management is recommended for very small problems. As the problem size grows however, the k-best heuristic is favored based on superior runtime performance and similar solution quality.
- For an “anytime algorithm,” we recommend the k-best heuristic followed by A*-search algorithm. Gating improves efficiency and the use of the upper bound from the heuristic allows one to fathom nodes that might otherwise be placed in the queue. Additional efficiency might be achieved through use of parent information in solving the assignment problem and linear least squares problem in the lower bound. We note that one can generate an initial approximation to the k-best bias-assignment pairs from the heuristic and then improve the approximation with the partial A*-search algorithm. In this case, one would use as the upper bound the value of the initial k^{th} -solution from the heuristic.
- For an optimal solution and for the k-best solutions, the depth-first-search branch and bound algorithm performs exceptionally well in both runtime and memory usage provided that one makes use of the k-best heuristic. If memory requirements in A*-search were not an issue, then A*-search within the branch and bound procedure or as a stand-alone would be another recommended algorithm. There are many desirable properties of A*-search, including its performance as an anytime algorithm, that make it an overall favorite.

Acknowledgements. This work was supported by the Air Force Office of Scientific Research through AFOSR Contract Number FA9550-04-1-0222 and by the Office of Naval Research through ONR Contract Number N00014-05-C-0413.

REFERENCES

1. S. Dandford, B. D. Kragel, and A. B. Poore, “Joint map bias estimation and association: Algorithms,” in *Signal and Data Processing of Small Targets*, O. E. Drummond, ed., *SPIE* **6699**, 2007.
2. S. Blackman and R. Popoli, *Design and Analysis of Modern Tracking Systems*, Artech House, Norwood, MA, 1999.
3. M. Levedahl, “An explicit pattern matching assignment algorithm,” in *Signal and Data Processing of Small Targets*, O. E. Drummond, ed., *SPIE* **4728**, 2002.
4. M. Levedahl, “Method and system for assigning observations.” United States Patent US 7,092,924 B1, August 2006.
5. K. Murty, “An algorithm for ranking all the assignments in order of increasing cost,” *Operations Research* **16**, pp. 682–687, 1968.
6. S. Russell and P. Norvig, *Artificial Intelligence A Modern Approach*, Prentice Hall, Upper Saddle River, New Jersey 07458, second ed., 2003.