2022

Bipartite Graph Learning for Autonomous Task-to-Sensor Optimization

Karpenko, Mark; Ross, Isaac M.; Proulx, Ronald J.; Magallanes, Lara C.

Monterey, California: Naval Postgraduate School

https://hdl.handle.net/10945/71883

NPS-MAE-22-002

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

BIPARTITE GRAPH LEARNING FOR AUTONOMOUS TASK-TO-SENSOR OPTIMIZAITON

by

Mark Karpenko (MAE)
Ronald. J. Proulx (SSAG)
LT Michael Zepeda USN (MAE)

October 2022

Prepared for: Naval Special Warfare Command. This research is supported by funding from the Naval Postgraduate School, Naval Research Program (PE 0605853N/2098).
NRP Project ID: NPS-22-N192-B

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
10/14/2022	Technical Report	10/21/2021	10/22/2022

4. TITLE AND SUBTITLE

BIPARTITE GRAPH LEARNING FOR AUTONOMOUS TASK-TO-SENSOR OPTIMIZAITON

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
		0605853N/2098
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**
NPS-22-N192-B; W2223		

6. AUTHOR(S)
Mark Karpenko, Ronald J. Proulx, LT Michael Zepeda USN

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Control and Optimization Laboratories
Department for Mechanical and Aerospace Engineering
Naval Postgraduate School
Monterey, CA, 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

NPS-MAE-22-002

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
NRP, Naval Special Warfare Command	NRP; NAVSPECWARCOM	NPS-MAE-22-002; NPS-22-N192-B

12. DISTRIBUTION/AVAILABILITY STATEMENT

Distribution statement A: Approved for public release. Distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This study explores the question of how machine learning can be applied to identify the most appropriate 'sensor' for a task by optimizing task-to-sensor matching. The concept of a bipartite graph provides a mathematical framework for task-to-sensor mapping by establishing connectivity between various high-level tasks and the specific sensors and/or processes that must be invoked to fulfil those tasks and other mission requirements. The connectivity map embedded in the bipartite graph can change depending on the availability/unavailability of resources, the presence of constraints (physics, operational, sequencing), and the satisfaction of individual tasks. All of these considerations may be encoded in the value matrix of the graph. Changes can also occur according to the valuation, re-assignment and re-valuation of the perceived task benefit and how the completion of a specific task (or group of tasks) can contribute to the state of knowledge prompting the periodically re-solve the matching. The results of this study show that a deep neural network architecture can be used to solve the bipartite matching problem in an autonomous fashion. The scalability of the problem is demonstrated using an 800 by 800 graph. Learning acceleration is also studied and it is recommended that this aspect be further explored as part of future investigations.

15. SUBJECT TERMS

bipartite graphs, matching problems, assignment problems, machine learning, deep learning, residual networks, large scale problems, accelerated learning

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT	b. ABSTRACT	C. THIS PAGE		
unclassified	unclassified	unclassified	UU	24

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
Mark Karpenko	831-656-3231

STANDARD FORM 298 (REV. 5/2020)
Prescribed by ANSI Std. Z39.18

THIS PAGE INTENTIONALLY LEFT BLANK

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

Ann E. Rondeau
President

Scott Gartner
Provost

The report entitled "Bipartite Graph Learning for Autonomous Task-to-Sensor Optimization" was prepared for Naval Special Warfare Command and funded by the Naval Postgraduate School, Naval Research Program (PE 0605853N/2098).

This report was prepared by:

Mark Karpenko
Research Professor

Ronald J. Proulx
Research Professor

LT Michael Zepeda, USN
MS Student (591)

Reviewed by:

Brian Bingham, Chairman
Mechanical and Aerospace Engineering

Released by:

Kevin B. Smith
Vice Provost for Research

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

This study explores the question of how machine learning can be applied to identify the most appropriate 'sensor' for a task by optimizing task-to-sensor matching. The concept of a bipartite graph provides a mathematical framework for task-to-sensor mapping by establishing connectivity between various high-level tasks and the specific sensors and/or processes that must be invoked to fulfil those tasks and other mission requirements. The connectivity map embedded in the bipartite graph can change depending on the availability/unavailability of resources, the presence of constraints (physics, operational, sequencing), and the satisfaction of individual tasks. All of these considerations may be encoded in the value matrix of the graph. Changes can also occur according to the valuation, re-assignment and re-valuation of the perceived task benefit and how the completion of a specific task (or group of tasks) can contribute to the state of knowledge prompting the periodically re-solve the matching. The results of this study show that a deep neural network architecture can be used to solve the bipartite matching problem in an autonomous fashion. The scalability of the problem is demonstrated using an 800 by 800 graph. Learning acceleration is also studied and it is recommended that this aspect be further explored as part of future investigations.

THIS PAGE INTENTIONALLY LEFT BLANK

A. BACKGROUND

A graph G is bipartite if the vertex set $V(G)$ can be partitioned into two sets U and V in such a way that no two vertices from the same set are adjacent. The sets U and V are called color classes of G and (U, V) is a bipartition of U and V. In fact, the vertices of G can be colored with at most two colors, so that no two adjacent vertices have the same color. A graph is called m by n bipartite, if $|U| = m$ and $|V| = n$. Bipartite graphs can also be multi-layered to represent complex relationships between the vertices. This is one reason why machine learning can be beneficial. That is, machine learning can identify relationships in a deep bipartite graph that may not be obvious by inspection. Bipartite representations can mathematically characterize large data sets and can be extremely advantageous to facilitate autonomous decision making. In this study, the development is restricted to single-layer bipartite graphs, though the results could be extended to deeper graphs as part of future work. In the context of a DoD problem space, the connectivity graph between the state of knowledge, high-level tasks and the optimal matching to individual sensors, can be periodically re-evaluated to provide appropriate online/real-time information for automating decision making and operations. One application area is joint targeting and fires.

B. MATHEMATICAL PROBLEM FORMUALTION

Decision making for task-to-sensor assignment can be abstracted as a bipartite matching problem, which can be written and solved as a constrained minimization program. The n by m bipartite matching problem can be specified as:

$$\begin{aligned} &\text{Minimize} && \sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij}c_{ij} \\ &\text{Subject to} && \sum_{i=1}^{n} x_{ij} = 1 \ (j = 1,2,3,\ldots \mathrm{m}) \\ & && \sum_{j=1}^{m} x_{ij} = 1 \ (i = 1,2,3,\ldots n) \\ & && x_{ij} = \begin{cases} 1 \ if \ row \ i \ is \ assigned \ to \ column \ j \\ 0 \quad otherwise \end{cases} \end{aligned} \tag{1}$$

In the specification of the optimization problem, the matrix of binary decision variables x_{ij} represents all possible the task-to-sensor pairings and c_{ij} are the entries of the assignment cost matrix. The one-to-one mapping of the bipartite graph is enforced by the constraint equations that ensure that there can be only a single non-zero entry in each row and each column of the matrix **x**.

C. CONVENTIONAL SOLUTION APPROACH

The optimization problem (1), is typically solved via its 'dual' through the introduction of dual variables *u* and *v*. The dual problem can be seen in (2) below. Duality theory requires that the optimal solution satisfies *complementary slackness*, which is that the dual variables satisfy (3).

$$\begin{aligned} &\text{Maximize} && \sum_{i=1}^{n} u_i + \sum_{j=i}^{m} v_j \\ &\text{Subject to} && u_i + v_j \le c_{ij} \quad (i = 1,2,3,\dots n), \\ & && \quad (j = 1,2,3,\dots,m \end{aligned} \tag{2}$$

$$x_{ij}\left(c_{ij} - u_i - v_j\right) = 0 \tag{3}$$

The advantage of working with the dual problem is it allows the use a reduced cost matrix $\overline{\mathbf{C}}$ which is generated by the row and column reduction of the primal cost matrix **C,** through the use of the dual variables *u* and *v* respectively. An example of this is seen through a partial solution by row reduction of **C** by subtracting from each row the minimum value of that row and then column reducing **C** by the least column value of each column in Figure 1. The row reduction column vector is *u = (4 6 4 3),* and the column reduction row vector is *v = (4 0 1 0)*. This row and column reduction is typical preprocessing and a common first step in solving the bipartite matching optimization. It is a first attempt at determining the dual variables and the primal solution.

$$\underset{C}{\begin{pmatrix} 9 & 4 & 9 & 11 \\ 10 & 10 & 7 & 6 \\ 9 & 4 & 7 & 9 \\ 13 & 3 & 9 & 6 \end{pmatrix}} \xrightarrow{u+v} \underset{\overline{C}}{\begin{pmatrix} 1 & \underline{0} & 4 & 7 \\ \underline{0} & 4 & 0 & 0 \\ 1 & 0 & 2 & 5 \\ 6 & 0 & 5 & 3 \end{pmatrix}}$$

Figure 1. Transforming an example assignment cost matrix **C** to reduced cost matrix $\overline{\mathbf{C}}$. The transformation is performed through row reduction with *u* = (4 6 4 3), and column reduction through *v* = (4 0 1 0). The underlined entries correspond to the assigned vertices.

Using $\overline{\mathbf{C}}$**,** the partial primal solution to the problem can be attempted by setting the matching according to the underlined zeros in the reduced cost matrix. That is, $u_1 \rightarrow v_2$ and $u_2 \rightarrow v_1$. Other candidate matchings are obtained from the dual problem for each $\overline{c}_{ij} = 0$ entry, subject to the constraint of only one allowed match per row and column. Graphically this is represented in Figure 2. The set of vertices *U* corresponds to the set of tasks and the set of vertices *V* corresponds to the set of sensors to be assigned. The edges, *E,* are valid connections (matchings) between the vertices which satisfy the constraint of $\overline{c}_{ij} = 0$. The

arrowed connections denote the chosen assignments determined from $\bar{\mathbf{C}}$, while the other connections represent the other possible assignments. The partial primal solution for the chosen dual variables is represented by the vector *row(j) = (2 1 0 0),* where *row(j)* currently indicates that the first row is assigned to the second column, the second row to the first column and rows 3 and 4 are currently unassigned. The definition for '*row(j)'* is laid out in (4). It may also be useful to look at the partial solution in terms of the columns instead of rows and this point of view is denoted by *φ(i)* in (5).

$$row(j) = \begin{cases} i & if\ column\ j \text{ is assigned } to\ row\ i \\ 0 & if\ column\ j\ is\ not\ assigned \end{cases} \quad (j = 1, 2, \ldots . m) \tag{4}$$

$$\varphi(i) = \begin{cases} j & if\ row\ i\ is\ assigned\ to\ column\ j \\ 0 & if\ row\ i\ is\ not\ assigned \end{cases} \quad (i = 1, 2, \ldots . n) \tag{5}$$

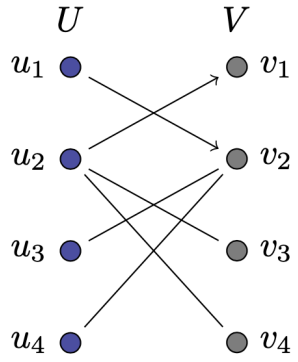


Figure 2. Intermediate bipartite graph $G^0 = (U, V; E)$. The vertices *U* and *V* are encoded in the rows and the columns respectively of $\bar{\mathbf{C}}$. Set E is the set of candidate edges (matchings) denoted by the lines between *U* and *V*.

1. Alternating the Bipartite Graph

In order to iterate the assignments to solve the bipartite matching, a variety of different algorithms can be used. The *Hungarian Algorithm* is one of these that operates by alternating the edges of the bipartite graph $G^i = (U, V; E^i)$, to try to find an additional edge matching. If the attempt is succesful, then the partial primal solution, *row(j)*, is updated. If the attempt is unsuccessful, then the dual variables are adjusted such that the constraints are still met and alternating of edges is attempted to produce a new bipartite graph $G^{i+1} = (U, V; E^{i+1})$. The process of alternating edges, updating primal, and editing the duals is repeated until a complete matching of the bipartite graph is accomplished.

The process of alternating the edges described above is laid out in detail as the algorithm *Procedure Alternate(k)* [1]. A couple of terms need to be defined for the purpose of following the logic of this algorithm. *SU* and *SV* are the scanned vertices *U* and *V*

repsectivily. LV is the labeled vertex of V, which means that the vertex has been assigned as a possible primal solution. The variables $pred_j$ and *sink* are used in the *Hungarian(O^4) Algorithm* which is covered later.

Procedure Alternate(k)
Find an alternating tree rooted at an unassigned vertex $k \in U$.
$SU := LV := SV := \emptyset$;
fail := *false*, *sink* := 0, $i := k$;
while (*fail* = false and *sink* = 0) **do**
    $SU := SU \cup \{i\}$;
    for each $j \in V \setminus LV : c_{ij} - u_i - v_j = 0$ **do** $pred_j = i$; $LV := LV \cup \{j\}$;
    if $LV \setminus SV = \emptyset$ **then** *fail* = true
    else
        let j be any vertex in $LV \setminus SV$;
        $SV := SV \cup \{j\}$;
        if *row(j)* = 0 **then** *sink* := j **else** i := *row(j)*
    endif
endwhile
return *sink*

Working through *Procedure Alternate(k)* suggests an edge between $U = 3$ and $V = 2$ as seen in Figure 3. Now, $U = 1$ has no valid connection so the alternating of edges has failed. *Procedure Alternate(k)* therefore ends so that the dual variables can be updated. For $G^0 = (U, V; E)$, *Procedure Alternate(k)* would have failed even if a different edge had been alternated.

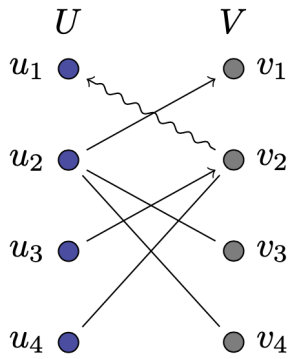


Figure 3. Alternating connecting edges using *Algorithm Alternate(k)*. The straight arrowed lines signify the assigned edges while the squiggled line shows the remapping of the different edges resulting in a vertex with no possible connection.

2. Updating the Dual Variables and the Hungarian Algorithm

Updating of the dual variables can be done through the definition of a new variable δ as in (6), in order to update the dual vairalbes accodirng to (7) and (8). In order for the update to provide valid options without violating the constraints, certain conditions must be met. The three conditions are [1]:

1) The labeled edges in E^i are still available after updating the dual variables. This is, the corresponding entries $\bar{c}_{ij}$ for the [i, j] in E^i remain unchanged with a zero value.
2) A new edge in E^i is added as a result of updating the dual variables.
3) Updating the dual variables still requires that $\bar{c}_{ij} - u_i - v_j \geq 0$ for all values in $\bar{C}$.

$$\delta = \min(c_{ij} - v_i - u_j)\ i \in SU;\ j \in V \setminus LV \tag{6}$$

$$u_i = u_i + \delta\ (i \in SU) \tag{7}$$

$$v_j = v_j - \delta\ (j \in LV) \tag{8}$$

An grpahical example of the dual updating can be seen in Figure 4. Only the circled entries are considered for δ giving a value of $\delta = 1$. This results in $u = (5\ 6\ 5\ 3)$ and $v = (4\ -1\ 1\ 0)$. With new edge connections being possible (see Figure 4b) one can alternate the edges in the new graph G^i shown in Figure 5. Going back to the *Algorithm Alternate(k)* with $k = 3$, no longer fails due to a deadend. *Algorithm Alternate(k)* instead terminates with a new edge being added to the labeled vertices LV.

$$\begin{array}{c} \\ \rightarrow \\ \\ \rightarrow \end{array} \begin{array}{c} \begin{array}{cccc} \downarrow & & \downarrow & \downarrow \end{array} \\ \begin{bmatrix} \textcircled{1} & 0 & \textcircled{4} & \textcircled{7} \\ 0 & 4 & 0 & 0 \\ \textcircled{1} & 0 & \textcircled{2} & \textcircled{5} \\ 6 & 0 & 5 & 3 \end{bmatrix} \end{array}$$

(a)

$$\bar{C} = \begin{bmatrix} 0 & 0 & 3 & 6 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 1 & 4 \\ 6 & 1 & 5 & 3 \end{bmatrix}$$

(b)

Figure 4. Illustrating the dual variable update: (a) Using $\bar{C}$ for finding δ and updating the dual variables. Arrows indicate that $i \in SU$ and $j \in V \setminus LV$ resulting in a $\delta = 1$; (b) updated $\bar{C}$ using the new dual variables $u = (5\ 6\ 5\ 3)$ and $v = (4\ -1\ 1\ 0)$.

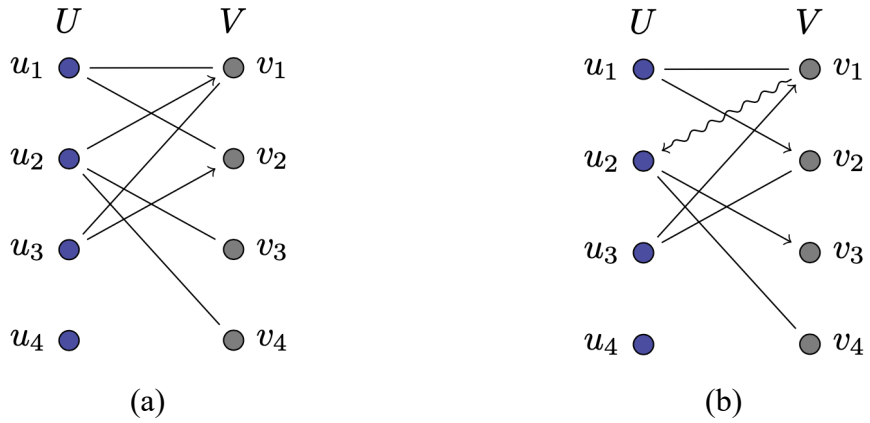


Figure 5. Iterating the bipartite graph: (a) possible graph edges after updating the dual variables (arrows indicate currently assigned edges); (b) alternating of the edges resulting in an additional assigned edge.

The succesful augmenting of the edges results both in an update and addition to the primal solution *row(j)*. The *Hungarian Algorithm(O^4)* is now used to update the primal solution through the use of the *sink* variable [1]. The updated primal solution is contained in the $pred_j$ variable. The algorithm tracks the magnitude of $|\bar{U}|$ to determine when the algorithm terminates. $\bar{U}$ is defined as the subset of U for which the assigned vertices in the primal solution are elements of. Updating the primal solution per the algorithm results in *row(j) = (3 1 2 0) and so* $\bar{U}$ *= (1 2 3),* Consequently, the only valid $k \notin U \setminus \bar{U}$ is $k = 4$. Proceeding further with the *Algorithm Alternate(k)* results in no possible alternating branches as seen in Figure 5b having no edge for u_4 resulting in $SU = (4)$, $LV = SV = \emptyset$. Updating the dual variables again gives $\delta = 1$, $u = (5\ 6\ 5\ 4)$, and v reamaining constant at $v = (4\ -1\ 1\ 0)$. The resulting $\bar{C}$ from the updated duals is seen in Figure 6a. The resulting graph iterate is seen in Figure 6b. Alternating the tree with $k = 4$ results again in a dead end with a rearranging of the assigned edges as seen in Figure 6c.

Hungarian(O^4) Algorithm
initialize *u, v, row, φ and* $\bar{U}$*:*
while $|\bar{U}| < n$ **do**
  let k be any vertex in $U \setminus \bar{U}$;
  while $k \notin \bar{U}$ **do** $\cup$ $\{j\}$;
    sink := Alternate(k);
    if *sink* > *0* **then**
      $\bar{U} := \bar{U} \cup \{k\}$, *j := sink;*
      repeat
      i := $pred_j$ *, row(j) := i, h :=φ(i), φ(i) := j, j := h*
      until $i = k$
    else
    $\delta := \min\{c_{ij} - u_i - v_j : i \in SU, j \in V \setminus LV\}$;
    for each $i \in SU$ **do** $u_i := u_i + \delta$;
    for each $j \in LV$ **do** $v_i := v_i - \delta$;
endif
endwhile
endwhile

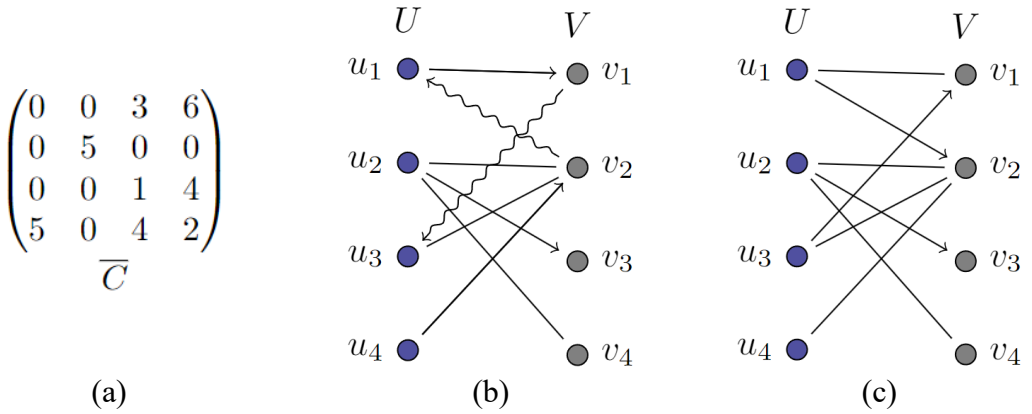


Figure 6. Further iteration of the bipartite graph: (a) $\overline{C}$ after second update of the dual variables; (b) possible re-mappings of the graph links with assigned connections denoted by arrows; (c) re-organization of the edges via *Algorithm Alternate(k).*

Algorithm Alternate(k) is applied again and terminates in failure triggering another update of the dual variables. This update results in $\delta = 1$, $u = (6\ 6\ 6\ 4)$, and $v = (3\ -2\ 1\ 0)$. The reduced cost matrix produced by the newly iterated dual variables is seen in Figure 7a. Inspection of the reduced cost matrix shows that it gives a solution to the matching problm since the complementary slackness conditions are fufilled. Matrix $\overline{C}$ has unique null entries for each column and row pair.

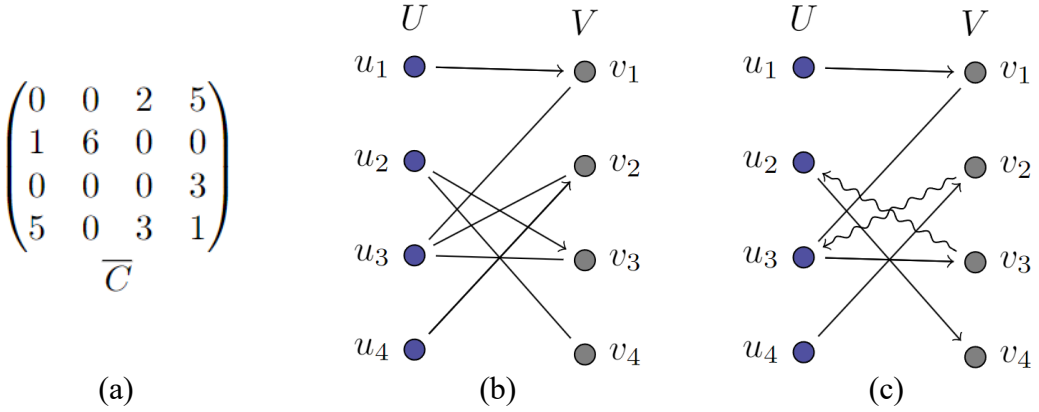


Figure 7. Further iteration of the bipartite graph: (a) $\overline{C}$ after third update of the dual variables; (b) re-organization of the edges via *Algorithm Alternate(k)*; (c) final application of *Algorithm Alternate(k)* results in termination of the matching search – alternate branching does not reduce the cost.

Applying the *Algorithm Alternate(k)* for the final time results in *sink = 4, SU = LV = SV = (1 2 3 4), pred$_j$ = (1 4 3 2)* – see Figures 7b and 7c. Updating the primal solution gives row*(j) = (1 4 3 2)* and *φ(i) = (1 4 3 2)* which corresponds to the assignment matrix **x** as seen in Figure 8a. The optimized assignment cost determed to be *25*. This value is obtained by adding together the cost associated with each vertex pairing – the circled entires in Figure 8b for which x_{ij}=1.

$$X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

(a)

$$C = \begin{pmatrix} \textcircled{9} & 4 & 9 & 11 \\ 10 & 10 & 7 & \textcircled{6} \\ 9 & 4 & \textcircled{7} & 9 \\ 13 & \textcircled{3} & 9 & 6 \end{pmatrix}$$

(b)

Figure 8. Final assignment for minimum cost bipartite matching problem: (a) assignment matrix; (b) cost matrix C where the sum of circled (assigned) entries gives the minimum cost.

D. NEURAL NETWORK REPRESETNATION

Machine learning can be used as an alternative to conventional algorithms (such as the ones described in the previous section) for which computing time can vary greatly with the dimension and structure of the problem. Machine learning can be advantageous over conventional algorithms especially in online/real-time applications because it is possible to transform the problem statement given in (1) into a differential equation. This representation can be executed exceptionally quickly on hardware. As an example, consider a residual network (see Figure 9) that implements a generic input-to-output mapping by applying a sequence of layer-wise transformations on a hidden state [2]:

$$\mathbf{h}_{i+1} = \mathbf{h}_i + f(\mathbf{h}_i, \theta_i) \tag{9}$$

In equation (9), vector $\mathbf{h}_i$ is the input of layer i and vector $\mathbf{h}_{i+1}$ is the output. Vector function f represents the processing applied via the layer weight parameters, θ_i, and activation functions. Referring to Figure 9, the presence of the 'skip' connection, allows some of the information to pass-through across the layers. This resolves issues with vanishing gradients in deep networks [3].

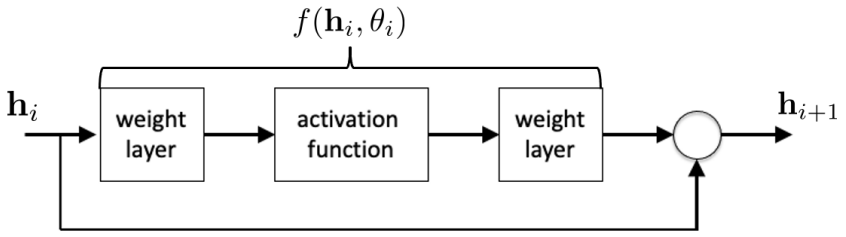


Figure 9. Schematic of a typical residual network illustrating the 'skip' connection.

The conventional residual neural network can be modified slightly by weighting the relative contribution of the pass-through connection. The equation for the layer-wise transformation can be re-written as

$$\mathbf{h}_{i+1} = (1-\sigma)\mathbf{h}_i + \sigma f(\mathbf{h}_i, \theta_i) \tag{10}$$

where σ is the weight. From a dynamical systems perspective, weighting the connections as described helps to stabilize the dissipation of the error function energy, which facilitates the learning process. Thus, the specific neural network architecture (weighted residual network) studied for solving the bipartite matching problem is shown in Figure 10.

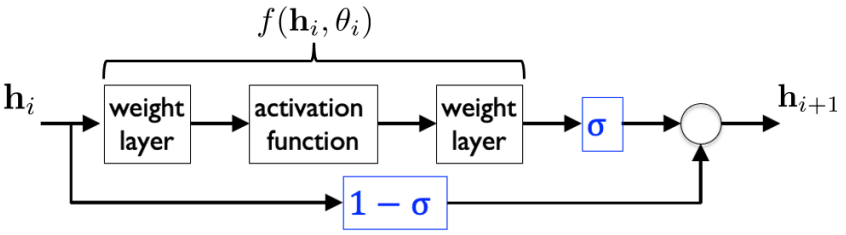


Figure 10. Schematic of a 'weighted' residual network used for solving the bipartite matching problem.

E. ILLUSTRATIVE EXAMPLES

Several illustrative examples of the application of the deep neural network architecture of Figure 10 are presented in this section. First, the problem presented in Section C is solved using machine learning to illustrate that the same solution as the conventional approach can be recovered using the neural network. Then, the problem size is increased in order to explore the scalability of the concepts.

1. Small-Scale Test Problem

Consider the 4×4 matching problem of Section C, where the objective is to obtain the maximum value assignment. The value matrix for the problem is repeated below:

$$C = \begin{pmatrix} 9 & 4 & 9 & 11 \\ 10 & 10 & 7 & 6 \\ 9 & 4 & 7 & 9 \\ 13 & 3 & 9 & 6 \end{pmatrix} \tag{11}$$

To objective of the machine learning problem is to produce (as the network output) the optimal bipartite matching. There is one output neuron for each of the $4 \times 4 = 16$ possible pairings. Figure 11 shows the evolution of the neural network outputs for the maximum value matching. The interpretation of the curves is as follows: In Figure 11, each row, *i*, and column, *j*, represents the activation, x_{ij}, of the connection between input u_i and output v_j. If $x_{ij} = 1$ then there is a connection from u_i to v_j. On the other hand, if $x_{ij} = 0$ then there is no connection between u_i to v_j. From the plots, it is possible to extract the minimum cost bipartite matching: $u_1 \rightarrow v_1$, $u_2 \rightarrow v_4$, $u_3 \rightarrow v_3$, and $u_4 \rightarrow v_2$. This results in the bipartite graph shown in Figure 12. This is which is the same solution determined by the conventional approach.

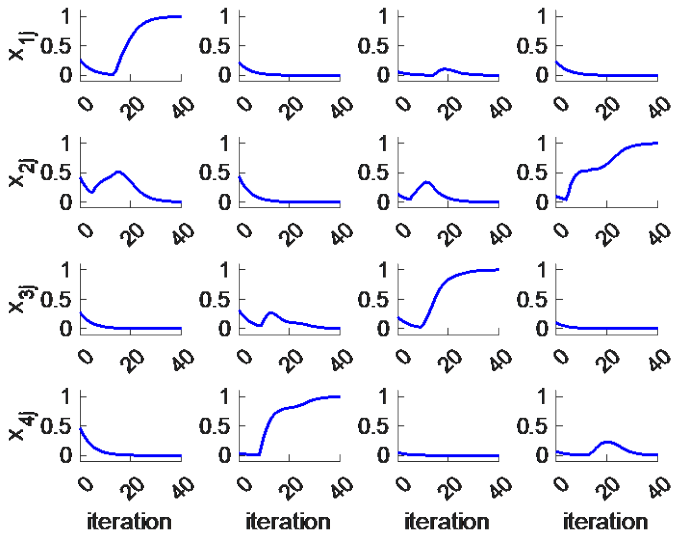


Figure 11. Evolution of the neural network outputs for minimum cost matching of small-scale test problem

The neural network architecture was also applied to solve the 'maximum value' bipartite matching by interpreting the cost matrix as $C = -V$ (see equation 11). Figure 13 shows the evolution of the network outputs for the maximum value problem. The interpretation of the curves is the same as before. Each row, *i*, and column, *j*, represents that strength, x_{ij}, of the connection between input u_i and output v_j. From rows 1 and 2, it is easy to see that the optimal matching is $u_1 \rightarrow v_4$ and $u_2 \rightarrow v_2$. However, in rows 3 and 4 the correct matching is ambiguous because output neurons have an output close to 1.

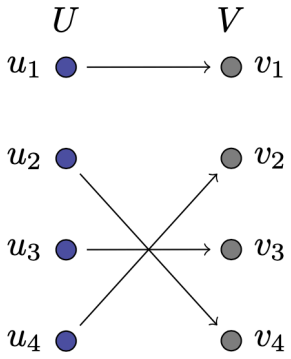


Figure 12. Bipartite graph for the minimum cost matching.

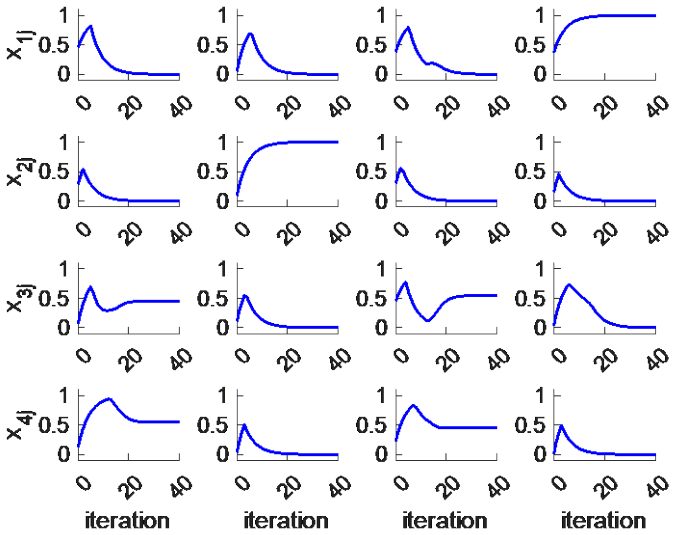


Figure 13. Evolution of the neural network outputs for maximum value matching of small-scale test problem. Note that the assignment of u_3 and u_4 is ambiguous since no output neuron shows an activation close to the value 1.

From the plots it appears that either $u_3 \rightarrow v_1$ or $u_3 \rightarrow v_3$ and that either $u_4 \rightarrow v_1$ or $u_4 \rightarrow v_3$. This is an artifact of solving a 'discrete' problem using 'continuous' network outputs and is related to the neural network fixed points. The issue can be resolved by selecting the winner-take-all solution for each row. This strategy gives $u_3 \rightarrow v_3$ and $u_4 \rightarrow v_1$ as the correct solution. The maximum value for this assignment is $-1 \times (-41) = 41$. The negation is because the network is set up to minimize which is a negated maximization problem. The matching for u_3 and u_4 can also be autonomously disambiguated by applying the network to the ambiguous subproblem (see Figure 14). The corresponding bipartite graphs are shown in Figure 15. Note that this is again the same solution that would be arrived at using the conventional solution approach.

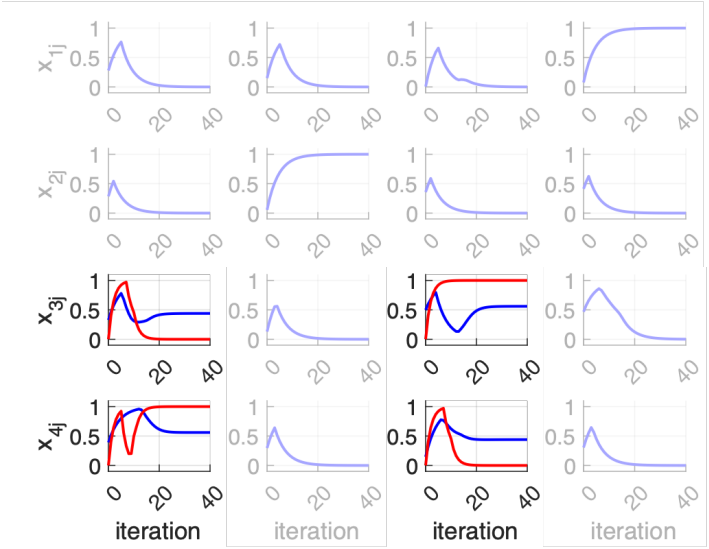


Figure 14. Evolution of the neural network outputs (red curves) for re-solving the ambiguous subproblem of Figure 13. The maximum value assignment of $u_3 \rightarrow v_3$ and $u_4 \rightarrow v_1$ has been disambiguated by the neural network.

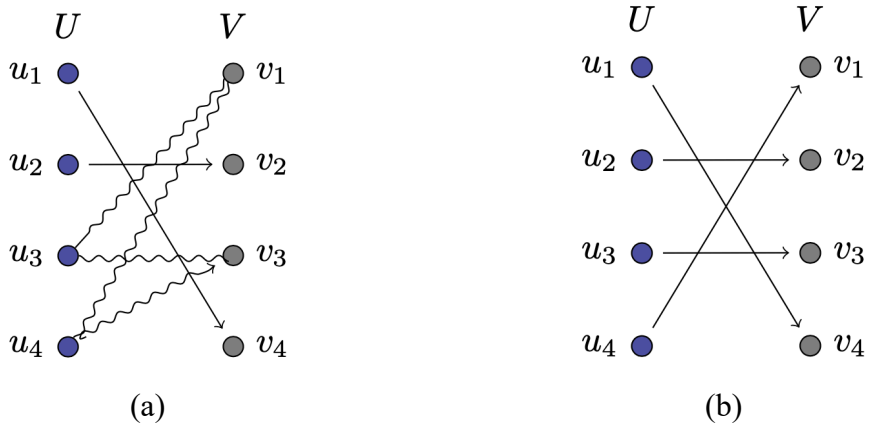


Figure 15. Bipartite graph for the maximum value matching: (a) ambiguous case corresponding to network outputs of Figure 13; (b) disambiguated matching.

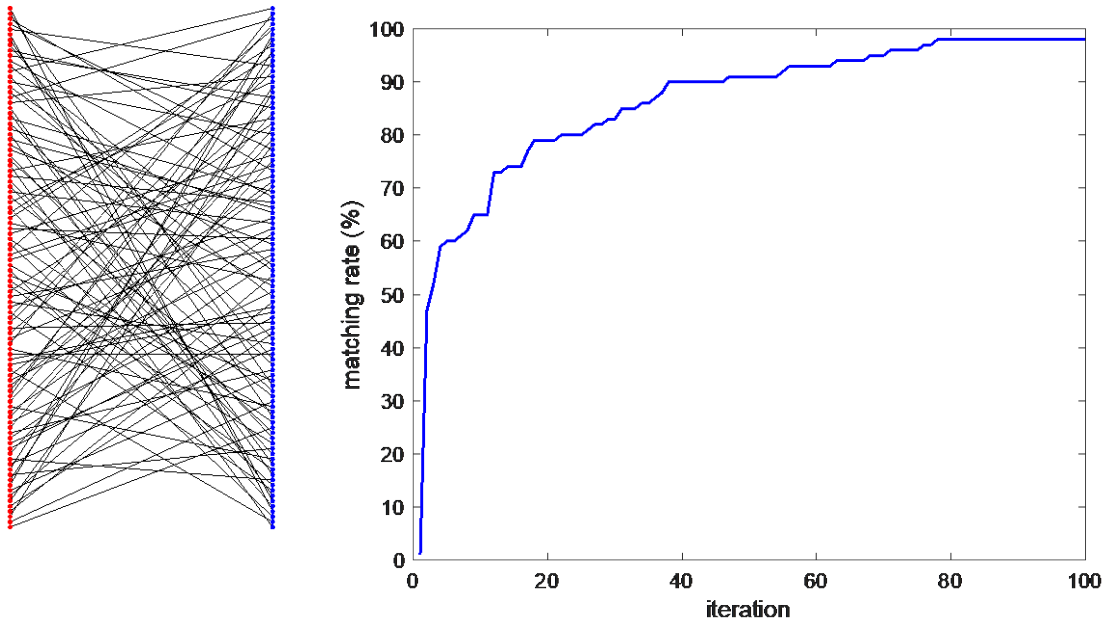
2. Larger Scale Problems

The scalability of the deep network was tested for several additional problems of increasing dimension using datasets available in the open literature [4]. Problem sizes n=m=100 to n=m=800 were tested for minimum cost matching. In each case, it was possible to obtain solutions having cost values in agreement with the expected results from [4] (see Table 1). Figure 16 shows an example bipartite graph solution for n=m=100. The matching rate as a function of the network iterations is also shown in the Figure. As can be seen the neural

network correctly assigned 98% of the input-output pairings on the first pass. The 2% ambiguous cases could be resolved by a second pass through the network as described previously. Similar results were obtained for the other cases tested. Bipartite graphs for n=m=400 and n=m=800 are shown in Figure 17 for reference.

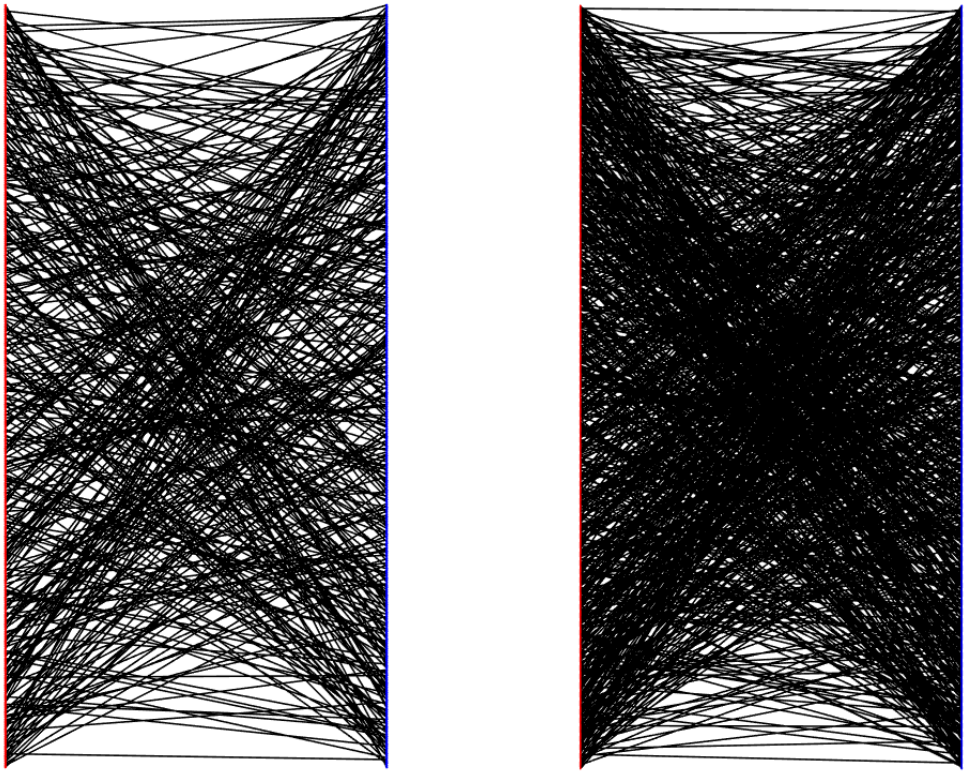
Table 1. Minimum matching cost obtained using neural network for large scale problem data sets.

Problem Size	Expected Cost [4]	Neural Network Cost
100	305	305.0249
200	475	474.9969
300	626	625.9997
400	804	804.0000
500	991	991.0000
600	1176	1176.0000
700	1362	1362.0000
800	1552	1552.0000



(a) (b)

Figure 16. Example solution for larger scale problem n=m=100: (a) bipartite graph for the minimum cost matching; (b) matching rate as a function of iteration.



(a) (b)

Figure 17. Minimum coast bipartite graphs for n=m=400 (a) and n=m=800 (b).

F. LEARNING ACCELERATION

Learning rates are influenced by the number of non-zero network weights, as these influence the connectivity of the network layers. To probe into how network weights can potentially influence learning of the neural network architecture studied here, several weight pruning templates were applied. Some results of the initial study are shown in Figure 18. Figure 18 shows the evolution of the neural network outputs for a small-scale test problem. The magenta-colored curves show the outputs for the baseline configuration. By applying one pruning template, learning was found to become destabilized as indicated by the oscillations in the network outputs (red curves). By applying a different pruning template, the network outputs converged more quickly to the correct output. This is an example of learning acceleration that can be used to improve the performance of the approach in an on-line/time-critical instantiation. In general, there exist a variety of different weight configurations that can be applied to the network. As part of a follow-on study, systematic investigation of this aspect is highly recommended. One potential approach is to apply the recent results of Ross [5].

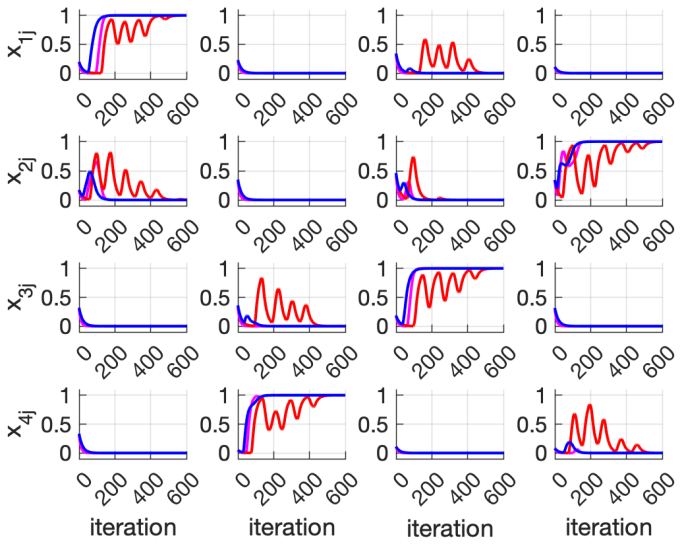


Figure 18. Illustration of the evolution of the neural network outputs for different layer weight configurations. Legend: magenta curves – baseline configuration; red curves – weight configuration destabilizes learning; blue curves – weight configuration for accelerating learning.

G. CONCLUSIONS AND FUTURE WORK

This study explored the question of how machine learning can be applied to identify the most appropriate 'sensor' for completing a 'task' by optimizing a task-to-sensor matching problem. The mathematical concept of a bipartite graph provides a framework for solving this problem. In this report, a conventional concept for bipartite matching was reviewed. Using such a conventional approach, computing time can vary greatly with the dimension and structure of the problem. Using machine learning as an alternative, it is possible to transform the problem statement into a differential equation which can be implemented efficiently to support online/real-time decision making. It is demonstrated that bipartite matching problems can be solved by using a 'weighted' residual neural network architecture. Moreover, the approach is scalable to large dimensional problems. Accelerating the machine learning process was also studied and it was found that different constructions of the neural network weight matrix can be used to speed up learning whereas others might destabilize the learning process. Further investigation into how learning can be accelerated for online/real-time applications is highly recommended. Future work that exercises the concept in the context of DoD specific problem data should also be done. One target area could be drawn from targeting and fires.

H. REEFERNECES

[1] Burkard, R. E., Dell'Amico, M., and Mertello, S. *Assignment Problems*, Society for Industrial and Applied Mathematics: Philadelphia, PA, 2012.

[2] Chen, R. T. Q., Rubanova, Y, Bettencourt, J., and Duvenaud, D., "Neural Ordinary Differential Equations", arXiv, 2018, doi:10.48550/ARXIV.1806.07366.

[3] Goodfellow, I., Bengio, Y., and Courville, A., Deep Learning, MIT Press: Cambridge, MA, 2016.

[4] Beasley, J. E., "Linear Programming on Cray Supercomputers", Journal of the Operational Research Society, vol. 41, no. 2., pp. 133—139, 1990.

[5] Ross, I. M., "An Optimal Control Theory for Accelerated Optimization," arXiv, 2019. Doi: 10.48550/ARXIV.1902.09004.