Deterministic annealing with Potts neurons for multi-robot routing

Jennifer David1 · Thorsteinn Rögnvaldsson1 · Bo Söderberg2 · Mattias Ohlsson1,2

Received: 24 September 2021 / Accepted: 10 April 2022 / Published online: 19 May 2022 © The Author(s) 2022

Abstract
A deterministic annealing (DA) method is presented for solving the multi-robot routing problem with min–max objective. This is an NP-hard problem belonging to the multi-robot task allocation set of problems where robots are assigned to a group of sequentially ordered tasks such that the cost of the slowest robot is minimized. The problem is first formulated in a matrix form where the optimal solution of the problem is the minimum-cost permutation matrix without any loops. The solution matrix is then found using the DA method based on mean field theory applied to a Potts spin model which has been proven to yield near-optimal results for NP-hard problems. Our method is bench-marked against simulated annealing and a heuristic search method. The results show that the proposed method is promising for small-medium sized problems in terms of computation time and solution quality compared to the other two methods.

Keywords Task allocation · Multiple robots · Task-ordering · Deterministic annealing · Approximation method

1 Introduction

Multi-robot system (MRS) is more robust and efficient when compared to single robots for doing advanced and complex tasks. They are very useful in applications like indoor warehouses [23], automated container terminals [25], search and rescue (coverage problems) [4], surveillance [10], etc. However, the success of MRS depends on the team’s goals like coordination, collaboration or cooperation between them. There are different types of MRS problems depending upon the team goals and objectives like multi-robot coordination, multi-robot task allocation, multi-robot path planning, etc. When a MRS problem involves assigning a set of tasks (or goals or targets) to a set of robots, such that a specific objective is achieved while taking constraints into account, it is called the multi-robot task allocation (MRTA) problem [11]. In this paper, we study one of the important MRTA problems termed as the multi-robot routing problem with min–max objective, where several homogeneous robots are required to visit a group of locations to accomplish certain tasks. The robots should be routed in such a way that the maximum cost incurred by any one of the robots is minimized. The robot locations and the task locations are all known and the locations are in the Euclidean plane with or without obstacles. However, the constraints are such that it may be necessary to do some tasks in a specific order. Moreover, we are specifically looking at multiple mobile robots seeking to distribute spatial motion tasks among them. A practical example is the container terminal with stacked containers; it does not make sense to pick a container at the bottom of a stack while others are piled on top of it. The problem is illustrated in Fig. 1. Most of the other multi-robot routing problems found in literature focuses only on minimizing the sum of the make-span of all the robots. In our case, the given tasks should be ordered, grouped and assigned in such a way that the cost for the slowest robot is minimized. This helps in reducing the load on a single robot and distributes the work load evenly among all robots, thus reducing the breakdown of robots due to unbalanced task distribution. Hence, this objective is more practical and useful in many real-world applications.

Given the high relevance in practical robot applications, there are surprisingly few studies of the MRR problem with min–max objective (MRR-MM) [2,3,9,21,24,30,32]. There are other works similar to this problem [5,7,22,23,27] but differ in a number of ways like triangle inequality, returning back to fixed positions, heterogeneous task, etc. A detailed review of the problem can be found in [6]. This is a NP-hard problem with the number of tasks [27] as the time to compute
the solutions grows exponentially with the size of the problem. Researchers have to trade-off between solution quality and the computational time to solve such problems. They use meta-heuristics, hybrid and stochastic algorithms to obtain reasonable solutions. In this paper, we seek to find near-optimal solutions for this problem in a considerable amount of time. So, in this paper, we propose a novel method adapted from the field of statistical physics that can be used to solve this problem. The method is called the deterministic annealing method (DA) using the concepts of mean field theory (MFT) and Potts neurons. This method has been previously applied successfully to other NP-hard problems like graph bisection [29], TSP [16], knapsack [26], air-crew scheduling [20], communication routing [12,13], etc., and has been successful in finding near-optimal deterministic solutions faster than other heuristics approaches.

In this paper, we first formulate the MRR-MM problem in the form of a matrix. The search space is the set of all the permutation matrices of the problem size and the permutation matrix with the lowest cost is the optimal solution. This is the first contribution of this paper where the MRR-MM problem is mapped into a permutation matrix. One can use any search algorithm to find the optimal matrix using this problem formulation (for example, we also apply simulated annealing method to this formulation to compare with our method). We use a deterministic annealing method to search for the permutation matrix by applying mean-field dynamics which produces very good solutions. Secondly, the previous works on DA which was used to solve other NP-hard problems made use of only the Potts vector, whereas we map it as a Potts matrix. This is inspired by ideas in a technical report by Jönsson and Söderberg [17] and we develop the method fully to the MRR-MM problem. This is the second contribution of this paper. Thirdly, we study the DA method’s performance with respect to scalability, optimality and computation time by comparing against two other generic methods: the stochastic—simulated annealing (SA) and a heuristic graph-based method (inspired from Turpin et al. [33]). The DA method shows promising results in terms of computation resources and optimality. The DA method used with Potts Neuron and MFT, when applied to NP-hard problems like the multi-robot routing variants, produces better solutions compared to the general meta-heuristics methods used in the literature. This is the third contribution of this paper.

The rest of the paper is organized as follows: The related work and the background for DA method with Potts neurons and MFT is presented in Sect. 2. The problem formulation is detailed in Sect. 3. The DA method is elucidated in Sect. 4. Detailed experiments and performance characteristics of the method, and benchmarks with other methods, are presented in Sect. 5 followed by concluding remarks in Sect. 6.

2 Background

2.1 MRR-MM problem

There are very few works on the MRR-MM problem and the constraints applied are different for each depending upon the application. For example, Parker [28] studied it with a focus on producing fault tolerant systems; suggesting a behavior-based architecture that enabled robots to cooperate and achieve tasks even if individual robots failed. Arkin et al. [1] worked on a similar problem termed as “min–max k-paths cover problem,” where k paths are found in a graph $G = (V,E)$ covering $V$ vertices and the length of each path is not more than $k$ times the optimal solution. However, in their work, the $k$ paths do not have different start locations (depots) as in our case. Lagoudakis et al. [3,21,30] assumed that the utility costs were symmetric and obeyed the triangle inequality. Lagoudakis et al. developed a theoretical
distributed framework using the auction algorithm that establishes performance guarantees for small to medium-sized problems. While Bae and Park developed a heuristic based on a primal-dual technique for heterogeneous robots. Faigl et al. [9] explored a similar problem where multiple unmanned aerial vehicles (UAVs) were required to visit the given targets in a short span of time. They considered the UAVs to have Dubin models (for respecting the curvature constraints) and a neighborhood visit by the UAVs (instead of the exact location) to adapt it to be used for real-world search and rescue operation. They made use of variable neighborhood search method and extended it to include self-organizing maps to learn the allocation from experience. Their problem differs from our problem in two aspects; it focused on the additional constraints (curvature constraints and neighborhoods) and focused on very fast computation time, instead of optimality. Turpin et al. [33] worked on a similar problem as ours with multiple starting locations for robots and path planning between the robots. They suggested a polynomial-time approximation algorithm with a sub-optimality bound of five times the cost obtained from the method by Arkin et al. [1]. This is the only work that is very closely related to our exact problem (but with an addition of collision avoidance) and focuses on optimality and computation time. So, a simpler form of this method will be used for comparing the results of our method in Sect. 5.

As seen from the literature, some researchers have focused on finding faster solutions rather than optimal solutions and focused on the scalability of their approaches. Some others focus on the practical parameters of the problem like fault-tolerant or dynamic planning of the task allocation with no emphasis on the solution quality or computation time required. So, in this paper, we seek to find optimal (small-size problems) and near-optimal (for large size problems) solutions in a considerable amount of time for the MRR-MM problem. In order to achieve this, we design a deterministic annealing (DA) with mean field theory (MFT) and Potts neurons method to find the global minima of the system. Most of the other related works in the literature traded optimality for computation time or scalability. For a multi-robot task routing scenario, near-optimal solutions can save huge operation cost of the robots, and hence, we focus on finding near-optimal solutions for this NP-hard problem using this novel approach. The DA method has produced near-optimal solutions for small-medium sized problems with shorter computation time than the other methods.

2.2 DA method with Potts neuron

The Hopfield neural network (HNN) [14] was first used to solve the classic traveling salesman problem. Similarly, many optimization problems were solved by mapping it into a Hopfield neural network. If the objective function of the problem can be expressed in terms of a Hopfield energy function $E$, then minimizing this energy function minimizes the objective function. This is because the constraints are “embedded” into the weights of the network. In a HNN, the general form for the energy function used is

$$E(s) = -\sum_{i,j} w_{ij} s_i s_j$$

where $w_{ij}$ is the connection strength between neural units $s_i$ and $s_j$, $s$ denotes the vector with binary $s_i$ values. The connection strengths $w_{ij}$ are designed in such a way that the configuration of $s$ with minimum energy $E(s)$ corresponds to the optimal solution of the problem.

In such an energy function, there are typically many local minima and one should therefore search for the global minima with methods that can escape local minima. Simulated annealing is one such method which uses a temperature $kT$ that controls the ability to escape local minima. As $kT \to 0$, the system is expected to converge to the global minimum.

Hopfield and Tank [15] later introduced a continuous neural unit $v_i \in [0, 1]$ instead of $s_i$ in the HNN. This continuous units $v_i$ denote the probability for the binary unit $s_i$ to have the value 1. This is referred to as deterministic annealing (DA) with MFT or mean field annealing.

Peterson and Söderberg [29] showed that if $v_{ij}$ unit is used instead of $v_i$, it gave more stable convergence. Because it imposes a constraint called “1-out-of-Z” and this Z-state Potts unit is a variable with Z possible states/values. This allows only one unit from a group of Z units to turn ON (1) and the rest of the units to OFF (0).

So, when an optimization problem is expressed as an energy function with the constraints embedded as weights of the Potts units and annealing is done, as $kT \to 0$, the system converges to the global minima giving the optimal solution of the problem.

3 Problem formulation

The requirements of our MRR-MM problem include:

- Every task is done.
- Every robot does at least one task.
- The robots can have different start locations (multiple depots).
- The robots are not required to return to their starting locations (depot) after the task completion. They return to any end locations.
- The utility costs are asymmetric (e.g., the cost for going from task T1 to task T2 is different from going from task T2 to task T1).
So, given \( M \) robots and \( N \) tasks, where \( M \leq N \), and a utility cost between \( M \) robots and \( N \) tasks, the aim is to find an optimal assignment of \( M \) robots to a set of \( N \) tasks such that the total utility cost of the slowest robot is minimized. The utility cost is divided into two as: the transfer costs \( \Delta_{ij} \) between any two consecutive tasks \( i \) and \( j \), expressed in the transfer matrix \( \Delta \), and \( T_i \), the cost for doing a task \( i \) expressed in the task vector \( T \).

A solution square matrix \( S \), of size \((2M + N) \times (2M + N)\), also denoted as the transfer assignment matrix, is defined with binary variables:

\[
s_{ij} = \begin{cases} 
1 & \text{if task } i \text{ is followed by task } j \\
0 & \text{otherwise} 
\end{cases} \quad (2)
\]

Special additional transfers are introduced for moving from the start position of a robot to a task and for going from a final task to a possible end position for a robot. They are labeled \( S \) and \( E \) followed by the robot letter. It is not necessary that the start and end positions of the robots are related to each other. It is, e.g., possible for robot \( X \) to start at \( SX \) and end at \( EY \).

For an example with two robots (\( X \) and \( Y \)) and three tasks (\( a \), \( b \), and \( c \)), the complete set of tasks is \{\( SX, SY, a, b, c, EX, EY \)\}. The matrix \( S \) is of size \( 7 \times 7 \):

\[
S = \begin{bmatrix}
SSX, SX & SSX, SY & SSX, a & SSX, b & SSX, c & SSX, EX & SSX, EY \\
SSX, SX & SSX, SY & SSX, SY & SSX, b & SSX, c & SSX, EX & SSX, EY \\
Sb, SX & Sb, SY & Sb, a & Sb, b & Sb, c & Sb, EX & Sb, EY \\
Sb, SX & Sb, SY & Sb, SY & Sb, b & Sb, c & Sb, EX & Sb, EY \\
Sc, SX & Sc, SY & Sc, a & Sc, b & Sc, c & Sc, EX & Sc, EY \\
Sc, SX & Sc, SY & Sc, SY & Sc, b & Sc, c & Sc, EX & Sc, EY \\
SEX, SX & SEX, SY & SEX, a & SEX, b & SEX, c & SEX, EX & SEX, EY \\
SEX, SX & SEX, SY & SEX, SY & SEX, b & SEX, c & SEX, EX & SEX, EY \\
SEY, SX & SEY, SY & SEY, a & SEY, b & SEY, c & SEY, EX & SEY, EY \\
SEY, SX & SEY, SY & SEY, SY & SEY, b & SEY, c & SEY, EX & SEY, EY
\end{bmatrix} \quad (3)
\]

The element \( s_{a,b} \) is read as “the task \( a \) is followed by task \( b \)”. Many of the transfers in the matrix \( S \) are dummy or undesired, and thus not free to take nonzero values. For instance, the element \( s_{a,SX} \) means “the task \( a \) is followed by going to the start position \( SX \)”, which is not desired and the element should always be zero. Similarly end position transfers like \( s_{EX,a} \) imply going from the end position \( EX \) to a previous task \( a \), which should not happen, and therefore, this element should also always be zero. The element \( s_{SX,EX} \) and similar elements imply a transfer directly from a start position to an end position, without doing any tasks. Also, there are transfers that lead to loops. They can be first-order loops like \( (a, a), (b, b) \), or higher order, which will be discussed later. Undesired transfers and first-order loops are avoided by fixing corresponding matrix elements to zero and the effective transfer assignment matrix looks like this:

\[
S = \begin{bmatrix}
0 & 0 & SSX,a & SSX,b & SSX,c & 0 & 0 \\
0 & 0 & SSY,a & SSY,b & SSY,c & 0 & 0 \\
0 & 0 & s_{a,b} & s_{a,c} & s_{a,EX} & s_{a,EY} & 0 \\
0 & 0 & s_{b,a} & s_{b,c} & s_{b,EX} & s_{b,EY} & 0 \\
0 & 0 & s_{c,a} & s_{c,b} & s_{c,EX} & s_{c,EY} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix} \quad (4)
\]

Thus, even though the full transfer assignment matrix is \( 7 \times 7 \), the effective matrix (the sub-matrix that changes) is a permutation matrix\(^1\) of size \( 5 \times 5 \). This matrix has 24 solutions. For example, one solution is,

\[
S = \begin{bmatrix}
SX & SY & a & b & c & EX & EY \\
SX & 0 & 0 & 0 & 1 & 0 & 0 \\
SY & 0 & 0 & 1 & 0 & 0 & 0 \\
a & 0 & 0 & 0 & 0 & 1 & 0 \\
b & 0 & 0 & 0 & 0 & 0 & 1 \\
c & 0 & 0 & 0 & 0 & 0 & 0 \\
EX & 0 & 0 & 0 & 0 & 0 & 0 \\
EY & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

This corresponds to \( SX \rightarrow b \rightarrow EX \) & \( SY \rightarrow a \rightarrow c \rightarrow EY \), which is read as “the robot \( X \) goes from start position, does task \( b \) and then goes to end position \( X \)”, and “the robot \( Y \) goes from start position, does task \( a \) followed by task \( c \), and then goes to end position \( Y \)”. The solution cost of this matrix can be found by using the values of the \( S \) matrix applied on the transfer cost matrix \( \Delta \) and task vector \( T \), similar to the manner used in the Hungarian algorithm [18], or using the transport cost design variable that is discussed in Sect. 4.2.2.

The problem is now encoded in a matrix and the search space is the list of all permutation matrices of the given size. The optimal solution to this problem is the valid permutation matrix with the minimum cost. Some of the permutation matrices will not correspond to valid solutions due to the presence of loops, i.e., a robot will eventually be routed to an already done task. Permutation matrices with loops must be avoided. In our example above, with a \( 5 \times 5 \) sub-matrix, there are 120 permutation matrices, of which 24 are valid without any loops and each robot does at least one task. The optimal solution is the valid solution matrix with the minimum cost. One can use any search method to find this “valid and minimum cost” permutation matrix, i.e., the optimal solution matrix.

\(^1\) The permutation matrices are the extreme points of the set of doubly stochastic matrices.
4 Methodology

In order to apply the DA method on this problem formulation, we do the following steps:

1. Convert \( S \) matrix into Potts transfer assignment matrix \( V \).
2. Design variables to \( a \). detect loop matrices using \( V \) matrix and to \( b \). find minimum cost permutation matrix.
3. Derive the local field energy function using the design variables.
4. Substitute this field energy function in the standard Potts matrix equation.
5. Tune an annealing schedule for reducing this energy function to find the global minima.

4.1 Potts transfer assignment matrix

The \( V \) matrix consists of continuous units \( v_{ij} \in [0, 1] \), and has the same size as the \( S \) matrix. The Potts unit \( v_{ij} \) represents a thermal average value for \( s_{ij} \) during an annealing process. Just like the \( S \) matrix, the \( V \) matrix has an effective sub-matrix that is doubly stochastic.

4.2 Design variables

4.2.1 The propagator matrix

To detect the matrices with loops and thereby impose an extra penalty for such solutions in the search, we introduce the propagator matrix \( P \), as suggested by Lagerholm et al. [19].

\[
P = (I - V)^{-1} = I + V + V^2 + V^3 + V^4 + \cdots \quad (5)
\]

The propagator matrix element \( p_{ij} \) expresses the number of paths that lead from task \( i \) to task \( j \). This can be understood from the Neumann series expansion in (5): the matrix elements in \( V \) tell the paths from one task to another task directly, the elements in \( V^2 \) tell the paths from one task to another task via an intermediate task, the elements in \( V^3 \) correspond to counting the paths between two tasks via two intermediate tasks, and so on and so forth. At least one propagator matrix element will approach infinity if \( V \) moves toward a solution with loops and the propagator matrix can be used to impose a penalty on the \( V \) matrix for loops.

4.2.2 Minimum cost

To compute the solution cost of a \( V \) matrix, two vectors \( L \) and \( R \) are introduced. The element \( L_i \) is the cost for reaching task \( i \) from the left, i.e., going from any start position to task \( i \) (including doing task \( i \)). The element \( R_i \) is the cost for reaching task \( i \) backward, i.e., to do task \( i \) and then continue from task \( i \) to any end position. The make-span for the robots is the run-time for the slowest robot, i.e., the one that has largest cost to finish its tasks, and the transport cost \( C \) (the make-span) is defined as the average of these two,

\[
C = \frac{1}{2} (L_a + R_b) \quad (6)
\]

where the indices \( \alpha \) and \( \beta \) correspond to the components in \( L \) and \( R \) with maximum values (only considering the last \( M \) elements in \( L \) and the first \( M \) elements in \( R \)). Thus, the solution cost of a Potts transfer assignment matrix \( V \) can be found using Eqs. (14), (15) and (6) (detailed in Appendix A).

In the following section, the deterministic annealing method is derived that makes use of these two design variables, the Potts \( V \) matrix, and an annealing schedule to find optimal solution for this problem.

4.3 The local field energy

The Potts neuron \( v_{ij} \) represents a transfer; doing task \( j \) after task \( i \). For each of the Potts neurons \( v_{ij} \), a corresponding local field \( U_{ij}^{\text{local}} \) is developed with three terms: the transport cost field \( U_{ij}^{\text{trans}} \) (to minimize the maximum of the transport costs), the loop cost field \( U_{ij}^{\text{loop}} \) (to suppress loop solutions), and the stabilization field \( U_{ij}^{\text{stable}} \) (to stabilize the system).

The transport cost field is computed by taking the derivative of expression (6) with respect to the Potts unit \( v_{ij} \),

\[
U_{ij}^{\text{trans}} = \frac{1}{2} \left[ \frac{\partial L_\alpha}{\partial v_{ij}} + \frac{\partial R_\beta}{\partial v_{ij}} \right] = \frac{1}{2} \left[ p_{ji} (L_i + \Delta_{ij}) + p_{ij} (R_j + \Delta_{ij}) \right] \quad (7)
\]

where \( \alpha \) and \( \beta \) correspond to the indices of the components in \( L \) and \( R \) with maximum values, respectively (considering only the last \( M \) elements in \( L \) and the first \( M \) elements in \( R \)).

The loop field is the same as introduced in Häkkinen et al. [13], which adds an extra penalty to avoid solution matrices with loop (i.e., repetitive tasks):

\[
U_{ij}^{\text{loop}} = \frac{p_{ji}}{p_{ij} - p_{ji}} \quad (8)
\]

The stabilization field helps to avoid oscillations in the DA updating due to the additional challenge of normalization. We use the same damping term as suggested by Eberhard et al. [8], which also corresponds to a soft normalization cost on the rows and columns:

\[
U_{ij}^{\text{stable}} = -v_{ij} \quad (9)
\]
The complete local field for the MFT update is thus

$$U_{ij}^\text{local} = U_{ij}^\text{trans} + \gamma U_{ij}^\text{loop} + \eta U_{ij}^\text{stable}$$

$$= \frac{1}{2} \left[ p_i (L_i + \Delta_{ij}) + p_j (R_j + \Delta_{ij}) \right]$$

$$+ \gamma \frac{p_i}{p_j} - \eta v_{ij}$$

(10)

where $\gamma$ and $\eta$ are the positive weighting factors for the loop and stabilization terms, respectively.

### 4.4 Potts matrix

Now, that we have the local field for the MFT update, we need to update the “Potts” matrix using the equation,

$$v_{ij} = \frac{e^{-\frac{U_{ij}^\text{local}}{kT}}}{\sum_m e^{-\frac{U_{mj}^\text{local}}{kT}}}$$

(11)

This problem is trickier than previous Potts MFT approaches since they were able to formulate the problem as a one-dimensional Potts vector but we have a two-dimensional Potts matrix due to the nature of our problem. Our Potts matrix should be maintained as doubly stochastic even after the update. So, an additional external normalization is required to ensure that the matrix is doubly stochastic. This is done using Sinkhorn–Knopp normalization [31] for a certain number of times, say $w$, until the $V$ matrix becomes doubly stochastic to a good approximation. This normalization, iteratively switching between normalizing rows and columns, is guaranteed to converge (albeit sometimes very slowly) to a doubly stochastic matrix if the matrix is nonnegative.

### 4.5 Annealing period

The DA process runs for a pre-defined annealing period, with an initial temperature $kT_{\text{start}}$, a final temperature $kT_{\text{stop}}$ and a factor $kT_{\text{factor}}$ for decreasing the temperature. All these parameters require proper tuning. At some point, the system reaches a critical temperature $kT_c$, where a phase transition takes place. Here, the system undergoes rapid changes and converges toward a solution, which is illustrated in Fig. 2. It is important to set the initial temperature $kT_{\text{start}} > kT_c$, but not by much since that leads to unnecessary long convergence time. The annealing factor $kT_{\text{factor}}$ gives a smooth decrease in temperature for the system to eventually go on-shell. It is beneficial to use an adaptive cooling schedule where the cooling rate is decreased during the transition phase to allow better convergence to solutions. In our case, the adaptive cooling is set up based on the value $dV_{\max}$. This is the largest absolute change in value of a single element of the $V$ matrix in the latest update. The exponential decrease in $kT$ is done by the adaptive annealing factor $kT_{\text{factor}}$, based on $dV_{\max}$, such that it lies within the predefined range $[kT_{\text{min}}^\text{fac}, kT_{\text{max}}^\text{fac}]$, as described in Algorithm 1.

### 4.6 DA algorithm

The DA method is outlined in Algorithm 1. The MRR-MM problem objective is defined with $M$ number of robots and $N$ number of tasks, with the utility costs expressed in the $\Delta$.
matrix and \(T\) vector. The algorithm begins by initializing a Potts matrix \(V\) of size \(n \times n\), where \(n = 2M + N\). After setting the unwanted transfers of the matrix to zero (the first \(M\) columns and the last \(M\) rows), the effective \(V\) matrix will be of the size \((M + N) \times (M + N)\). Now, each element of this sub-matrix is set to a nonzero random value \(\frac{4}{\pi + 5}\), where \(\epsilon\) is a very small value such that \(\sum_i v_{ij} = 1\) and \(\sum_j v_{ij} = 1\). After initializing the \(V\) matrix (shown in line no 2 of Algorithm 1), the design variables - propagator and the transport costs are calculated in line no 3. These variables are then used to find the transport cost field, stabilization field and the loop field which are then used to find the local field in line no 4. The iteration begins by setting the system to the predetermined start temperature \(kT_{\text{start}}\). After every iteration, the temperature is decreased by a factor \(kT_{\text{factor}}\), and the Potts units are updated according to the local field Eq. (10) line no 4, and Potts MFT, Eq. (11) line no 7. The resulting \(V\) matrix (line no 7) is a non-normalized matrix and an external Sinkhorn normalization is done after each update to ensure that the \(V\) matrix is (approximately) doubly stochastic (line nos 8–10). Note that it is only the effective sub-matrix in \(V\) that is updated and normalized to be doubly stochastic.

The process continues, reducing the temperature by an adaptive factor \(kT_{\text{factor}}\) (line no 13 and 14) after every iteration. Hence, when the system goes through the phase transition period, the adaptive cooling slows down the system leading to better convergence of solutions. A solution is considered to be reached if \(M + N\) values in the Potts matrix are greater than 0.9 and the remaining variables are less than 0.1 (line no 16).

4.7 Parameter tuning

Proper values for the weighting factors \(\eta\) and \(\gamma\), the parameters \(kT_{\text{start}}\) and \(kT_{\text{stop}}\), and the annealing factor range, \([kT_{\text{min}}^{\text{fac}}, kT_{\text{max}}^{\text{fac}}]\), are all found by fine-tuning using trial-and-error methods. It is a simple and exhaustive method, where a parameter value is chosen based on experience and it is slowly adjusted by keeping other parameters to a constant value. The \(kT_{\text{start}}\) and \(kT_{\text{stop}}\) values are chosen such that the phase transition happens (the start and the end) smoothly. The \(kT_{\text{min}}^{\text{fac}}\) and \(kT_{\text{max}}^{\text{fac}}\) values are chosen based on the rate of solution convergence desired which again depends on our available computational resource. The three field functions are given equal weights, and hence, \(\eta\) value and \(\gamma\) value are also set to 1. Increasing their values will reduce the quality of the solutions. Also, \(\eta\) value set to 1 at all times enforces that the system is stable. The \(\gamma\) value is also set to 1 to ensure there is no loop solutions. However, for small size problems, the \(\gamma\) value can be set to a value less than 1 as there are not many loop solutions.

### Algorithm 1: Deterministic Annealing Algorithm

| Data        | \(M, N, T, \Delta\) |
|-------------|---------------------|
| Result      | Permutation matrix \(V\) with no loops and an optimal transport cost |
| 1 | Set \(kT_{\text{start}}, kT_{\text{stop}}, kT_{\text{fac}}^{\text{min}}, kT_{\text{fac}}^{\text{max}}(0.99), \eta\) and \(\gamma\) |
| 2 | Initialize \(V\) matrix |
| 3 | Calculate  |
| i. | \(P = (I - V)^{-1}\)  |
| ii. | \(L_i\) and \(R_i\)  |
| iii. | \(U_{ij}^{\text{loop}} = \frac{p_{ij}}{p_{ii} - p_{jj}}\)  |
| iv. | \(U_{ij}^{\text{trans}} = \frac{1}{2}[p_{ii}(L_i + \Delta_{i}) + p_{jj}(R_j + \Delta_{j})]\)  |
| v. | \(U_{ij}^{\text{stable}} = \eta v_{ij}\)  |
| 4 | Update \(U_{ij}^{\text{local}} = U_{ij}^{\text{trans}} + \gamma U_{ij}^{\text{loop}} - \eta U_{ij}^{\text{stable}}\)  |
| 5 | Set \(kT = kT_{\text{start}}\)  |
| 6 | while \(kT > kT_{\text{stop}}\), do  |
| 7 | Update \(v_{ij} = \frac{\exp(-U_{ij})/kT}{\sum_i e^{-U_{ij}}/kT}\) in \(V\) matrix  |
| 8 | while \(\sum_i v_{ij} < 0.95\) for some \(j\), or \(\sum_j v_{ij} < 0.95\) for some \(i\), do  |
| 9 |  Sinkhorn–Knopp normalization  |
| 10 | end  |
| 11 | Repeat steps 3 and 4  |
| 12 | Calculate \(dV_{\text{max}}\)  |
| 13 | Set the adaptive annealing factor \(kT_{\text{factor}} = kT_{\text{min}}^{\text{fac}} + (kT_{\text{max}}^{\text{fac}} - kT_{\text{min}}^{\text{fac}}) \times \tanh(dV_{\text{max}})\)  |
| 14 | Update the temperature \(kT = kT * kT_{\text{factor}}\)  |
| 15 | end  |
| 16 | Print \(V\) matrix |

5 Experiments

5.1 Implementation details

This section outlines the details of the experiments where our method is compared with two other methods—simulated annealing and a heuristic method. The utility costs \(\Delta\) and \(T\) are randomly generated using a uniform distribution. The DA and SA methods require the annealing parameters \(kT_{\text{start}}, kT_{\text{stop}},\) and \(kT_{\text{factor}}\). The DA algorithm also requires two additional parameters to be tuned: \(\gamma\), the weighting factor for the \(U_{ij}^{\text{loop}}\) function, and \(\eta\), the weighting factor for the \(U_{ij}^{\text{stable}}\) function. All three algorithms\(^2\) are implemented using C++ with Eigen library and MATLAB running on Ubuntu 16.04 with octa-core Intel i7 processor and 15.6 GB memory. The DA code is based on the algorithm originally written for the communication routing problem by Häkkinen et al. [13].

5.2 Scalability comparison

An extensive study was done to compare the scalability of the three approaches. In the first experiment, shown in Figs. 3 and 4, the number of robots \((M)\) was kept constant \((M = 10)\) and

\(^2\) [https://github.com/jenniferdavid/taskallocation/tree/master/potts]
5.2.1 Optimality and computation time

In the first set of experiments, $M$ was kept constant and $N$ varied. The slow cooling rate for SA (0.999) was only tested for small problems; for bigger problems the convergence was too slow. Figure 3 shows the performance of the three methods in terms of solution cost and computation time (in log scale) for $N = 10, 20, \ldots, 200$. Figure 4 shows the same performance metrics for larger problem sizes like $N = 10, 20, \ldots, 780$. For very small problem sizes (i.e., few tasks per robot), SA is a good algorithm. However, as the number of tasks per robot increases, DA clearly outperforms SA in terms of solution quality and computation time. Moreover, DA has a smaller distribution of final cost for the solutions compared to SA (results not shown). The second set of experiments, where $N$ was kept constant and $M$ varied, reconfirmed the findings (Figs. 4 and 5).

The heuristic method outperforms the other methods in terms of computation time by a wide margin. For bigger problems, i.e., when the number of tasks per robot is large, it is also better than the other methods in terms of solution cost. DA is the second choice after the heuristic method as
the trade-off between solution cost and computation time is far better than SA (at any annealing rate).

The results from the third set of experiments are listed in Table 1, which shows the performance of SA at different annealing rates compared with DA and the heuristic approach. The Δ matrices differed more between the tests in experiment three than in the other two experiments, and the results are less clear. In terms of final cost, the heuristic method is better for large problems and the DA method is often best for smaller (but not very small) problems. For very small problems, SA is the method of choice. The heuristic method is always the quickest.

5.3 Annealing rate

The effect of annealing rate for SA and DA is illustrated in Fig. 6. The problem size is $M = 10$ and $N = 200$, where the utility costs are generated from a uniform random distribution. The heuristic method is also shown for reference. Figure 6 shows how the final solution cost varies with the computation time for annealing rates ($kT_{\min}$) 0.9, 0.95, 0.99, 0.992 and 0.999. Both SA and DA yield better solutions with slower cooling rate.

5.4 The $\gamma$ and $\eta$ weight factors

The loop weight factor, $\gamma$, determines the relative importance given to the loop cost in comparison to the other two costs. When $M = N$ with the requirement that each robot does at least one task, the problem is reduced to a linear assignment problem based on a min–max objective and the loop cost is irrelevant. So, in this case, $\gamma$ is set to zero as the search space is comparatively small and it allows the DA algorithm to focus fully on the transport cost. However, when the number of tasks grows in relation to the number of robots, e.g., if $M/N < 0.3$, $\gamma$ should be set at 1 to avoid loop solutions. If $\gamma > 1$ then the final solution costs ended up too large. The normalization weight factor $\eta$ was set to 1 in all experiments, which avoided oscillations. If $\eta > 1$ then the algorithm converged too fast and the solutions were sub-optimal.
5.5 Normalization time

Jönsson and Söderberg [17] used Sinkhorn normalization to get a doubly stochastic matrix when working with the “Double Potts Approach.” We also adapt the Sinkhorn normalization [34] for the same process. But it is the most expensive step of the DA algorithm, contributing to more than 90% of the total computation time. The normalization accuracy was set to 0.95 (instead of 1), i.e., the row and column normalization were repeated until all row and column units sum to above 0.95 (some would sum to > 1). If the accuracy of the normalization step was reduced (i.e., below 0.95), the computation time was significantly reduced but at the cost of losing solution quality. For example, if the normalization accuracy was set at 0.9 then the computation time decreased by 30% but the solution quality degraded by 20%.

5.6 Solutions

5.6.1 $M/N$ ratio

As the problem size increases, the solution space also increases and the problem generally becomes harder. However, the ratio of the number of robots to the number of tasks ($M/N$) has a significant impact. Figure 7 illustrates how the solution quality depends on the ratio $M/N$ for a wide range of problem sizes. If the number of tasks per robot is low, then all methods find the minimum cost (albeit with different computation time). If the number of tasks per robot grows,
DA and the heuristic algorithm find lower cost solutions than SA, and the difference between worst and best grows.

5.6.2 Distribution of solutions

To estimate how good the annealing methods were at finding low cost solutions, the costs for all valid solution matrices \( \mathbf{V} \) for two specific problem sets were compared to the solution set obtained with SA and DA at different annealing rates. Figure 8 shows this for a problem with \( M = 2 \) and \( N = 4 \), with 144 valid solutions. SA was run with annealing rates 0.9, 0.92, 0.95, 0.97, and 0.999 and DA was run with annealing rate 0.9. All methods found the optimal solutions with cost 67. Figure 9 shows a case where \( M = 5 \) and \( N = 6 \), with 432,000 valid solutions, of which 64,946 were used for the histogram. SA was run with annealing rates 0.9, 0.99, and 0.999 and DA was run with an annealing rate of 0.9. The lowest cost was 78.46, obtained with SA and an annealing rate of 0.999 (lowest cost with the heuristic method was 117.9 and lowest with DA was 95.08). The examples show that the
Table 1  Comparison of all the three approaches for distinct problems of sizes varying from 5×5 to 145×145

| M | N | COST SA 0.9 | COST SA 0.99 | COST DA 0.9 | COST HEURI | TIME SA 0.9 | TIME SA 0.99 | TIME DA 0.9 | TIME HEURI |
|---|---|-------------|-------------|-------------|------------|-------------|-------------|-------------|------------|
| 1 | 3 | 37.85 | 37.85 | 37.85 | 37.84 | 37.84 | 0.21 | 1.93 | 17.46 | 0.12 | 0.002 |
| 1 | 5 | 120.85 | 150.85 | 150.85 | 215.5 | 1.93 | 32.09 | 32.09 | 0.21 | 0.004 |
| 2 | 3 | 56.46 | 56.46 | 56.46 | 61.52 | 61.52 | 0.4 | 3.4 | 32.7 | 0.24 | 0.003 |
| 2 | 4 | 89.99 | 89.99 | 89.99 | 99.48 | 99.48 | 3 | 34 | 40.3 | 0.276 | 0.003 |
| 3 | 3 | 73.28 | 73.28 | 73.28 | 99.93 | 99.93 | 0.5 | 5.2 | 49 | 0.81 | 0.003 |
| 5 | 5 | 123.57 | 109.44 | 109.44 | 123.78 | 169.83 | 0.51 | 3.27 | 44.89 | 0.325 | 0.004 |
| 3 | 4 | 80.8 | 80.8 | 80.8 | 90.62 | 90.62 | 0.4 | 4.4 | 41.42 | 0.952 | 0.004 |
| 2 | 6 | 22.94 | 21.18 | 21.18 | 22.83 | 22.83 | 0.47 | 6.68 | 51.7 | 0.77 | 0.006 |
| 1 | 8 | 228.65 | 162.23 | 162.23 | 208.63 | 208.63 | 0.59 | 4.6 | 46.7 | 0.396 | 0.014 |
| 2 | 8 | 457.79 | 419.69 | 419.69 | 493.75 | 493.75 | 0.42 | 4.4 | 46.08 | 1.15 | 0.009 |
| 1 | 10 | 266.59 | 208.29 | 168.78 | 178.05 | 178.05 | 0.38 | 4.3 | 54.53 | 0.587 | 0.017 |
| 3 | 6 | 93 | 90 | 90 | 115.46 | 115.46 | 0.4 | 6.7 | 65.85 | 2.35 | 0.027 |
| 4 | 5 | 102.52 | 94.56 | 94.56 | 115.46 | 115.46 | 0.6 | 6.1 | 64.78 | 1.1 | 0.005 |
| 3 | 8 | 373.93 | 349.09 | 349.09 | 356.54 | 356.54 | 0.67 | 6.23 | 59.6 | 1.68 | 0.0135 |
| 2 | 12 | 404.71 | 369.47 | 369.47 | 383.75 | 383.75 | 0.67 | 6.7 | 65.85 | 2.35 | 0.027 |
| 1 | 15 | 485.36 | 317.9 | 219.25 | 275.89 | 275.89 | 0.7 | 6.9 | 66.6 | 1.14 | 0.051 |
| 3 | 12 | 98 | 82.4 | 76.05 | 115.46 | 115.46 | 0.6 | 6.1 | 64.78 | 1.1 | 0.005 |
| 4 | 10 | 98.95 | 72.01 | 57.56 | 67.48 | 67.48 | 1.05 | 9.17 | 180.25 | 2.49 | 0.02 |
| 1 | 8 | 280.16 | 191.08 | 156.75 | 222.40 | 222.40 | 0.7 | 8.3 | 85 | 4.25 | 0.082 |
| 2 | 10 | 640.51 | 379.19 | 265.81 | 339.98 | 339.98 | 0.76 | 7.8 | 89.65 | 1.83 | 0.103 |
| 3 | 18 | 334.63 | 261.17 | 223.35 | 308.53 | 308.53 | 1.02 | 10.45 | 103.5 | 5.32 | 0.063 |
| 2 | 25 | 801.35 | 499.44 | 331.47 | 473.89 | 473.89 | 1.12 | 9.56 | 110.15 | 2.69 | 0.19 |
| 3 | 29 | 29 | 82.4 | 76.05 | 141.77 | 141.77 | 2 | 8.2 | 63.66 | 3.11 | 0.03 |
| 4 | 10 | 98.95 | 72.01 | 57.56 | 67.48 | 67.48 | 1.05 | 9.17 | 180.25 | 2.49 | 0.02 |
| 1 | 20 | 640.51 | 379.19 | 265.81 | 339.98 | 339.98 | 0.76 | 7.8 | 89.65 | 1.83 | 0.103 |
| 2 | 25 | 801.35 | 499.44 | 331.47 | 473.89 | 473.89 | 1.12 | 9.56 | 110.15 | 2.69 | 0.19 |
| 4 | 30 | 905.73 | 723.72 | 527.82 | 652.10 | 652.10 | 1.09 | 11.5 | 134 | 5.87 | 0.19 |
| 5 | 40 | 255 | 102 | 100.56 | 126.3 | 126.3 | 1.34 | 12.2 | 120 | 7.28 | 0.198 |
| 6 | 42 | 255 | 102 | 100.56 | 126.3 | 126.3 | 1.34 | 12.2 | 120 | 7.28 | 0.198 |
| 7 | 40 | 480.47 | 339.98 | 286 | 356.43 | 356.43 | 1.34 | 12.2 | 120 | 7.28 | 0.198 |
| 8 | 50 | 490 | 216.48 | 144.61 | 162.85 | 162.85 | 1.34 | 12.2 | 120 | 7.28 | 0.198 |
| 9 | 55 | 471.52 | 217.6 | 117.97 | 133 | 133 | 1.34 | 12.2 | 120 | 7.28 | 0.198 |
| 10 | 80 | 514.52 | 449.17 | 361.97 | 534.65 | 534.65 | 1.73 | 15.99 | 151.32 | 8.6 | 0.225 |
| 11 | 105 | 342.62 | 249.52 | 161.8 | 217.38 | 217.38 | 1.14 | 13 | 132 | 6.99 | 0.199 |

The blue color shows the best results followed by green color for second best.

Still, Table 1 shows that the heuristic does not always perform better than the DA method in terms of solution cost. SA at annealing rate 0.9 is fast but gives very poor solutions. With a slow annealing rate (like 0.99 or 0.999), solutions are comparatively better but they come with a high computation cost. The problem size and $M/N$ ratio affect SA much more than the other methods.

5.7 Advantages and disadvantages

It is evident from Figs. 4 and 5 that the heuristic method performs better than the other two methods by a large margin.

The blue color shows the best results followed by green color for second best.
A difficulty with the DA method is tuning the parameters, especially $\eta$ and $\gamma$, which affect the solution quality (this is commented by Söderberg and Jönsson [17]). Some tests are always required to tune these parameters. When it comes to the annealing period, SA and DA get improved quality of the solution for slower annealing rate but DA always yields better solution than SA at the same annealing rate. Another important factor to be noted is that the external normalization required for DA means a considerable extra computational burden. When it comes to the heuristic method, it does not explore all possible solutions and the solution quality highly depends on how the discrete edges are found in the multidigraph and how they are cut.

6 Conclusions

This paper presents a novel deterministic annealing method based on mean-field theory and the Potts model for solving the multi-robot routing problem with min–max objective. A novel feature is the generic formulation of this problem in terms of a permutation matrix. Another feature includes the application of a Potts matrix for defining the problem, instead of the Potts vector case. A dedicated mean-field equation is derived that minimizes the maximum cost of the robots (i.e., the robot that takes longest time). By use of Sinkhorn normalization and the path propagator, the method converges to good quality solutions. As a side note: Jönsson and Söderberg [17] tried some preprocessing techniques to improve the convergence at lower temperatures, which did lead to shorter computation times. Applying such methods would likely make the DA method faster here as well, but we opted to use only the Sinkhorn normalization.

Extensive experiments are conducted by comparing this method to two other methods: simulated annealing (based on the same problem formulation) and a heuristic method. Deterministic annealing produces better results than simulated annealing for comparable convergence times if the number of tasks is large compared to the number of robots. The heuristic method outperforms both simulated and deterministic annealing in terms of computation speed. For problems with few robots, deterministic annealing sometimes produces better solutions than the heuristic method. However, for problems with many robots and many tasks, the heuristic method is a clear winner. The future work involves extending this problem formulation to linear assignment and scheduling problem, and adapting the problem formulation and method accordingly.

Acknowledgements The C++ implementation used for the DA algorithm is based on the code developed by Jari Häkkinen et al. [13] in their work on communication routing.

Funding Open access funding provided by Halmstad University. Not applicable.

Data Availability Not applicable.

Code availability The source code for this algorithm is available in GitHub (link provided above).

Declarations

Conflict of interest Not applicable.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

Appendix A

The elements $L_i$ and $R_i$ can be computed by iterating the following relations to say, $k$, times:

$$L_i(k) = \sum_j v_{ji} \left[ L_j(k-1) + \Delta_{ji} + T_i \right]$$

$$R_i(k) = \sum_j v_{ij} \left[ R_j(k-1) + \Delta_{ij} + T_i \right]$$

and see which values they converge to when all $L_i$ and $R_i$ are set to zero from the start, i.e., $L_i(0) = 0$ and $R_i(0) = 0$ for all $i$. Writing them in matrix form and iterating them to $k \to \infty$ leads to the expressions

$$L = P^T \left[ T + \text{diag} \left( V^T \Delta \right) \right]$$

$$R = P \left[ T + \text{diag} \left( V \Delta^T \right) \right]$$

where $P$ is the propagator matrix and $\text{diag}(X)$ denotes the column vector with the diagonal elements from the matrix $X$.

For our previous $7 \times 7$ example, we have two vehicles, $A$ and $B$. The left vector is

$$L = \begin{pmatrix} L_{SX} \\ L_{SY} \\ L_a \\ L_b \\ L_c \\ L_{EX} \\ L_{EY} \end{pmatrix}$$
and the right vector is

\[
\mathbf{R} = \begin{pmatrix} R_{SX} \\ R_{SY} \\ R_a \\ R_b \\ R_c \\ R_{EX} \\ R_{FY} \end{pmatrix}
\]

(17)

Thus, the robots’ estimated run-times are the last \( M \) elements in \( \mathbf{L} \) (estimated from the start to the end) and the first \( M \) elements in \( \mathbf{R} \) (estimated from the end to the start).

References

1. Arkin EM, Hassan R, Levin A (2006) Approximations for minimum and min-max vehicle routing problems. J Algorithms 59(1):1–18
2. Bae J, Lee J, Chung W (2019) A heuristic for task allocation and routing of heterogeneous robots while minimizing maximum travel cost. In: 2019 international conference on robotics and automation (ICRA), pp. 4531–4537. https://doi.org/10.1109/ICRA.2019.8794257
3. Bae J, Park M (2021) A heuristic for efficient coordination of multiple heterogeneous mobile robots considering workload balance. IEEE Robot Automat Lett 6(2):4064–4070
4. Baxter JL, Burke E, Garibaldi JM, Norman M (2007) Multi-robot search and rescue: a potential field based approach. In: Autonomous robots and agents, pp 9–16. Springer
5. Bektaš T (2006) The multiple traveling salesman problem: an overview of formulations and solution procedures. Omega 34(3):209–219
6. David J, Rögnvaldsson T (2021) Multirobot routing problem with min-max objective. Robotics 10(4):122
7. Ding L, Zhao D, Ma H, Wang H, Liu L (2018) Energy-efficient min–max planning of heterogeneous tasks with multiple uavs. In: 2018 IEEE 24th international conference on parallel and distributed systems (ICPADS), pp 339–346. https://doi.org/10.1109/PADSW.2018.8794257
8. Eberhardt SP, Daud T, Kerns D, Brown TX, Thakoor A (1991) Competitive neural architecture for hardware solution to the assignment problem. Neural Netw 10(4):431–442
9. Faigl J, Váňa P, Pěnička R, Saska M (2019) Unsupervised learning-based flexible framework for surveillance planning with aerial vehicles. J Field Robot 36(1):270–301
10. Fisher ML, Jaikumar R (1981) A generalized assignment heuristic for vehicle routing. Networks 11(2):109–124
11. Gerkey BP, Mataric MJ (2003) Multi-robot task allocation: Analyzing the complexity and optimality of key architectures. In: 2003 IEEE international conference on robotics and automation (Cat. No. 03CH37422), vol 3, pp 3862–3868. IEEE
12. Häkkinen J, Lagerholm M, Peterson C, Söderberg B (1998) A Potts neuron approach to communication routing. Neural Comput 10:1587–1599
13. Häkkinen J, Lagerholm M, Peterson C, Söderberg B (2000) Local routing algorithms based on Potts neural networks. IEEE Trans Neural Netw 11(4):970–977
14. Hopfield JJ (1982) Neural networks and physical systems with emergent collective computational abilities. Proc Natl Acad Sci 79(8):2554–2558
15. Hopfield JJ, Tank DW (1985) Neural computation of decisions in optimization problems. Biol Cybern 52(3):141–152
16. Ishii S, Ma Sato (1997) Chaotic Potts spin model for combinatorial optimization problems. Neural Netw 10(5):941–963
17. Jönsson H, Söderberg B (2001) Deterministic annealing and nonlinear assignment. arXiv preprint cond-mat/0105321
18. Kuhn HW (1955) The hungarian method for the assignment problem. Naval Res Log Quart 2(1-2):83–97
19. Lagerholm M, Peterson C, Söderberg B (1997) Airline crew scheduling with Potts neurons. Neural Comput 9(7):1589–1599
20. Lagerholm M, Peterson C, Söderberg B (2000) Airline crew scheduling using Potts mean field techniques. Eur J Oper Res 120(1):81–96
21. Lagoudakis MG, Markakis E, Kempe D, Keskinocak P, Kleywegt AJ, Koenig S, Tovey CA, Meyerson A, Jain S (2005) Auction-based multi-robot routing In: Robotics: Science and Systems, vol. 5, pp. 343–350
22. Luo L, Chakraborty N, Sycara K (2012) A distributed algorithm for constrained multi-robot task assignment for grouped tasks
23. Ma H, Koenig S (2016) Optimal target assignment and path finding for teams of agents. arXiv:1612.05693
24. Narasimha KSV, Kivelévitch E, Kumar M (2012) Ant colony optimization technique to solve the min-max multi depot vehicle routing problem. In: 2012 American control conference (ACC), pp 3980–3985. https://doi.org/10.1109/ACC.2012.6315583
25. Ng W, Mak K, Zhang Y (2007) Scheduling trucks in container terminals using a genetic algorithm. Eng Optim 39(1):33–47
26. Ohlsson M, Peterson C, Söderberg B (2001) An efficient mean field approach to the set covering problem. Eur J Oper Res 133(3):583–595
27. Parker LE (1995) L-alliance: a mechanism for adaptive action selection in heterogeneous multi-robot teams. Tech. rep., Oak Ridge National Lab., TN (United States)
28. Parker LE (1998) Alliance: an architecture for fault tolerant multi-robot cooperation. IEEE Trans Robot Automat 14(2):220–240
29. Peterson C, Söderberg B (1989) A new method for mapping optimization problems onto neural networks. Int J Neural Syst 1(01):3–22
30. Scott D, Manyam SG, Casbeer DW, Kumar M (2020) Market approach to length constrained min-max multiple depot multiple traveling salesman problem. In: 2020 American Control Conference (ACC), pp 138–143. https://doi.org/10.23919/ACC45564.2020.9147207
31. Sinkhorn R, Knopp P (1967) Concerning nonnegative matrices and doubly stochastic matrices. Pac J Math 21(2):343–348
32. Turner JS (1989) Approximation algorithms for the shortest common superstring problem. Inf Comput 83(1):1–20
33. Turpin M, Michael N, Kumar V (2015) An approximation algorithm for time optimal multi-robot routing. In: Algorithmic foundations of robotics XI, pp 627–640. Springer
34. Young D (2020) Sinkhorn-knopp algorithm for matrix normalisation. MATLAB Central File Exchange Retrieved October 23

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.