On Implementing 2D Rectangular Assignment Algorithms

DAVID F. CROUSE, Member, IEEE
Naval Research Laboratory
Washington, DC, USA

This paper reviews research into solving the two-dimensional (2D) rectangular assignment problem and combines the best methods to implement a *k*-best 2D rectangular assignment algorithm with bounded runtime. This paper condenses numerous results as an understanding of the "best" algorithm, a strong polynomial-time algorithm with a low polynomial order (a shortest augmenting path approach), would require assimilating information from many separate papers, each making a small contribution. 2D rectangular assignment Matlab code is provided.

Manuscript received December 21, 2014; revised August 28, 2015, December 30, 2015; released for publication March 14, 2016.

DOI. No. 10.1109/TAES.2016.140952.

Refereeing of this contribution was handled by S. Maskell.

This research is supported by the Office of Naval Research through the Naval Research Laboratory (NRL) Base Program.

Author's address: Naval Research Laboratory, Code 534, 4555 Overlook Ave., SW, Washington, DC 20375-5320. E-mail: (david.crouse@nrl.navy.mil).

I. INTRODUCTION

The two-dimensional (2D) assignment problem, also known as the linear sum assignment problem and the bipartite matching problem, arises in many contexts such as scheduling, handwriting recognition, and multitarget tracking, as discussed in [12]. Strong and weak polynomial time algorithms exist for solving the 2D assignment problem, unlike assignment problems involving more than two indices, such as the multiframe/S-dimensional assignment problem, which are NP complete [40, ch. 15.7].[1] The execution time of strong polynomial algorithms scales polynomially with the size of the problem; the execution time of weak polynomial algorithms scales polynomially with the size of the problem but also depends on values within the problem, in some cases allowing for very slow worst case execution time depending on particular values chosen. This paper considers the task of obtaining the best (lowest cost) and the *k*-best solutions to the 2D rectangular assignment problem.[2] Only strong polynomial time algorithms are given serious consideration as they are best suited for critical applications that cannot tolerate rare, very slow run times given certain inputs.

Given an $N_R \times N_C$ matrix $\mathbf{C}$ of costs (which might be positive, negative, or zero) with $N_C \geq N_R$, the 2D rectangular assignment problem consists of choosing one element in each row and at most one element in each column such that the sum of the chosen elements is minimized or maximized. For example, a hotel might want to assign rooms to clients based upon the price that the clients have bid to stay in each room. If there are more rooms than clients, then the clients are the rows and some rooms will remain unassigned; if there are more clients than rooms, then the rooms are the rows and some clients will not be able to stay in the hotel.

Expressed mathematically, the 2D rectangular assignment problem for minimization is

$$\mathbf{X}^* = \arg\min_{\mathbf{x}} \sum_{i=1}^{N_R} \sum_{j=1}^{N_C} c_{i,j} x_{i,j} \tag{1}$$

$$\text{subject to} \sum_{j=1}^{N_C} x_{i,j} = 1 \quad \forall i \qquad \text{Every row is assigned to a column.} \tag{2}$$

$$\sum_{i=1}^{N_R} x_{i,j} \leq 1 \quad \forall j \qquad \text{Not every column is assigned to a row.} \tag{3}$$

$$x_{i,j} \in \{0, 1\} \quad \forall x_{i,j} \qquad \text{Equivalent to } x_{i,j} \geq 0 \quad \forall x_{i,j}, \tag{4}$$

[1] The relationship between the complexity classes *P* and *NP* is a major unsolved problem in theoretical computer science for which a million dollar prize is offered [17]. Many believe that $P \neq NP$, though no proof exists to date.

[2] This paper is an extension of the second half of the conference publication [22].

where min is replaced by max if one wishes to maximize the cost function, $c_{i,j}$ is the element in row i and column j of the cost matrix $\mathbf{C}$, and the matrix $\mathbf{X}$ is the set of all of the $x_{i,j}$. If $x_{i,j} = 1$, then the item in row i is assigned to the item in column j. Implicitly, the cost of not assigning a column to a row is zero.

In (4), it is indicated that the binary constraint on the $x_{i,j}$ terms can be replaced by a nonnegativity constraint. This substitution is acceptable as it has been proven that such a substitution does not change the optimal value of the 2D optimization problem [11, ch. 7.3, 7.8]. The optimal $\mathbf{X}$ satisfying all the constraints will still be such that all elements are binary. The substitution of inequality constraints for integer constraints is possible on families of optimization problems that are considered unimodular [7, ch. 5.5.1]. This substitution turns the 2D assignment problem into a linear programming problem.

This paper focuses on solving the problem in (1) when the cost of the globally optimal solution is finite. The solution is derived for the minimization problem where all of the costs in the matrix are positive, because any minimization problem with finite negative costs can be transformed to have all positive costs and any maximization problem can be transformed into an equivalent minimization problem. Specifically, if one wishes to perform minimization on a matrix $\tilde{\mathbf{C}}$ (where an element in the ith row and jth column is $\tilde{c}_{i,j}$) that might have negative elements, one can transform the matrix into a usable cost function $\mathbf{C}$ as

$$\mathbf{C} = \tilde{\mathbf{C}} - \min_{i,j} \tilde{c}_{i,j}. \tag{5}$$

The requirement that the globally optimal solution have a finite cost ensures that the term $\min_{i,j} \tilde{c}_{i,j} > -\infty$. However, the transformation in (5) does not preclude the use of certain $\tilde{c}_{i,j}$ values being set to ∞ to forbid certain assignments. Detecting whether a problem with positive, infinite costs is feasible (whether any finite-cost solution exists) will be subsequently discussed and is essential to implementing an efficient algorithm for finding the k-best assignments. Similarly, the problem of maximizing the cost of assignments on a matrix $\tilde{\mathbf{C}}$ can be transformed into a problem of minimizing the cost of assignments with the matrix

$$\mathbf{C} = -\tilde{\mathbf{C}} + \max_{i,j} \tilde{c}_{i,j}, \tag{6}$$

under the assumption that none of the elements of $\tilde{\mathbf{C}}$ is ∞, though elements of $\tilde{\mathbf{C}}$ are allowed to be $-\infty$ to forbid certain assignments.

Section II describes how the 2D rectangular assignment algorithm can be solved in polynomial time with an upper bound on the total number of instructions necessary to run to completion and how such an algorithm can be implemented to quickly determine whether or not a cost matrix presents a feasible solution. Simulation examples demonstrating the runtime of the algorithm are also given. Section III then describes how a k-best 2D assignment algorithm can be implemented using the 2D rectangular assignment algorithm of Section II, where simulation examples are also presented. The results are summarized in Section IV. To facilitate the understanding and use of the algorithms described here, the Matlab code implementing the 2D rectangular shortest augmenting path algorithm is given in the Appendix.

II. A 2D RECTANGULAR ASSIGNMENT ALGORITHM WITH FEASIBILITY DETECTION

A. Problem Formulation and Background

One of the most frequently used techniques for solving the linear sum assignment problem is the auction algorithm, described in its basic form in [4, 11, ch. 7.8]. The basic form of the auction algorithm assumes that $N_R = N_C$, that is, that all items represented by rows must be assigned to all items represented by columns, which limits the scope of problems that can be solved by the algorithm. Generalizations of the basic auction algorithm are discussed in [5, 8, 9, 42]. However, the auction algorithm is not always the best solution. All formulations of the auction algorithm are weakly polynomial time algorithms. That means that the worst case computational complexity of the algorithms depends not only on the size of the problem (in this case on N_R and N_C), but also on the relative values of the elements of $\mathbf{C}$. Given an appropriately degenerate $\mathbf{C}$ matrix, if one wants to be guaranteed the globally optimal solution, then an upper bound on the execution time of the algorithm can become arbitrarily long. Versions of the auction algorithm utilizing ϵ-scaling have the lowest theoretical bound, which does not always translate into fast execution times in practice [6, ch. 7.1.4], [10, ch. 5.4], as demonstrated in Subsection II-D.[3]

On the other hand, a number of other 2D assignment algorithms exist. Many of these have strong polynomial complexity; their worst case execution time scales polynomially dependent only on the dimensions N_R and N_C and not on the actual values of the elements of $\mathbf{C}$. The first such algorithm is often referred to as the "Hungarian algorithm"[4] and is described in [14, ch. 4.2], among many other places. When considering a square cost matrix, $N_R = N_C = N$, the Hungarian algorithm has a complexity of $O(N^4)$. Many of the most efficient 2D assignment algorithms tend to be variants of the Hungarian algorithm. For example, the algorithm of Jonker and Volgenant [32], which unbeknownst to many can be considered a particularly efficient variant of the Hungarian algorithm

[3] It is not unusual for an algorithm with a low worst case bound to have poor average performance. For example, when considering linear programming, the popular simplex algorithm has an exponential worst case complexity [11, ch. 3.7], whereas the ellipsoid method is weakly polynomial in complexity [11, ch. 3.7]. However, the simplex method is generally much faster than the ellipsoid method [11, ch. 3.7].

[4] The algorithm was first named the "Hungarian" algorithm by Kuhn [34], who based the approach on work done by Jenő Egerváry that was published in Hungarian.

[14, ch. 4.4], has a complexity of $O(N^3)$ [32]. A variant of the Jonker-Volgenant algorithm that has been generalized to rectangular cost matrices is often called the JVC algorithm, with the C standing for Castañon, who provided a generalized implementation of the algorithm to work with rectangular matrices and replaced the original initialization step with a few iterations of the auction algorithm, which is faster [27].[5]

A number of studies have been conducted comparing algorithms for 2D assignment in numerous applications, such as multiple target tracking. In [41], three 2D assignment algorithms are compared considering their performance in multiframe optimization, namely, the auction algorithm, the RELAX II algorithm and the generalized signature method, with the auction algorithm performing the best. In [27], the JVC algorithm is compared with three variants of the Munkres algorithm [38],[6] with the JVC algorithm performing the best. In [15], the JVC algorithm is compared with variants of the auction algorithm, with a scaled forward-reverse auction algorithm performing the best. Other studies have concluded that the JVC algorithm is often a better alternative to the auction algorithm. In [33], the Munkres, JVC, deepest hole,[7] and auction algorithms are compared based on a measurement assignment problem for multiple target tracking, where the JVC algorithm is found to be the best overall, with the auction algorithm being faster on sparse problems. In the assignment problem, "sparse" means that many elements of $\mathbf{C}$ are not finite (certain rows cannot be assigned to certain columns).

However, it is noted that the auction algorithm in the simulations in [33] does not always produce optimal results. Such suboptimal solutions can arise when the ϵ parameter in the auction algorithm is not small enough. The auction algorithm is a type ϵ relaxation dual optimization technique [7, ch. 6.3.4]. When considering $N_R = N_C = N$, the accuracy of the cost function is within ϵN of the optimal value in both the forward and reverse [9] versions of the algorithm. Though papers on the auction algorithm generally consider setting ϵ in view of integer values of $c_{i,j}$, nothing in the proof [4] of that accuracy bound requires the costs to be integers. Thus, to ensure convergence to an optimal solution ϵN should be less than the minimum nonzero difference between all pairs of elements in $\mathbf{C}$. However, as ϵ decreases, the worst case computational complexity of the auction algorithm increases [4].

In [36], it is noted that the JVC algorithm had been implemented in a subsystem used in a (at that time) next generation helicopter for the U.S. Army. The JVC algorithm is compared with an unpublished, proprietary, heuristic algorithm called "competition" and is shown to be faster. However, the competition algorithm is shown to have fewer assignment errors. Note, however, that Jonker and Volgenant's algorithm [32] is guaranteed to converge to the globally optimal solution at each time-step. The source of the suboptimal convergence of the JVC algorithm used in [36] probably comes from the fact that the auction algorithm was used in an initialization step. The auction algorithm does not strictly guarantee complementary slackness (which shall subsequently be defined), as the Jonker-Volgenant algorithm requires, but only does so within a factor of ϵ. Thus, the accelerated initialization provided with the code in [27], which uses a fixed, heuristic value of ϵ for all problems, is probably the cause of the suboptimal results. Such problems will be avoided in the 2D assignment algorithm presented in this section.

In [35], the JVC algorithm, the auction algorithm, the Munkres algorithm, and a suboptimal greedy approach are compared in a tracking scenario. The Munkres algorithm is found to be significantly slower than the other methods, agreeing with the study done in [44] that found the auction algorithm to be faster than the Munkres algorithm. The JVC algorithm is found to be fast enough to negate any speed benefit from using a suboptimal greedy technique. In [28], it is also concluded that the speed of the JVC algorithm negated the need for greedy approximations.

In [35], the JVC algorithm is shown to be faster than the auction algorithm in all instances when implemented in C, and on dense problems (mostly finite elements in $\mathbf{C}$) when implemented in Matlab. At first, this seems to contradict the results of [33], which deemed the auction algorithm superior on sparse problems. However, the focus of [33] was on sparse problems for target tracking applications, which at the time were considered difficult, because one did not always include missed detection hypotheses in the hypothesis matrix $\mathbf{C}$. In such an instance, if the only thing that two targets could be assigned to was a single, common measurement, no feasible assignment would be possible and 2D assignment algorithms would fail.

Eight different assignment algorithms are compared with $N_R = N_C$ in [14, ch. 4.10]. Variants of the Jonker-Volgenant algorithm are shown to perform the best on the majority of dense problems, and are competitive on sparse problems (when many elements of $\mathbf{C}$ are not finite, representing forbidden assignments). However, the Jonker-Volgenant algorithm variants are beaten on the most difficult problem by the cost scaling implementation [30] of the algorithm of push-relabel algorithm [18, ch. 26.4], which performs poorly on one of the sparse problems. The cost scaling algorithm of [30] is an ϵ-scaling technique like that used in the auction algorithm. This paper avoids such algorithms as the choice of ϵ is

[5] In other words, the shortest augmenting path algorithm of Jonker and Volgenant was kick-started by a form of the auction algorithm. Jonker and Volgenant's algorithm does not actually have to be initialized, though that can speed it up.

[6] The Munkres algorithm is an old $O(N^4)$ version of the Hungarian algorithm.

[7] The deepest hole algorithm is suboptimal. Given the speed of optimal 2D assignment algorithms, there is seldom need to use a suboptimal approach now.

problem dependent and an algorithm that provides globally optimal solutions given noninteger costs in $\mathbf{C}$ without "tweaking" is desired. However, it has been noted that good implementations of the auction algorithm tend to be relatively insensitive to the range of costs in $\mathbf{C}$ [48]. In writing this paper, it was empirically observed that large magic squares[8] tend to make particularly bad cost matrices, causing poor implementations of the auction algorithm to grind to a halt. Magic squares can be generated in Matlab using the magic command.

In general, the JVC algorithm has come to be considered good at solving assignment problems under most conditions. The only real area of contention is with sparse problems and with extremely large problems, where some authors have considered adaptively choosing the assignment technique (JVC or auction) based upon the sparsity of the problem [43]. As sparse problems are naturally faster than dense problems, such adaptive algorithmic switching will not be considered in this paper, with the focus being placed on the worst case (dense) scenario. When considering very large problems, the algorithm with the lowest strong polynomial computational complexity on sparse problems is the primal simplex method in [1], which makes use of dynamic trees [45] and Fibonacci heaps [18, ch. 20] to speed up the implementation. However, the algorithm can be difficult to implement, and its polynomial complexity on large, dense problems is no better than the Jonker-Volgenant algorithm.

The following subsection develops a modified version of the Jonker-Volgenant algorithm that can handle the case where $N_R \leq N_C$, and that can detect when the assignment problem is infeasible, that is, when it is not possible to assign every row to a column keeping the cost function finite. Though an initialization stage could potentially speed up the algorithm, one was omitted both for brevity and because it is not necessary. The omission of an initialization stage based on the auction algorithm avoids many of the pitfalls of other implementations and guarantees convergence to a globally optimal solution. Despite the omission of an initialization step, the algorithm still executes quickly on the simulations in this paper in Subsection II-C.

Additionally, omitting the initialization step allows for worst case execution times to be obtained, letting one determine whether the algorithm can be *g*uaranteed to run in real-time on problems of certain sizes. The algorithm in the following subsection was designed with its use in a larger approach that obtains not just the best hypothesis, but the k-best hypotheses. The generalization to the k-best hypotheses is presented in Section III. The original version of the Jonker-Volgenant algorithm assumes that $N_R = N_C$. As such, some adaptations of the algorithm to the case where $N_R \leq N_C$ add "dummy rows" to make the cost matrix $\mathbf{C}$ square again. Though dummy rows prove necessary in Section III when finding the k-best associations, they are not necessary when finding only the most likely hypothesis.

[8] An $n \times n$ magic square is a matrix containing the positive integers from 1 through n^2 such that all row sums, column sums and main diagonal sums are equal [50].

B. A Modified Jonker-Volgenant Algorithm

The Jonker-Volgenant algorithm [32] uses a shortest augmenting path approach to perform dual-primal optimization to solve the assignment problem. Jonker and Volgenant's paper describes a solution to the assignment problem when $N_R = N_C$, meaning that the inequality constraint in (3) becomes an equality constraint. In other words, each target must be assigned to an event and each event can only be assigned to one target. Here, the more general case where $N_R \leq N_C$ is considered.

Let $\mathbf{x}$ be the vector obtained by stacking the columns of $\mathbf{X}$. That is

$$\mathbf{x} = [x_{1,1}, x_{2,1}, \ldots, x_{N_R,1}, x_{1,2}, x_{2,2}, \ldots, x_{N_R,2}, x_{1,3}, \ldots, x_{N_R,N_C}]'. \tag{7}$$

Written in the more traditional vector notation used in linear programming literature, the optimization problem can be stated as

$$\hat{\mathbf{x}} = \arg\min_{\mathbf{x}} \ \mathbf{c}'\mathbf{x} \tag{8}$$

$$\text{subject to } \mathbf{A}\mathbf{x} = \mathbf{1} \tag{9}$$

$$\mathbf{B}\mathbf{x} \leq \mathbf{1} \tag{10}$$

$$\mathbf{x} \geq \mathbf{0} \tag{11}$$

where $\mathbf{c}$ is the matrix $\mathbf{C}$ with all of its columns stacked into a single row-vector. The matrices $\mathbf{A}$ and $\mathbf{B}$ are such that (9) and (10), respectively, represent the constraints in (2) and (3). That is,

$$\mathbf{A} = [\mathbf{I}_{N_R \times N_R} \quad \mathbf{I}_{N_R \times N_R} \quad \ldots \quad \mathbf{I}_{N_R \times N_R}] \tag{12}$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{1}_{1\times N_R} & \mathbf{0}_{1\times N_R} & \ldots & \mathbf{0}_{1\times N_R} \\ \mathbf{0}_{1\times N_R} & \mathbf{1}_{1\times N_R} & \ldots & \mathbf{0}_{1\times N_R} \\ \vdots & \vdots & \ddots & \mathbf{0}_{1\times N_R} \\ \mathbf{0}_{1\times N_R} & \mathbf{0}_{1\times N_R} & \ldots & \mathbf{1}_{1\times N_R} \end{bmatrix} \tag{13}$$

where $\mathbf{I}_{N_R \times N_R}$ $\mathbf{1}_{1\times N_R}$ and $\mathbf{0}_{1\times N_R}$ are, respectively, the identity matrix and matrices of ones and zeros, all having the dimensionalities given by their respective subscripts. The $\mathbf{A}$ is $N_R \times N_R N_C$ dimensional and $\mathbf{B}$ is $N_C \times N_R N_C$ dimensional. The (unknown) optimal solution to the optimization problem in (8) will be designated as $\mathbf{x}^*$. Using this notation, the basic concepts behind dual optimization, which underly the Jonker-Volgenant algorithm are discussed.

The aforementioned linear programming problem is known as the primal problem. However, handling equality and inequality constraints is difficult, so a dual problem

will be formulated. The function

$$g(\mathbf{u},\mathbf{v}) = \min_{\mathbf{x},\mathbf{x}\geq\mathbf{0}} \left[\sum_{i=1}^{N_R}\sum_{j=1}^{N_C} c_{i,j}x_{i,j} + \sum_{i=1}^{N_R} u_i \left(1 - \sum_{j=1}^{N_C} x_{i,j}\right) + \sum_{j=1}^{N_C} v_j \left(1 - \sum_{i=1}^{N_R} x_{i,j}\right)\right] \quad (14a)$$

$$= \min_{\mathbf{x},\mathbf{x}\geq\mathbf{0}} \left[\mathbf{c}'\mathbf{x} + \mathbf{u}'(\mathbf{1} - \mathbf{A}\mathbf{x}) + \mathbf{v}'(\mathbf{1} - \mathbf{B}\mathbf{x})\right] \quad (14b)$$

$$= \min_{\mathbf{x},\mathbf{x}\geq\mathbf{0}} \left[\left(\mathbf{c} - \mathbf{A}'\mathbf{u} - \mathbf{B}'\mathbf{v}\right)'\mathbf{x}\right] + \mathbf{u}'\mathbf{1} + \mathbf{v}'\mathbf{1} \quad (14c)$$

is known as the dual cost function. The **u** and **v** variables are known as Lagrange multipliers or dual variables and their use in eliminating constraints in optimization problems is known as Lagrangian relaxation [7, ch. 3]. There is one Lagrange multiplier variable per constraint that has been eliminated, except for the nonnegativity constraint on **x**, which will not be relaxed. Due to the extra degrees of freedom introduced by the dual variables, under the constraint that $\mathbf{v} \leq \mathbf{0}$, it can be shown that $g(\mathbf{u}, \mathbf{v}) \geq \mathbf{c}'\mathbf{x}^*$ [11, ch. 4.1]. In other words, the dual cost function forms a lower bound on the value of the primal cost function at the globally optimal solution. The dual optimization problem seeks to find the values of **u** and **v** that maximize this lower bound.

To formulate the dual optimization problem note that

$$\left(\mathbf{c} - \mathbf{A}'\mathbf{u} - \mathbf{B}'\mathbf{v}\right)'\mathbf{x} = \sum_{i=1}^{N_R}\sum_{j=1}^{N_C} \left(c_{i,j} - u_i - v_j\right)x_{i,j} \quad (15)$$

Given that the binary constraint on **x** has been relaxed to a nonnegativity constraint, if $c_{i,j} - u_i - v_j < 0$, then $x_{i,j}$ can be chosen arbitrarily large so that $g(\mathbf{u}, \mathbf{v})$ is arbitrarily small. Since the dual optimization problem concerns maximizing the dual cost function, it makes sense to only consider solutions greater than $-\infty$ by introducing the constraint that $c_{i,j} - u_i - v_j \geq 0$, or expressed in vector form that $\mathbf{c} - \mathbf{A}'\mathbf{u} - \mathbf{B}'\mathbf{v} \geq \mathbf{0}$. This constraint is known as a complementary slackness condition [11, ch. 4.3]. However, if $c_{i,j} - u_i - v_j > 0$, then the value of $x_{i,j}$ that minimizes (15) is 0, implying that the globally minimum value of (15) is always zero if $c_{i,j} - u_i - v_j \geq 0$. Put differently, given the complementary slackness condition, the following equation is true,

$$\min_{\mathbf{x}:\mathbf{x}\geq\mathbf{0}} \left(\mathbf{c} - \mathbf{A}'\mathbf{u} - \mathbf{B}'\mathbf{v}\right)'\mathbf{x} = \mathbf{0}. \quad (16)$$

Substituting this into the dual cost function of (14c), the dual optimization problem is

$$\{\mathbf{u}^*, \mathbf{v}^*\} = \arg\max_{\mathbf{u},\mathbf{v}} \ \mathbf{u}'\mathbf{1} + \mathbf{v}'\mathbf{1} \quad (17)$$

$$\text{subject to } \mathbf{v} \leq \mathbf{0} \quad (18)$$

$$\mathbf{c} - \mathbf{A}'\mathbf{u} - \mathbf{B}'\mathbf{v} \geq \mathbf{0} \qquad \text{Complementary Slackness} \quad (19)$$

Because the binary constraint on **x** in the primal problem was replaced by a vector nonnegativity (inequality) constraint, the primal problem is a linear programming problem. For linear programming problems, the strong duality theorem says that the duality gap, that is, the expression $\mathbf{c}'\mathbf{x}^* - g(\mathbf{u}^*, \mathbf{v}^*)$, is zero [11, ch. 4.3]. Thus, solving the dual problem provides the value of $\mathbf{c}'\mathbf{x}^*$, but does not directly provide the value of $\mathbf{x}^*$. Given (16), one can say that $x_{i,j} = 0$ if $c_{i,j} - u_i - v_j > 0$. However, this does not explicitly say which values of **x** should be one. It could be possible that multiple values of **x**, not all of which satisfy the constraints of the original problem, yield the same cost.

Consider, first, the case where $N_R = N_C$, meaning that the inequality constraint in (3) becomes an equality constraint. In this case, given the optimal dual solution, valid solutions for **x** are those satisfying the equality constraints. The satisfaction of the constraints means that the terms in (14b) involving dual variables disappear so that the dual and primal costs are equal, as expected. On the other hand, if $N_R \leq N_C$, then by the strong duality theorem, it is known that the dual and primal cost functions should be equal at the globally optimal values $\mathbf{u}^*$, $\mathbf{v}^*$, and $\mathbf{x}^*$. To eliminate the terms in (14b) involving **v**, it is necessary that

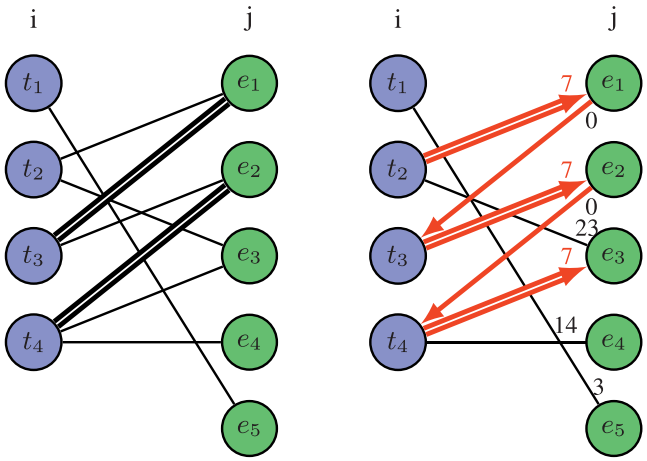
$$\mathbf{v}'(\mathbf{1} - \mathbf{B}\mathbf{x}) = \mathbf{0} \quad (20)$$

Expressed in scalar form, this says that

$$v_j \left(1 - \sum_{i=1}^{N_R} x_{i,j}\right) = 0 \quad \forall j \quad (21)$$

This requirement is also known as another complementary slackness condition [11, ch. 4.3], [7, ch. 3.3]. The complementary slackness theorem [11, ch. 4.3] springs logically from this. The complementary slackness theorem says that given vectors **u**, **v**, and **x** such that (16) and (20) hold, then **u**, **v,** and **x** are optimal solutions to both the primal as well as the dual optimization problems.

The complementary slackness theorem plays an important role in algorithms such as the Jonker-Volgenant algorithm that use shortest augmenting path techniques for solving the assignment problem, as well as in more general augmenting path methods, such as the Ford-Fulkerson and Edmonds-Karp algorithms [18, ch. 26.2], for use in general network optimization problems. Such algorithms solve the assignment problem by sequentially solving a series of assignment problems with $N_R = 1$, $N_R = 2$, et cetera. The complementary slackness theorem is used to verify that the globally optimal solution to each subproblem is obtained. Most presentations of shortest augmenting path algorithms, such as in [26, 46] and [14, ch. 4.4], relate them to minimum cost network flow problems from which the shortest path algorithm can be derived. Others, such as [24], simply provide the algorithm and then show that the complementary slackness conditions hold after each step. Due to the complexity of the minimum cost network flow problem, a direct derivation directly along those lines will be avoided.



(a) A Partial Assignment

(b) A Minimum Cost Alternating Path

Fig. 1. (a) The bipartite graph corresponding to regular and reduced cost matrices in (22) and (23). Partial assignment of two rows is marked with bold lines. Nodes for rows are on left and columns are on right. In absence of rows t_1 through t_3, partial assignment is globally optimal. (b) Minimum cost alternating path starting from node t_2 when using reduced costs of (23). Only forward arcs contribute to cost, which is 21; all reverse arcs have reduced costs of 0. Other alternating paths starting at t_2 are $t_2 \rightarrow e_3$, which has reduced cost of 23, and $t_2 \rightarrow e_1 \rightarrow t_3 \rightarrow e_2 \rightarrow t_4 \rightarrow e_r$, which has reduced cost of 21. Forward arcs form new, larger assignment.

Shortest augmenting path algorithms for solving the 2D assignment problem can be decomposed into a few basic steps:

1) Initialize.
2) Find the shortest augmenting path.
3) Update the dual variables to assure complementary slackness.
4) Augment the previous solution with the shortest path.
5) If all rows have been assigned, then the problem is solved. Otherwise, go to step 2.

To understand such algorithms, the notion of an augmenting path must be defined. The concept comes from graph theory, and is easiest to explain with an example. Consider the following square cost matrix:

$$\mathbf{C} = \begin{bmatrix} \infty & \infty & \infty & \infty & 3 \\ 7 & \infty & 23 & \infty & \infty \\ 17 & 24 & \infty & \infty & \infty \\ \infty & 6 & 13 & 20 & \infty \end{bmatrix} \tag{22}$$

The infinite entries represent forbidden assignments. Fig. 1(a) shows a graph representation of the structure of the cost matrix. The left-hand nodes in Fig. 1(a) represent the rows in the assignment matrix. The right-hand nodes represent the columns. A line is drawn between a node and a column if the cost of the association in $\mathbf{C}$ is finite. Illustrated in bold in the figure is a partial assignment: $t_3 \rightarrow e_1$ and $t_4 \rightarrow e_2$, meaning that the third and fourth rows are assigned to the first and second columns, respectively. In the Jonker-Volgenant algorithm, all partial assignments that are formed have the minimum cost for all of the rows included in the assignment, which need not be the same assignment present once all rows are assigned.

An alternating path is a path that starts at an unassigned row, in this case t_1 or t_2, and ends at an unassigned column. In between, the path must alternate between assigned rows and columns. In Fig. 1(a), there are four possible alternating paths. The first is $t_1 \rightarrow e_5$, which illustrates that an alternating path can go directly between an unassigned row and column without visiting any assigned nodes. The second is $t_2 \rightarrow e_1 \rightarrow t_3 \rightarrow e_2 \rightarrow t_4 \rightarrow e_3$, which begins at the unassigned node t_2 and alternates back and forth from assigned rows and columns until reaching e_3, which is unassigned. The fourth possible alternating path is the same as the third, except it ends at e_4, which is unassigned. The fifth possible alternating path is $t_2 \rightarrow e_3$.

A maximum cardinality bipartite match is the largest possible assignment of rows to columns such that no two rows are assigned to the same column and no two columns are assigned to the same row. It is considered regardless of the cost of the assignment. As discussed in [18, ch. 26.3] and [24], alternating paths play a role in determining the maximum bipartite match. In the rectangular assignment problem, the maximum bipartite match should be such that all rows are assigned. However, when finding the k-best hypotheses, it is possible that infeasible problems might be presented. Given any partial assignment, such as that shown in Fig. 1(a), a larger assignment can be achieved by finding any alternating path, and then augmenting with that path. Augmentation means that all parts of the path that go from rows to columns are the new assignments, any parts of the path that go from columns to rows are unassigned, and any assignments that do not overlap with the path remain unchanged. If no augmenting path can be found, then the maximum possible assignment of rows to columns has been found [24], [18, ch. 26.3]. If not all rows are assigned, then that the assignment problem under consideration is infeasible.

Shortest augmenting path algorithms, such as the Jonker-Volgenant algorithm, solve the assignment problem by finding a series of minimum cost augmenting paths to iteratively increase the number of assigned objects. However, what is less intuitive is that dual variables must be maintained and the cost matrix must be modified using the dual variables after each assignment to ensure that a globally optimal solution is ultimately obtained. In the example at hand, for the given partial assignment illustrated in Fig. 1(a), the reduced cost matrix is

$$\bar{\mathbf{C}} = \begin{bmatrix} \infty & \infty & \infty & \infty & 3 \\ 7 & \infty & 23 & \infty & \infty \\ 0 & 7 & \infty & \infty & \infty \\ \infty & 0 & 7 & 14 & \infty \end{bmatrix} \tag{23}$$

How the costs in the matrix were adjusted using dual variables $\mathbf{u}$ and $\mathbf{v}$ will be discussed shortly. One thing to note is that the zero entries in the matrix correspond to the chosen assignments. Using this reduced cost matrix, the

shortest augmenting path in Fig. 1(b) starting from node t_2 is highlighted, and the new assignment, which is given by the parts of the path going from rows to columns, is illustrated.

In this example, it was decided that the shortest augmenting path starting at row t_2 would be found. However, the shortest augmenting path among all possible augmenting paths is $t_1 \rightarrow e_5$, which has a cost of 3. If one were to augment using that path, one would get new assignments of $t_3 \rightarrow e_1$, $t_4 \rightarrow e_2$, and $t_1 \rightarrow e_5$. Because that path did not overlap with any already existing assignments, those assignments would remain unchanged. To solve the complete assignment problem, however, it does not matter whether a shortest augmenting path starting from t_1 or t_2 is found, because the algorithm provides the minimum cost assignment for the rows that have been chosen to be assigned [26], [14, ch. 4.4.2], as shall be elucidated when the dual update step is discussed. An important aspect of this realization is that if an augmenting path to add a given unassigned target to the partial assignment cannot be found, then the assignment problem is infeasible.[9] The rapid identification of infeasible assignment problems is useful in efficiently implementing techniques that find the k-best assignments and is often overlooked in the literature.

The simplest way to show that augmenting with the shortest augmenting path algorithm produces an optimal assignment for the subproblem involving only the rows that one has chosen to add to the problem is to present the algorithm with its dual update step and show that the result satisfies the complementary slackness conditions for the dual problem that were previously mentioned. The procedure for finding the shortest augmenting path (given a partial assignment) that is commonly used in the assignment problem is the Dijkstra algorithm [25], [18, ch. 24.3], [11, ch. 7.9]. A particularly clear description of the Dijkstra algorithm is given in [11, ch. 7.9]. A modified version of the algorithm that properly utilizes and updates the dual variables for calculating the reduced costs is presented in [14, ch. 4.4], [32] and is given as steps 2–4 of a complete rectangular version of the 2D Jonker-Volgenant algorithm as follows.

1) **Initialize.** Initialize the $N_R \times 1$ vector **u**, the dual cost vector for the rows, and the $N_C \times 1$ vector **v**, the dual cost vector for the columns, to all zeros. The scalar value *cur Row* will be the index of the current unassigned row that is to be assigned. Set *cur Row* $= 0$ to select the first row to assign (indexation is assumed to start from 0). The $N_R \times 1$ vector **col4row** and the $N_C \times 1$ vector **row4col** will hold the values of the assigned indices. As nothing has been assigned yet, set all of their elements to –1. The set *AC* will be the set of all columns. Allocate $N_C \times 1$ space for a vector called **path**, which will specify which row is associated with which column in the minimum cost path.

2) **Prepare for Augmentation**. Set all elements of the $N_C \times 1$ vector **shortestPathCosts** equal to infinity. Set *SR* and *SC* equal to the empty sets. They will hold the row and column vertices, respectively, that have been reached by a shortest path emanating from row k. Set *sink* $= -1$, *minVal* $= 0$ and the current row to be assigned is set to $i =$ *curRow*. The variable *minVal* will ultimately hold the cost of the shortest augmenting path that is found, and *sink* will ultimately be the index of the final column in the alternating path. The variable j will select the current column.

3) **Find the Shortest Augmenting Path.**

while sink = -1 **do**
  SR ← *SR* ∪ {*i*};
  for all *j* ∈ *AC**SC* such that *minVal* + **C** [*i,j*] – **u**[*i*] – **v**[*j*] < **shortestPathCosts** [*j*] **do**
    path [*j*]← *i*;
    shortestPathCosts [*j*]← *minVal* + **C** [*i,j*]– **u** [*i*]– **v** [*j*];
  end for
  j ← arg min (**shortestPathCosts**[*h*] given *h* ∈ *AC**SC*); (If **shortestPathCosts**[*j*] = ∞, then infeasible!)
  SC← *SC* ∪ *j*;
  minVal ← **shortestPathCosts**[*j*]
  if **row4col** [*j*] = –1 **then**
    sink ← *j*;
  else
    i ← **row4col** [*j*];
  end if
end while

The ∪ operation means that the two sets are being merged. Thus, *SR* ← *SR* ∪ {*i*} means that row *i* is being added to the collection of rows that have been visited. A backslash means that the right-hand quantity is subtracted from the set. Vectors are indexed using brackets. The shortest augmenting path ends at column *sink*. The first row in the path is thus, $r =$ **path**[*sink*]. The next column is given by **col4row**[r]. The following row is then **path**[r] and so on. The path is traced until row *curRow*, which began the path, is reached. After updating the dual variables, the assignments from the shortest augmenting path step must be saved for the next iteration.

4) **Update the Dual Variables**.

  u [*cur Row*] ← **u** [*cur Row*] + *minVal*;
  for all *i* ∈ *SR* *curRow* **do**
    u [*i*] ← **u**[*i*] + *minVal* – **shortestPathCosts** [**col4row** [*i*]];
  end for
  for all *j* ∈ *SC* **do**
    v [*j*] ←**v** [*j*] –*minVal* + **shortestPathCosts** [*j*];
  end for

5) **Augment the Previous Solution.**

  j← *sink;*
  do
    i ← **path** [*j*];

[9] When considering infeasible assignments as having infinite cost, an augmenting path can always be found, but it might have infinite cost. Since adding more rows (targets) to the problem can only increase the cost of the assignment problem, the total cost will always be infinite, so the algorithm can be stopped once it find a single infeasible (infinite cost) path.

row4col $[j] \leftarrow i$;
$temp \leftarrow$ **col4row** $[i]$;
col4row $[i] \leftarrow j$;
$j \leftarrow temp$;
while $i \neq curRow$
6) **Loop**
$curRow \leftarrow curRow + 1$;
if $curRow = N_R$ **then**
Exit.
else
Go to step 2.
end if

By the nature of Dijkstra's shortest augmenting path algorithm, assuming that all entries in $\mathbf{C}$ are nonnegative, step 3 finds the minimum cost augmenting path for the reduced cost matrix $\overline{\mathbf{C}}$ such that $\bar{c}_{i,j} = c_{i,j} - u_i - v_j$ [11, ch. 7.9]. Equation (23) is the reduced cost matrix for (22) after the bottom two rows have been assigned. By the nature of augmenting path techniques for maximum bipartite matching, the association obtained by augmenting with the path that was found is feasible [18, ch. 26.3]. Additionally, it was proven in [14, ch. 4.4] that the resulting dual solution satisfies the complementary slackness condition in (19), and assures that $c_{i,j} - u_i - v_j \geq 0$. The satisfaction of the complementary slackness condition is why the reduced costs in (23) have zeros in the entries for the partial assignment. After updating the dual solution, zeros are placed in the locations of the new assignment by step 4. Note that the update also does not change the optimal solution to the subproblem (considering only the rows that have been assigned), meaning that the solution is optimal for the reduced primal. Thus, if $N_R = N_C$, once all rows have been assigned, the algorithm will have terminated with a solution that, based upon the duality theorem, is optimal. Another proof of the optimality of the solution is given in [26].

The assumption that $N_R = N_C$ is important for assuring optimality of the solution, because it means that the complementary slackness constraint of (20), which pertains to the inequality constraints, need not be proven to have been fulfilled to assure optimality. What is of interest here, however, is the case where $N_R \leq N_C$. Traditional approaches to the problem, such as that in [37] and [14, ch. 5.4.4], add $N_C - N_R$ dummy rows to the cost matrix that gate with all events. If the dummy rows have costs that are significantly greater than $\max_{i,j} c_{i,j}$, then they will only be assigned to columns that would not have participated in the original assignment problem. However, it does not matter in which order the rows are added to the assignment problem. Thus, one could choose to add the dummy rows last. Consequently, if all of the dummy rows have zero cost, and are added to the problem last, they cannot change any existing assignments. Thus, the dummy rows are not necessary and the assignment algorithm will provide an optimal solution once all $N_R \leq N_C$ have been assigned. Consequently, one does not need to directly prove that the complementary slackness condition in (20) for the inequality constraints has been satisfied.

The fact that dummy rows are not necessary when $N_R < N_C$ and one uses a shortest augmenting path algorithm has been previously considered. In [47], the idea of terminating after all rows have been assigned is mentioned. In [12], the rectangular assignment problem is considered in more detail, where it is also noted that the augmentation phase of the algorithm produces optimal partial assignments at each step. Related algorithms for more general network optimization problems, such as the push-relabel method [18, ch. 26.4], can also be stopped with optimal partial assignments. Auction algorithms for rectangular assignment problems without using dummy rows are presented in [8, 9].

C. Discussion of the Implementation

The Jonker-Volgenant algorithm [32] is typically implemented with an initialization step to accelerate the convergence rate. In its basic form, when considering the case where $N_R = N_C = N$, the initialization algorithm has a worst case complexity of $O(N^3R)$, where R is the range of the elements in $\mathbf{C}$ [14, ch. 4.4.4], [32]. However, it has been noted [14, ch. 4.4.4], [32] that a modified version of the initialization routine can be made to have an $O(N^3)$ complexity. Castañon's modification to the algorithm [27] uses an initialization step that is similar to the auction algorithm with a fixed ϵ-scaling parameter, which means that one is not always guaranteed to obtain a globally optimal solution. The modified Jonker-Volgenant algorithm described in this paper does not use any initialization.

The median and worst case execution times of the implementation of the Jonker-Volgenant algorithm given in this paper are considered when the algorithm is run on random matrices where every element was chosen uniformly between 0 and 1. It does not matter whether the elements were chosen between 0 and 1 or between 0 and some other number, because unlike other assignment algorithms, simply multiplying the cost matrix by a positive constant does not change the computational complexity of the algorithm. Because fourth-generation programming languages, such as Matlab, are commonly used for prototyping algorithms, whereas third-generation languages, such as C, C++, are commonly used for more practical implementations that can be built into real systems,[10] the computational speed of the algorithm is determined for two implementations in Matlab, two implementations in C and one implementation in C++.

[10] Though some systems might be implemented using field programmable gate arrays (FPGAs) that are programmed using hardware description languages, such as Verilog or the Very High Speed integrated Circuits Hardware Description Language (VHDL), C is simpler to program and a quick search online will reveal multiple programs that can convert C code into such hardware languages.

TABLE I
The Median and Worst Observed Execution Times of the Modified Jonker-Volgenant Algorithm without Initialization when run on Random Matrices of the Sizes Indicated, Implemented in Matlab, C, and C++. In Matlab and C, the Algorithm was Implemented Either so that the Innermost Loop Scanned the Data Across Rows or Across Columns. The C++ Implementation, Which Forms the Basis of the First Step of the *k*-Best Rectangular 2D Assignment Algorithm of Section III, Scanned only Across Rows. The Execution Times are taken from 1000 Monte Carlo Runs. Note that the Execution Times for the 500×1000 Problem are Always Less Than Those for the 500×500 Problem and that the Median Execution Times of the Row-Wise Algorithms are Always Less Than Those of the Corresponding Column-Wise Algorithms

	Matlab Implementation				C Implementation				C++ Implementation	
	Row-Wise		Column-Wise		Row-Wise		Column-Wise		Row-Wise	
Problem Size	Median	Worst Case	Median	Worst Case	Median	Worst Case	Median	Worst Case	Median	Worst Case
100×100	22.1 ms	63.4 ms	22.4 ms	68.4 ms	0.518 ms	1.34 ms	0.553 ms	1.28 ms	0.723 ms	2.20 ms
200×200	65.9 ms	139 ms	71.5 ms	145 ms	2.59 ms	5.07 ms	3.08 ms	9.18 ms	3.09 ms	6.12 ms
500×500	376 ms	697 ms	530 ms	892 ms	20.9 ms	54.2 ms	46.4 ms	163 ms	26.4 ms	62.3 ms
500×1000	165 ms	288 ms	137 ms	206 ms	14.7 ms	26.2 ms	23.0 ms	57.5 ms	18.1 ms	32.2 ms
3000×3000	20.6 s	25.4 s	49.2 s	63.3 s	1.90 s	3.56 s	9.81 s	1351 s	2.16 s	3.73 s

The C implementations of the algorithm mirror the Matlab implementations, except care is taken to allocate all memory outside of the loops. The reason two implementations are present each in C and in Matlab is because modern processors, such as the Intel Xeon E5645 [19], on which the simulations are run, contain sophisticated prefetch algorithms that try to fill the processor's cache with data (prefetch data) that it anticipates the program will need. However, if the program requests data from widely-separated places in memory, then the prefetch algorithms will perform poorly leading to a large number of cache misses and slower execution time. The implementations of the 2D assignment algorithm in Matlab and C thus differ in the order in which the rows or columns of the assignment matrix are scanned.

The assignment matrices given to the different algorithms are generated in Matlab. The implementations in C and C++ are called from Matlab. Matlab stores matrices in memory with column-major ordering.[11] Consequently, the Jonker-Volgenant algorithm implemented such that the shortest path portion scans across rows rather than columns (unlike the implementation described in Section II-B) would be expected to be faster. The C++ implementation of the 2D rectangular algorithm is shared by the *k*-best 2D rectangular assignment algorithm of Section III and only scans across rows in the innermost loop of the algorithm.

One benefit of the simple implementation of the Jonker-Volgenant algorithm without initialization is that the worst case execution time can be estimated without running many Monte Carlo runs. Ignoring the influence of background processes running on a computer, the execution time of the algorithm varies only depending upon how long it takes to find the shortest augmenting path each time. Thus, by modifying the termination condition to force the algorithm to take the maximum number of loops, modifying the elements in the loops to force the if-statement to always be true, and adjusting the code to make sure that no invalid memory locations are accessed, one can estimate the worst case execution time of the algorithm. Whereas such worst case execution times were given in the conference work preceding this paper [22], they are omitted here as a truly firm bound is very processor specific, requiring one to force a as many false branch predictions[12] and cache misses on modern processors as possible for the bound to be valid.

The algorithms are run on random assignment matrices of varying sizes, as shown in Table I. All of the algorithms modify copies of the input matrices using (5) and (6) to guarantee that the matrices are appropriate for the algorithm. Only minimization is performed in the simulations. One thousand Monte Carlo runs are performed on a computer made by the Xi Corporation running Windows 7 with two Intel Xeon E5645 processors and 12 gigabytes (GB) of random access memory (RAM) in Matlab 2013b. In order to speed up the simulations, the parallel processing toolbox is used to run Monte Carlo runs across 12 processor cores simultaneously. (The computer has 24 cores in total).

As can be seen, even for hundreds of targets, the execution time of the algorithms implemented in C is on the order of milliseconds. The row-wise implementations of the algorithms, which allow for fewer cache-misses, are faster than the column-wise implementations with the

[11] The first column of data is followed by the second column and so on. The opposite of column major ordering is row-major ordering.

[12] When an "if" statement arises in the code, modern processors might try to anticipate the outcome of the conditional statement before the condition has been evaluated in order to be able to execute a large block of instructions in parallel. A false prediction means that the preexecuted results of the false branch must be discarded and leads to a worse execution time.

TABLE II

The Median and Worst-Observed Execution Times of the ϵ-Scaled Forward Auction Algorithm, the Forward Auction Algorithm with a Fixed ϵ Guaranteeing Global Convergence and the Modified Jonker-Volgenant Algorithm of this Paper Implemented in Matlab Solving the Maximization Problem over 1002 Monte Carlo Runs, whereby the First Two Runs were not Counted, because they were Generally Significantly Slower (Presumably, Matlab Compiled and Optimized the Code in Those Steps). The Problem Size is the Size of the Random Matrices (4×4, 8×8, ...). The Auction Algorithms were Designed for Use with Integer Costs, so the Cost Matrix $\mathbf{C}$ was Randomly Chosen to Provide 9 Digits of Precision. In All Instances, the Modified Jonker-Volgenant Algorithm had Better Mean and Worst Case Performance

	ϵ-Scaled Auction		Forward Auction		Modified Jonker-Volgenant	
Problem Size	Median	Worst Case	Median	Worst Case	Median	Worst Case
4	5.44 ms	11.3 ms	296 μs	1.52 ms	352 μs	556 μs
8	9.02 ms	23.1 ms	762μs	8.14 ms	665μs	797μs
16	15.9 ms	35.7 ms	2.59 ms	26.8 ms	1.38 ms	2.00 ms
32	29.1 ms	62.3 ms	9.97 ms	78.9 ms	3.13 ms	4.16 ms
64	54.5 ms	112 ms	43.6 ms	309 ms	7.79 ms	10.37 ms
128	116 ms	246 ms	199 ms	2.13s	22.2 ms	30.7 ms

difference increasing as a function of the dimensionality of the cost matrix. The Matlab implementations, with Matlab being an interpreted language rather than a compiled programming language, are the slowest of all. The execution time for the rectangular assignment problem is less than that of the square assignment problem with the same number of rows. An explanation for this is that the extra columns decreased the likelihood that two rows would contest the same column.

The 3000×3000 matrix example is chosen to demonstrate how the need for parallelization has changed with advances in hardware and algorithms over the years. In 1991, a parallelized shortest augmenting path algorithm that ran over 14 processors (and was implemented in such a manner that the run time depended on the range of values of the costs) took 811 s to run in the worst case on a random 3000×3000 matrix [2]. That is about 427 times slower than the median run time of this algorithm. However, for such large problems, the quality of the assignment algorithm is more important than the hardware on which the algorithm is run. For example, if one were to solve the 3000×3000 assignment problem by evaluating all combinations via brute force, one would need to consider $3000! \approx 4 \times 10^{9130}$ different possible assignments, which is not computationally feasible on modern hardware.

In the event that the algorithm given here is not fast enough to solve a particularly massive optimization problem within a desired time interval, then modifications for parallelization considered in [2, 48] can be used. Additional references for parallelization techniques applied to shortest augmenting path algorithms are given in [14, ch. 4.11.3]. One of the main arguments for using the auction algorithm over the shortest augmenting path algorithms is its ability to be more easily parallelized. Given well-structured problems, a well-implemented highly parallelized auction algorithm can be faster than a shortest augmenting path algorithm for assignment in the average case [51]. However, the modified Jonker Volgenant algorithm used in this paper is good on generic, unstructured problems.

D. Comparison to Auction Algorithms

To better understand why this paper focusses on shortest augmenting path 2D assignment algorithms rather than using variants of the auction algorithm, which are significantly more common in the literature, this subsection looks at specific scenarios where the auction algorithm can perform poorly. Here, all of the auction algorithm variants are implemented in Matlab. The variants considered are:

1) the forward auction algorithm using ϵ-scaling described in [6, ch. 7.1] using the open-source implementation for square matrices given in [3]. Like most versions of the auction algorithm in the literature, this is only suited for integer-valued cost matrices $\mathbf{C}$. The ϵ-scaling causes this variant of the auction algorithm to have a particularly low computational complexity. The default heuristic method of initially setting the ϵ parameter used in the code is

$$\epsilon = \max_{i,j} \left| c_{i,j}(N+1) \right| \tag{24}$$

if the cost matrix $\mathbf{C}$ is an $N \times N$ matrix.

2) the basic forward auction algorithm for square matrices with a fixed ϵ set to guarantee an optimal solution. This is described in [6, ch. 7.2], among other sources. An optimal solution is guaranteed by setting the ϵ as

$$\epsilon = \frac{\Delta_{\min}}{1.01N} \tag{25}$$

where $\Delta_{\min}$ is the smallest, positive nonzero difference between entries in $\mathbf{C}$. The 1.01 term could be any value larger than 1 to guarantee that the algorithm converges to the globally optimal solution.

Table II shows the runtimes of the different algorithms when run on random integer cost matrices with up to 9

TABLE III
The Execution Times of the Auction Algorithms and the Modified Jonker-Volgenant Algorithm when Run in Matlab 2015b on Magic Matrices or Matrices Full of Ones of the Indicated Dimensions (32×32, 64×64, ...). The Forward Auction Algorithm Performs Significantly Worse than with Random Matrices in Table II as the Size Increases. It Can be Seen that the Values of the Matrices have an Effect on Runtime. This Changes with Sparsity, as Table IV Shows

Problem Size	ϵ-Scaled Auction	Forward Auction	Modified Jonker-Volgenant
32 (Magic)	11.3 ms	235 ms	1.87 ms
32 (Ones)	4.49 ms	8.17 ms	2.62 ms
64 (Magic)	33.8 ms	4.64 s	6.61 ms
64 (Ones)	9.04 ms	34.3 ms	10.0 ms
128 (Magic)	119 ms	93.6 s	31.8 ms
128 (Ones)	115 ms	147 ms	46.6 ms

TABLE IV
The Execution Times of the Auction Algorithms and the Modified Jonker-Volgenant Algorithm when Run in Matlab 2015b on Magic Matrices of the Indicated Dimensions where all Odd Numbered Entries were Marked as Impermissible Assignments. The Results are Compared with Matrices of all Ones with the Same Impermissible Entries. Compare with Table III to see the Effects of Sparsity on the Problem

Problem Size	ϵ-Scaled Auction	Forward Auction	Modified Jonker-Volgenant
32 (Magic)	14.3 ms	2.60 ms	1.34 ms
32 (Ones)	2.92 ms	4.72 ms	1.80 ms
64 (Magic)	42.2 ms	9.45 ms	4.17 ms
64 (Ones)	5.30 ms	17.58 ms	6.20 ms
128 (Magic)	155 ms	38.5 ms	18.0 ms
128 (Ones)	64.27 ms	76.5 ms	28.7 ms

digits per entry. In Matlab, the command C=floor (rand (N) *1e10); was used to generate each random cost matrix. The simulations were run in Matlab 2015a under Mac OS X on a computer with two 6-core Intel Xeon processors at 2.93 GHz with 32 GB of RAM. 100 Monte Carlo runs were performed and the times for the first two runs were discarded. The runs were performed sequentially (not parallelized). In all instances, the median and worst case execution times of the modified Jonker Volgenant algorithm of this paper were better than either of the auction algorithms.

However, the performance of the algorithms changes depending on the structure and sparsity of the matrices. The run times of the forward auction algorithm without ϵ-scaling can become particularly bad given certain types of matrices. For example, Table III shows the run times of the algorithms from a single Monte Carlo run (three runs were performed and the time of the first two discarded) when run on magic matrices in Matlab 2015b and also on matrices of all ones of the same size. The magic matrices were generated using the magic command in Matlab. It can be seen that the modified Jonker-Volgenant algorithm is always fastest or nearly the fastest and that the forward auction algorithm without ϵ-scaling performs extremely poorly when dealing with magic matrices.

The behavior of the algorithms also differs greatly as a function of the sparsity of the problem. Table IV is the same as Table III, except all odd-valued entries (50% of the entries) in the magic matrices were marked as forbidden assignments. The matrices of all ones were made to have the same forbidden assignments. It can be seen that the execution times of the forward auction algorithm without ϵ-scaling is dramatically faster when the problem is sparser. The modified Jonker-Volgenant algorithm also speeds up, whereas the ϵ-scaled auction algorithm is slower for all of the magic matrices, but faster for the matrices of all ones. Additionally, the ϵ-scaled auction algorithm is now slower on all of the magic matrix problems than either algorithm.

Thus, the modified Jonker-Volgenant algorithm is chosen in this paper over variants of the auction algorithm, because the auction algorithm is often slower, and implementations without ϵ-scaling can have extremely poor performance when run on certain types of matrices. Additionally, the modified Jonker-Volgenant algorithm works well with noninteger costs, whereas the auction algorithms are often derived requiring integer costs to assure convergence, implying that the performance can worsen as the number of significant digits increases.

III. FINDING THE k-BEST 2D ASSIGNMENTS

A. Murty's Algorithm

In addition to the best solution to the 2D assignment problem, one often wants to obtain a ranked set of the k-best solutions to the 2D assignment problem. For example, the production of multiple hypotheses is useful for assessing the ambiguity of the solution, which might be expressed in terms of entropy [29] or the identity variance given by a set of hypotheses [23]. Most algorithms for solving the k-best 2D assignment problem are based on a variant of Murty's algorithm [39], [14, ch. 5.4.1], which repeatedly solves 2D assignment problems with an increasing number of constraints. Alternate alternative algorithms are compared with Murty's algorithm in [20, 21] and are found to have worse computational efficiency. It is noted in [21] that Murty's algorithm is fast enough to perform real-time data association.

Murty's algorithm in its most basic form is given by the following steps

Murty's Algorithm

1) (Initialize) An empty ordered list is created and the value $k = 1$ represents the current ranking of the solution being found.

2) (The Best Solution) Find the globally optimal (lowest cost) solution to the 2D assignment problem for cost matrix $\mathbf{C}$ using, for example, the modified Jonker-Volgenant algorithm described in Section II and insert the solution into the ordered list. The list is ordered

by the cost of each solution. The solution S is associated with an empty list of constraints, C_S.

3) (Record the Solution) Let S and C_S be the lowest cost solution on the ordered list and its associated list of constraints. Remove the lowest cost solution from the list and record it as the kth solution. If k is the maximum number of solutions desired, then terminate the algorithm.

4) (Split the Solution) Solution S will be split into a number of disjoint subproblems each with an increasing number of constraints. All subproblems inherit the constraint set C_S of the parent solution augmented by something else.

- The first subproblem has the constraint set C_S augmented with the constraint that the first assignment in the solution to S that is not directly specified by a constraint in C_S is forbidden. Forbidding an assignment is the same as replacing an entry in $\mathbf{C}$ with ∞. Solve this assignment problem and add its solution and constraint set to the ordered list.
- The second subproblem has the constraint set C_S augmented with the constraint that the first assignment in S that is not forced due to a constraint in C_S must be made and the second assignment in S not directly specified by a constraint in C_S is forbidden. Adding a constraint that an assignment be made is the same as removing a row and column from $\mathbf{C}$, since there is no longer a choice on that assignment. Forbidding an assignment is the same as replacing the corresponding element with ∞. Solve this assignment problem and add its solution and constraint set to the ordered list.
- The nth subproblem is generated by requiring that the first $n-1$ assignments in S that are not strictly assigned due to constraints in C_S be assigned and the nth assignment in S that is not due to a constraint in C_S be forbidden. Each new problem will be solved and added to the ordered list along with its associated constraint set. If a problem is infeasible, then it will be discarded and will not be added to the list. This continues until solution S can no longer be split into subproblems.

5) (Loop) If the ordered list is empty, then all possible solutions to the problem have been found and the algorithm terminates. Otherwise, set $k = k + 1$ and go to step 3.

The description of Murty's algorithm given above makes use of an ordered list. However, any type of ordered data structure can be used. For example in [31], the use of a binary search tree was suggested, and in [20], it was suggested that a heap be used.[13] The Matlab implementation used for the simulations in this paper uses a binary heap as described in [49, ch. 6.4].

[13] A heap is a type of advanced data structure whose complexity adding and deleting elements is $O(\ln[n])$ or $O(1)$ depending upon the type of heap, where n is the number of items in the heap when the operations is being performed, and finding the minimum item also occurs in $O(\ln[n])$ or $O(1)$ [18].

For square assignment matrices such that $N_R = N_C = N$, the complexity of Murty's algorithm when implemented with an $O(N^3)$ assignment algorithm without using any particular optimizations is $O(kN^4)$. However, multiple authors have noted [16, 20, 37] that when using a 2D assignment algorithm, such as that described in Section II, and a square cost matrix, a partial solution can be used to kickstart the assignment algorithm from the problem that is being split, bringing the computational complexity down to $O(kN^3)$.

Let $\mathbf{col4row}_S$ and $\mathbf{row4col}_S$, $\mathbf{u}_S$, and $\mathbf{v}_S$ contain the optimal solution and dual variables to problem S, which is being split in step 4 of Murty's algorithm. Every split hypotheses contains a new constraint for an assignment that is no longer allowed. What was realized in [16, 20, 37] is that the optimal solution to S with this assignment removed from $\mathbf{col4row}_S$ and $\mathbf{row4col}_S$ and the unchanged dual solutions $\mathbf{u}_S$ and $\mathbf{v}_S$, satisfy the complementary slackness condition in (19). Thus, $\mathbf{u}_S$ and $\mathbf{v}_S$ as well as $\mathbf{u}_S$ and $\mathbf{v}_S$ with the forbidden assignment removed can be the initial values to the shortest path algorithm described in the pseudocode in Section II-B to complete the assignment problem. The previous best solution is forbidden (by putting an infinite value in the appropriate entry in the cost matrix) and the assignments that are not allowed to change are not considered in the shortest path optimization (like removing rows and columns form the cost matrix). Thus, only one run of the shortest path algorithm is needed to update the solution.

The requirement of using that optimization, however, is that the assignment matrix be square, as there is no longer a guarantee that after the shortest path augmentation, the second complementary slackness condition in (20) will be fulfilled. Any rectangular cost matrix with $N_C > N_R$ can be made square by adding zero-cost dummy rows, so this is not an issue. In [37], it was noted that no constraints should be applied to the dummy rows when splitting hypotheses in step 4 of Murty's algorithm lest one obtain hypotheses that differ only in meaningless assignments of dummy rows to otherwise unassigned columns. This obviates the need for steps described in some implementation, such as in [13, ch. 6.5.2], where duplicate solutions must be removed.

This modified version of Murty's algorithm is not the only $O(kN^3)$ complexity algorithm for finding the k-best hypotheses. Another is presented in [16] and [14, ch. 5.4.1]. That algorithm does not require any complicated inheritance of dual variables to assure its computational complexity, but it does appear to be a more complicated algorithm, requiring the use of a special algorithm to find second-best assignments, and it makes use of a tree data structure. For that reason, only Murty's algorithm is considered here.

B. Discussion of the Implementation

Though the previous subsection discussed enforcing constraints akin to setting elements in the cost matrix

TABLE V
The Median and Worst Case Execution Times to Generate the Given Number of Hypotheses Shown for Varying Problem Sizes with 1000 Monte Carlo Runs in Matlab and C++. The Median Speed of the C++ Implementation is up to 379 Times Faster than the Matlab Implementation in the Scenarios Considered

		Matlab Implementation		C++ Implementation	
Size	# Hyps	Median	Worse Case	Median	Worst Case
10×20	2	13.6 ms	109.6 ms	0.092 ms	2.22 ms
	25	188 ms	592 ms	0.241 ms	0.804 ms
	50	388 ms	1.07 s	0.374 ms	0.974 ms
50×100	2	200 ms	504 ms	1.60 ms	2.99 ms
	25	1.74 s	4.73 s	7.13 ms	13.9 ms
	50	3.19 s	7.38 s	11.8 ms	26.5 ms
100×100	2	400 ms	886 ms	1.68 ms	3.03 ms
	25	6.58 s	10.83 s	17.8 ms	34.2 ms
	50	12.71 s	18.75 s	33.5 ms	51.9 ms

equal to zero, or removing rows and columns from the cost matrix for assignments that are fixed, such an implementation is quite computationally inefficient, particularly for large cost matrices, requiring allocating and copying large amounts of data for each cost matrix in addition to allocating space for the partial solutions that are constrained to exist. A more efficient implementation never copies the cost matrix, but rather keeps track of the partial assignment as well as which rows and columns are fixed or forbidden. Such constraints can be stored in arrays. Moreover, an efficient implementation would inherit the dual variables and partial solution from each problem as it splits to bring down the computational complexity to $O(kN^3)$. Doing that, however, would require the addition of dummy rows to a rectangular cost matrix when $N_R \neq N_C$. Dummy rows were used in the implementation of the algorithm for this paper in Matlab and C++. C++ was used instead of C so that the priority_queue template class could be used as an ordered list. In Matlab, a binary heap class was created to function as an ordered list.

Table V shows the execution times for 1000 Monte Carlo runs of the two implementations on Murty's algorithm on assignment matrices of various sizes. As is the case in Section II-C, the cost matrices are generated with elements randomly generated uniformly between 0 and 1. Both implementations of the algorithm scanned the assignment matrices row-wise to obtain the best computational performance. The results demonstrate that the algorithm can produce a large number of hypotheses on moderate sized problems in a fraction of a second.

IV. CONCLUSIONS

An overview of the literature covering 2D assignment algorithms for rectangular problems and for k-best assignment was given. It was determined that the auction algorithm, despite its simplicity, is not always an ideal choice for performing 2D assignment if one requires globally optimal solutions, due to difficulties associated with choosing the complementary slackness parameter. A complementary slackness parameter that is too large cannot guarantee a globally optimal solution; a parameter that is too small can have a very slow rate of convergence, and adaptive methods for setting the complementary slackness parameter can be difficult to implement for use with arbitrary cost matrices. Moreover simulations demonstrated that a forward auction algorithm with scaling of the complementary slackness parameter was slower than a modified form of the Jonker-Volgenant algorithm. Consequently, the Jonker-Volgenant shortest augmenting path algorithm was chosen for implementation in this paper.

The implementation combines concepts from multiple papers in the literature to create an efficient algorithm that can be generalized to find the k-best 2D assignments rather than just the best 2D assignment. Whereas previous work has considered dual variable inheritance to improve k-best 2D assignment algorithms, this paper appears to be the only work that combines aspects of rapid infeasibility detection and dual variable inheritance to create a more efficient k-best 2D assignment algorithm. It was demonstrated that the order in which one scans the elements in the cost matrix matters, with the influence being attributable to cache prediction algorithms that are built into the processor. This is an implementation aspect that is not necessarily obvious to noncomputer science professionals. Additionally, execution time differences between the C and the C++ implementations of the 2D assignment algorithm demonstrate the difference that the compiler and a few minor changes between languages can make.

Matlab code implementing the 2D rectangular shortest augmenting path algorithm is given in the Appendix. It has been tested in Matlab 2013b, 2014a, 2014b, and 2015a.

APPENDIX. MATLAB CODE FOR A RECTANGULAR SHORTEST AUGMENTING PATH 2D ASSIGNMENT ALGORITHM

Below is code for a 2D assignment algorithm in Matlab. Code for 2D and *k*-best 2D assignment is available online at https://github.com/DavidFCrouse/Tracker-Component-Library.

```
function [col4row, row4col, gain, u, v]=assign2D(C,maximize)
%%ASSIGN2D Solve the two-dimensional assignment problem with a
% rectangular cost matrix C, scanning row-wise using a shortest
% augmenting path algorithm.
%
%INPUTS: C A numRowXnumCol cost matrix that does not contain
% any NaNs and where the largest finite element minus
% the smallest element is a finite quantity (does not
% overflow).
% maximize If true, the minimization problem is transformed
% into a maximization problem. The default if this
% parameter is omitted is false.
%
%OUTPUTS: col4row A numRowX1 vector where the entry in each element
% is an assignment of the element in that row to a
% column. 0 entries signify unassigned rows.
% row4col A numColX1 vector where the entry in each element
% is an assignment of the element in that column to a
% row. 0 entries signify unassigned columns.
% gain The sum of the values of the assigned elements in
% C.
% u The dual variable for the columns.
% v The dual variable for the rows.
%
%DEPENDENCIES: None
%
%If the number of rows is <= the number of columns, then every row is
%assigned to one column; otherwise every column is assigned to one row. The
%assignment minimizes the sum of the assigned elements (the gain).
%During minimization, assignments can be forbidden by placing Inf in
%elements. During maximization, assignment can be forbidden by placing -Inf
%in elements. The cost matrix can not contain any -Inf elements during
%minimization nor any +Inf elements during maximization to try to force an
%assignment. If no complete assignment can be made with finite cost,
%then col4row and row4col are empty and gain is set to -1.
%
%Note that the dual variables produced by a shortest path assignment
%algorithm that scans by row are not interchangeable with those of a
%shortest path assignment algorithm that scans by column. Matlab stores
%matrices row-wise. Additionally, the dual variables are only valid for the
%transformed cost matrix on which optimization is actually performed, which
%is not necessarily the original cost matrix provided.
%
%October 2013 David F. Crouse, Naval Research Laboratory, Washington D.C.
if(nargin<2)
maximize=false;
end
numRow=size(C, 1);
numCol=size(C, 2);
didFlip=false;
if(numCol>numRow)
C=C';
temp=numRow;
```

```
numRow=numCol;
numCol=temp;
didFlip=true;
end
%The cost matrix must have all non-negative elements for the assignment
%algorithm to work. This forces all of the elements to be positive. The
%delta is added back in when computing the gain in the end.
if(maximize== true)
CDelta=max(max(C));
C=-C + CDelta;
else
CDelta=min(min(C));
C=C-CDelta;
end
%These store the assignment as it is made.
col4row=zeros(numRow, 1);
row4col=zeros(numCol, 1);
u=zeros(numCol, 1);%The dual variable for the columns
v=zeros(numRow, 1);%The dual variable for the rows.
%Initially, none of the columns are assigned.
for curUnassCol=1:numCol
%This finds the shortest augmenting path starting at k and returns
%the last node in the path.
[sink,pred,u,v]=ShortestPath(curUnassCol,u,v,C,col4row,row4col);
%If the problem is infeasible, mark it as such and return.
if(sink== 0)
col4row=[];
row4col=[];
gain=-1;
return;
end
%We have to remove node k from those that must be assigned.
j=sink;
while(1)
i=pred(j);
col4row(j)=i;
h=row4col(i);
row4col(i)=j;
j=h;
if(i== curUnassCol)
break;
end
end
end
%Calculate the gain that should be returned.
if(nargout>2)
gain=0;
for curCol=1:numCol
gain=gain + C(row4col(curCol),curCol);
end
%Adjust the gain for the initial offset of the cost matrix.
if(maximize== true)
gain=-gain + CDelta*numCol;
else
gain=gain + CDelta*numCol;
end
end
if(didFlip== true)
```

```
temp=row4col;
row4col=col4row;
col4row=temp;
temp=u;
u=v;
v=temp;
end
end
function [sink, pred, u, v]=ShortestPath(curUnassCol,u,v,C,col4row,row4col)
%This assumes that unassigned columns go from 1:numUnassigned
numRow=size(C, 1);
numCol=size(C, 2);
pred=zeros(numCol, 1);
%Initially, none of the rows and columns have been scanned.
%This will store a 1 in every column that has been scanned.
ScannedCols=zeros(numCol,1);
%This will store a 1 in every row that has been scanned.
ScannedRow=zeros(numRow,1);
Row2Scan=1:numRow;%Columns left to scan.
numRow2Scan=numRow;
sink=0;
delta=0;
curCol=curUnassCol;
shortestPathCost=ones(numRow,1)*inf;
while(sink== 0)
%Mark the current row as having been visited.
ScannedCols(curCol)=1;
%Scan all of the columns that have not already been scanned.
minVal=inf;
for curRowScan=1:numRow2Scan
curRow=Row2Scan(curRowScan);
reducedCost=delta + C(curRow,curCol)-u(curCol)-v(curRow);
if(reducedCost<shortestPathCost(curRow))
pred(curRow)=curCol;
shortestPathCost(curRow)=reducedCost;
end
%Find the minimum unassigned column that was
%scanned.
if(shortestPathCost(curRow)<minVal)
minVal=shortestPathCost(curRow);
closestRowScan=curRowScan;
end
end
if(~isfinite(minVal))
%If the minimum cost column is not finite, then the problem is
%not feasible.
sink=0;
return;
end
closestRow=Row2Scan(closestRowScan);
%Add the column to the list of scanned columns and delete it from
%the list of columns to scan.
ScannedRow(closestRow)=1;
numRow2Scan=numRow2Scan-1;
Row2Scan(closestRowScan)=[];
delta=shortestPathCost(closestRow);
%If we have reached an unassigned row.
if(col4row(closestRow)== 0)
```

```
sink=closestRow;
else
curCol=col4row(closestRow);
end
end
%Dual Update Step
%Update the first row in the augmenting path.
u(curUnassCol)=u(curUnassCol) + delta;
%Update the rest of the rows in the agumenting path.
sel=(ScannedCols~=0);
sel(curUnassCol)=0;
u(sel)=u(sel) + delta-shortestPathCost(row4col(sel));
%Update the scanned columns in the augmenting path.
sel=ScannedRow~=0;
v(sel)=v(sel)-delta + shortestPathCost(sel);
end
```

REFERENCES

[1] Akgül, M.
A genuinely polynomial primal simplex algorithm for the assignment problem.
Discrete Applied Mathematics, **45**, 2 (1993), 93–115.

[2] Balas, E., Miller, D., Pekny, J., and Toth, P.
A parallel shortest augmenting path algorithm for the assignment problem.
Journal of the Association for Computing Machinery, **38**, 4 (Oct. 1991), 985–1004.

[3] Bernard, F.
Fast linear assignment problem using auction algorithm.
Nov. 13, 2014. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/48448-fast-linear-assignment-problem-using-auction-algorithm

[4] Bertsekas, D. P.
The auction algorithm: A distributed relaxation method for the assignment problem.
Annals of Operations Research, **14**, 1 (Dec. 1988), 105–123.

[5] Bertsekas, D. P.
Auction algorithms for network flow problems: A tutorial introduction.
Computational Optimization and Applications, **1**, 1 (Oct. 1992), 7–66.

[6] Bertsekas, D. P.
Network Optimization: Continuous and Discrete Models.
Belmont, MA: Athena Scientific, 1998.

[7] Bertsekas, D. P.
Nonlinear Programming, 2nd ed. Belmont, MA: Athena Scientific, 2003.

[8] Bertsekas, D. P. and Castañón, D. A.
A forward/reverse auction algorithm for asymmetric assignment problems.
Computational Optimization and Applications, **1**, 3 (Dec. 1992), 277–297.

[9] Bertsekas, D. P., Castañón, D. A., and Tsaknakis, H.
Reverse auction and the solution of inequality constrained assignment problems.
SIAM Journal on Optimization, **3**, 2 (May 1993), 268–297.

[10] Bertsekas, D. P., and Tsitsiklis, J. N.
Parallel and Distributed Computation: Numerical Methods.
Englewood Cliffs, NJ: Prentice-Hall, 1989.

[11] Bertsimas, D., and Tsitsiklis, J. N.
Introduction to Linear Optimization. Belmont, MA: Athena Scientific/Dynamic Ideas, 1997.

[12] Bijsterbosch, J., and Volgenant, A.
Solving the rectangular assignment problem and applications.
Annals of Operations Research, **181**, 1 (Dec. 2010), 443–462.

[13] Blackman, S. S., and Popoli, R.
Design and Analysis of Modern Tracking Systems. Norwood, MA: Artech House, 1999.

[14] Burkard, R., Dell'Amico, M., and Martello, S.
Assignment Problems. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2009.

[15] Castañón, D. A.
New assignment algorithms for data association.
In *Proceedings of SPIE: Signal and Data Processing of Small Targets Conference*, Orlando, FL, Apr. 20, 1992, 313–323.

[16] Chegireddy, C. R., and Hamacher, H. W.
Algorithms for finding *k*-best perfect matchings.
Discrete Applied Mathematics, **18**, 2 (Nov. 1987), 155–165.

[17] Cook, S.
The P versus NP problem.
In *The Millenium Prize Problems*, J. Carlson, A. Jaffe, and A. Wiles, (Eds.) Providence, RI: The American Mathematical Society for the Clay Mathematics Institute, 2006.

[18] Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C.
Introduction to Algorithms, 2nd ed. Cambridge, MA: The MIT Press, 2001.

[19] Intel Corporation.
Intel(R) 64 and IA-32 architectures optimization reference manual.
Intel Corporation, Tech. Rep. 248966-026, Apr. 2012. [Online]. Available: http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

[20] Cox, I. J., and Miller, M. L.
On finding ranked assignments with application to multi-target tracking and motion correspondence.
IEEE Transactions on Aerospace and Electronic Systems, **32**, 1 (Jan. 1995), 486–489.

[21] Cox, I. J., Miller, M. L., Danchick, R., and Newman, G. E.
A comparison of two algorithms for determining ranked assignments with application to multitarget tracking and motion correspondence.
IEEE Transactions on Aerospace and Electronic Systems, **33**, 1 (Jan. 1997), 295–301.

[22] Crouse, D. F.
Advances in displaying uncertain estimates of multiple targets.
In *Proceedings of SPIE: Signal Processing, Sensor Fusion, and Target Recognition XXII*, Baltimore, MD, Apr. 2013.

[23] Crouse, D. F., and Willett, P.
Identity variance for multi-object estimation.
In *Proceedings of SPIE: Signal and Data Processing of Small Targets*, Vol. 8137, San Diego, CA, Aug. 25, 2011.

[24] Derigs, U.
The shortest augmenting path method for solving assignment problems.
Annals of Operations Research, **4**, 1 (Dec. 1985), 57–102.

[25] Dijkstra, E. W.
A note on two problems in connection with graphs.
Numerische Mathematik, **1**, 1 (Dec. 1959), 269–271.

[26] Dorhout, B.
Experiments with some algorithms for the linear assignment problem.
Stichting Mathematisch Centrum, Amsterdam, The Netherlands, Tech. Rep., Nov. 1970.

[27] Drummond, O., Castañon, D. A.,and Bellovin, M.
Comparison of 2-D assignment algorithms for sparse, rectangular, floating point, cost matrices.
Journal of the SDI Panels on Tracking, **4** (1990), 81–97.

[28] Fitzgerald, R. J.
Performance comparisons of some association algorithms.

In *ONR/GTRI Workshop on Target Tracking and Sensor Fusion*, Key West, FL, June 22-23, 2004.

[29] Gadaleta, S., Herman, S., Miller, S., Obermeyer, F., Slocumb, B., Poore, A., and Levedahl, M.
Short-term ambiguity assessment to augment tracking data association information.
In *Proceedings of the 8th International Conference on Information Fusion*, Vol. 1, Philadelphia, PA, July 25-29, 2005, 691–698.

[30] Goldberg, A. V., and Kennedy, R.
An efficient cost scaling algorithm for the assignment problem.
Mathematical Programming, **71**, 2 (Dec. 1995), 153–177.

[31] Hamacher, H. W., and Queyranne, M.
k-best solutions to combinatorial optimization problems.
Annals of Operations Research, **4**, 1 (Dec. 1985), 123–145.

[32] Jonker, R., and Volgenant, A.
A shortest augmenting path algorithm for dense and sparse linear assignment problem.
Computing, **38**, 4 (Mar. 1987), 325–340.

[33] Kadar, I., Eadan, E. R., and Gassnet, R. R.
Comparison of robustized assignment algorithms.
In *Proceedings of SPIE: Signal Processing, Sensor Fusion, and Target Recognition VI*, Vol. 3068, Orlando, FL, Apr. 21, 1997, 240–249.

[34] Kuhn, H. W.
The Hungarian method for the assignment problem.
Naval Research Logistics, **2**, 1–2 (Mar. 1955), 83–97.

[35] Levedahl, M.
Performance comparison of 2-D assignment algorithms for assigning truth objects to measured tracks.
In *Proceedings of SPIE: Signal and Data Processing of Small Targets*, Vol. 4048, Orlando, FL, Apr. 24, 2000, 380–389.

[36] Malkoff, D. B.
Evaluation of the Jonker-Volgenant-Castanon (JVC) assignment algorithm for track association.
In *Proceedings of SPIE: Signal Processing, Sensor Fusion, and Target Recognition VI*, Vol. 3068, Orlando, FL, Apr. 21, 1997, 228–239.

[37] Miller, M. L., Stone, H. S., and Cox, J.
Optimizing Murty's ranked assignment method.
IEEE Transactions on Aerospace and Electronic Systems, **33**, 3 (July 1997), 851–862.

[38] Munkres, J.
Algorithms for the assignment and transportation problems.
Journal of the Society for Industrial and Applied Mathematics, **5**, 1 (Mar. 1957), 32–38.

[39] Murty, K. G.
An algorithm for ranking all the assignments in order of increasing cost.
Operations Research, **16**, 3 (May-June 1968), 682–687.

[40] Papadimitriou, C. H.
Combinatorial Optimization: Algorithms and Complexity. Englewood Cliffs, NJ: Prentice-Hall, 1982.

[41] Pattipati, K., and Deb, S.
Comparison of assignment algorithms with applications to the passive sensor data association problem.
In *Proceedings of the IEEE International Conference on Control and Applications*, Jerusalem, Israel, Apr. 3-6, 1989, 317–322.

[42] Pattipati, K. R., Deb, S., Bar-Shalom, Y., and Washburn, R. B., Jr.
A new relaxation algorithm and passive sensor data association.
IEEE Transactions on Automatic Control, **37**, 2 (Feb. 1992), 198–213.

[43] Popp, R. L., Pattipati, K. R., and Bar-Shalom, Y.
Dynamically adaptable *m*-best 2-D assignment algorithm and multilevel parallelization.
IEEE Transactions on Aerospace and Electronic Systems, **35**, 4 (Oct. 1999), 1145–1160.

[44] Rink, K. A., and O'Conner, D. A.
Use of the auction algorithm for target object mapping.
Lincoln Laboratory, Cambridge, MA, Tech. Rep. 1044, Feb. 9, 1998.

[45] Sleator, D. D., and Tarjan, R. E.
A data structure for dynamic trees.
Journal of Computer and System Sciences, **26**, 3 (June 1983), 362–391.

[46] Tomizawa, N.
On some techniques useful for solution of transportation network problems.
Networks, **1**, 2 (1971), 173–194.

[47] Volgenant, A.
Linear and semi-assignment problems: A core-oriented approach.
Computers and Operations Research, **23**, 10 (Oct. 1996), 917–932.

[48] Wang, Z.
The shortest augmenting path algorithm for bipartite network problems.
Ph.D. dissertation, Southern Methodist University, Dallas, TX, May 19, 1990.

[49] Weiss, M. A.
Data Structures and Algorithm Analysis in C++, 2nd ed.
Reading, MA: Addison-Wesley, 1999.

[50] Weisstein, E. W. Magic square. 2012. Mathworld. [Online].
Available: http://mathworld.wolfram.com/MagicSquare.html

[51] Zaki, H. A.
A comparison of two algorithms for the assignment problem.
Computational Optimization and Applications, **4**, 1 (Jan. 1995), 23–45.

David Frederic Crouse (S'05—M'12) received B.S., M.S., and Ph.D. degrees in electrical engineering in 2005, 2008, and 2011 from the University of Connecticut (UCONN). He also received a B.A. degree in German from UCONN for which he spent a year at the Ruprecht-Karls Universität in Heidelberg, Germany.

He is currently employed at the Naval Research Laboratory in Washington, D.C. and serves as an associate editor for the *IEEE Aerospace and Electronic Systems Magazine* and has shared online a library of reusable algorithms for target trackers called the Tracker Component Library. His interests lie in the areas of stochastic signal processing and tracking.