mathematics

MDPI

Article

Solution Methods for the Dynamic Generalized Quadratic Assignment Problem

Yugesh Dhungel and Alan McKendall *

Department of Industrial and Management Systems Engineering, West Virginia University, Morgantown, WV 26506, USA; yugeshdc@gmail.com
* Correspondence: alan.mckendall@mail.wvu.edu

Abstract

In this paper, the generalized quadratic assignment problem (GQAP) is extended to consider multiple time periods and is called the dynamic GQAP (DGQAP). This problem considers assigning a set of facilities to a set of locations for multiple periods in the planning horizon such that the sum of the transportation, assignment, and reassignment costs is minimized. The facilities may have different space requirements (i.e., unequal areas), and the capacities of the locations may vary during a multi-period planning horizon. Also, multiple facilities may be assigned to each location during each period without violating the capacities of the locations. This research was motivated by the problem of assigning multiple facilities (e.g., equipment) to locations during outages at electric power plants. This paper presents mathematical models, construction algorithms, and two simulated annealing (SA) heuristics for solving the DGQAP problem. The first SA heuristic (SAI) is a direct adaptation of SA to the DGQAP, and the second SA heuristic (SAII) is the same as SAI with a look-ahead/look-back search strategy. In computational experiments, the proposed heuristics are first compared to an exact method on a generated data set of smaller instances (data set 1). Then the proposed heuristics are compared on a generated data set of larger instances (data set 2). For data set 1, the proposed heuristics outperformed a commercial solver (CPLEX) in terms of solution quality and computational time. SAI obtained the best solutions for all the instances, while SAII obtained the best solution for all but one instance. However, for data set 2, SAII obtained the best solution for nineteen of the twenty-four instances, while SAI obtained five of the best solutions. The results highlight the effectiveness and efficiency of the proposed heuristics, particularly SAII, for solving the DGQAP.

Keywords: dynamic generalized quadratic assignment problem; generalized quadratic assignment problem; mathematical models; simulated annealing; heuristic

MSC: 90B06; 90B80; 90C27; 90C59; 49M99; 37N40

check for updates

Academic Editor: Cristian Lazureanu

Received: 12 November 2025
Revised: 4 December 2025
Accepted: 8 December 2025
Published: 17 December 2025

Citation: Dhungel, Y.; McKendall, A. Solution Methods for the Dynamic Generalized Quadratic Assignment Problem. *Mathematics* **2025**, *13*, 4021. https://doi.org/10.3390/math13244021

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Assignment problems are common combinatorial optimization problems (COPs) that carry much importance in different fields spanning logistics, manufacturing, business, telecommunication, healthcare, etc. There are mainly two types of assignment problems: one-to-one and many-to-one. For example, consider the quadratic assignment problem (QAP), a one-to-one assignment-type problem, which assigns equal-area facilities to locations while minimizing the sum of the costs of transporting materials between facilities and the costs of assigning (or installing) facilities to locations. In the QAP, each facility is

assigned to a location, and each location is assigned to a facility (i.e., one-to-one assignment). The QAP was introduced by Koopmans and Beckmann [1] and was proven to be NP Hard by Sahni and Gonzales [2]. See Burkard et al. [3] and Loiola et al. [4] for an extensive review of the solution techniques for the QAP.

There are many variants of the QAP. For some examples of different variants of the QAP see Silva et al. [5]. One of them is the generalized QAP (GQAP) which is a many-to-one assignment-type problem. That is, more than one facility may be assigned to each location without violating location capacities, but each facility is assigned to exactly one location. More specifically, the GQAP is stated as the problem of assigning a set of M facilities to a set of N locations ($M > N$) without exceeding the capacities of the locations, such that the sum of the assignment and transportation costs is minimized. The formulation of the GQAP is given below and is an adaptation of the model presented by Lee and Ma [6].

$$\text{Minimize } z = \sum_{i=1}^{M}\sum_{k=1}^{N} a_{ik}x_{ik} + \sum_{i=1}^{M}\sum_{\substack{j=1\\ j\neq i}}^{M}\sum_{k=1}^{N}\sum_{\substack{l=1\\ l\neq k}}^{N} c_{ijkl}f_{ij}d_{kl}x_{ik}x_{jl} \tag{1}$$

$$\text{Subject to} \sum_{k=1}^{N} x_{ik} = 1 \quad i = 1, \ldots, M \tag{2}$$

$$\sum_{i=1}^{M} r_i x_{ik} \leq C_k \quad k = 1, \ldots, N \tag{3}$$

$$x_{ik} = 0 \text{ or } 1 \quad \forall\, i,\ k \tag{4}$$

where M is the number of facilities, N is the number of locations, a_{ik} is the cost of assigning (installing) facility i to (at) location k, f_{ij} is the amount of materials flowing from facility i to facility j, d_{kl} is the distance from location k to location l, c_{ijkl} is the unit cost per distance unit of transporting materials from facility i (at location k) to facility j (at location l), r_i is the resource (or space) requirement of facility i, and C_k is the amount of resource (or space) capacity available at location k. The decision variable x_{ik} = 1 indicate that facility i is assigned to location k. When x_{ik} = 0, facility i is not assigned to location k. Objective function (1) minimizes the sum of the assignment (or installation) and transportation costs. Constraint (2) ensures that each facility is assigned to only one location. Constraint (3) ensures that the capacity of each location is not exceeded, and the restrictions on the decision variables are given in (4).

The term in objective function (1) used to obtain transportation cost has a quadratic term. As a result, the mathematical formulation (1)–(4) is a binary integer nonlinear programming model. The model is linearized by using a simple standard linear programming transformation (i.e., by substituting w_{ijkl} for $x_{ik}x_{jl}$). Then, replace objective function (1) with the following.

$$\text{Minimize } z = \sum_{i=1}^{M}\sum_{k=1}^{N} a_{ik}x_{ik} + \sum_{i=1}^{M}\sum_{\substack{j=1\\ j\neq i}}^{M}\sum_{k=1}^{N}\sum_{\substack{l=1\\ l\neq k}}^{N} c_{ijkl}f_{ij}d_{kl}w_{ijkl} \tag{5}$$

In addition, add the following constraints.

$$x_{ik} + x_{jl} - 1 \leq w_{ijkl} \quad i,\ j = 1, \ldots, M\ (j \neq i),\ k, l = 1, \ldots, N (l \neq k) \tag{6}$$

$$w_{ijkl} \geq 0 \quad \forall\, i,\ j \neq i,\ k, l \neq k \tag{7}$$

As a result, the linearized model, categorized as a mixed integer linear programming (MILP) model, for the GQAP consists of objective function (5) subject to constraints (2)–(4), (6) and (7). For other linearizations of the GQAP see Lee and Ma [6] and Cordeau et al. [7].

The GQAP literature is very limited compared to the QAP literature. Lee and Ma [6] presented the first formulation for the GQAP. Also, the authors presented three methods for the linearization of the formulation, and a branch-and-bound algorithm to optimally solve the GQAP. Hahn et al. [8] proposed a new algorithm based on a reformulation linearization technique (RLT) dual ascent procedure to optimally solve the GQAP. Similarly, Pessoa et al. [9] proposed two exact algorithms for the GQAP which combine a previously proposed branch-and-bound scheme with a new Lagrangean relaxation procedure over a known RLT formulation. Guignard [10] presented RLT models for the GQAP and special cases of the problem (i.e., crossdock door assignment problem and 0-1 quadratic knapsack problem). It is important to note that the exact algorithms presented above are unable to solve large-size problems within a reasonable timeframe. However, the following heuristics (or approximation algorithms) are able to obtain "good" solutions for large-size problems within a reasonable timeframe. Cordeau et al. [7] presented a linearization of the GQAP formulation as well as a memetic heuristic for the GQAP, which combines genetic algorithms (Holland [11]) and tabu search (Glover [12]). Mateus et al. [13] proposed several greedy randomized adaptive search procedures (GRASPs) with path-relinking heuristics for the GQAP using different construction, local search, and path-relinking procedures. McKendall and Li [14] presented three construction algorithms and a tabu search heuristic for the GQAP. Silva et al. [5] proposed a parallel memetic iterated tabu search heuristic for the QAP and four of its variants, which includes the GQAP as one of its variants. More recently, Greistorfer et al. [15] presented a tabu search matheuristic for the GQAP. Also, Fathollahi-Fard et al. [16] proposed a Benders reformulation based on RLT inequalities as well as a construction and a metaheuristic algorithm based on adaptive large neighborhood search (ALNS) for the GQAP. Within the ALNS heuristic, the authors incorporate removal–insertion heuristics, a local search algorithm which uses a tabu list, and a decision rule inspired by simulated annealing. In McKendall and Dhungel [17], the authors presented a simulated annealing heuristic for the GQAP. Sohrabi et al. [18] proposed a modified genetic algorithm (GA) called genetic engineering algorithm.

The GQAP has numerous applications related to facility design, scheduling, and network design. Lee and Ma [6] first introduced the GQAP to solve the problem of assigning a set of production equipment (or facilities) to manufacturing sites (or locations). Cordeau et al. [7] considered the application of GQAP on container yard management where the yard is partitioned into storage areas (or locations) and containers (or facilities) are assigned to these storage areas. McKendall and Li [14] considered the application of assigning a set of machines (or facilities) to a set of locations on the floor of a manufacturing plant. Also, Zou et al. [19] considered a GQAP with an additional capacity constraint to assign tasks to processors. Unal & Uysal [20] implemented GQAP to design the curriculum at a university. As stated above, the crossdock door assignment problem and the 0-1 quadratic knapsack problem are special cases of the GQAP (Guignard [10]). As stated in Greistorfer et al. [15], the GQAP has been used to model several applications such as order picking, storage layout in warehouse management, relational database design, and scheduling activities in semiconductor wafer processing.

In this paper, the dynamic GQAP (DGQAP) is considered. It is defined as the task of assigning a set of facilities to a set of locations for multiple periods in the planning horizon such that the sum of the transportation, assignment, and reassignment costs is minimized. The facilities may have different space requirements, and the capacities of the locations may vary during a multi-period planning horizon. Also, multiple facilities may be assigned to

each location during each period without violating the capacities of the locations. To the best of the authors' knowledge, this extension has not been studied before. Nevertheless, this research was motivated by the problem of assigning multiple facilities (equipment) to locations at electric power plants during planned outages. Since the equipment are of different sizes, the facilities may have different space requirements. More importantly, during planned outages some location capacities and distances between locations may change due to the performance of maintenance activities as well as the storage of the materials required for these activities. A similar application of the DGQAP is the management of construction facilities during construction projects, where construction facilities (e.g., cranes, equipment, materials) are assigned to locations at a construction site for a multiple period planning horizon (Elbeltagi et al. [21]). As the layout changes at the construction site, location capacities and distances between locations may change. In contrast, there are other applications where this may not be the case. Consider the applications presented above for the GQAP in Cordeau et al. [7] where the facilities (containers) are assigned to locations (storage areas) in the container yard during a multi-period planning horizon. This is an application of the DGQAP where the capacities of the locations and the distances between the locations may not change. Nevertheless, the DGQAP can be considered in the other applications presented above for the GQAP where there are multiple periods in the planning horizon such as in Zou et al. [19] (i.e., assigning tasks to processors), Unal & Uysal [20] (i.e., designing the curriculum at a university), and McKendall and Li [14] (i.e., assigning machines to locations in a manufacturing plant). In general, when there are changes in the problem (input) data during the planning horizon—more specifically, if one or more of the following input data changes during the planning horizon—it would be better to solve the resulting problem as a DGQAP as opposed to solving multiple GQAPs, as will be discussed later.

- There are changes in either the amount of materials transported between facilities or the costs of transporting materials between facilities.
- There are changes in the location and location capacities, which may change the distances between the locations.
- There are changes in the facilities' requirements.

As stated previously, to the best of our knowledge, the GQAP has not been extended to the dynamic GQAP. However, there are many dynamic (or multi-period) COPs in the literature, besides the one mentioned earlier (i.e., assigning facilities to locations at a construction site as in Elbeltagi et al. [21]). In 1986, Rosenblatt [22] extended the QAP to consider multiple discrete time periods and presented a dynamic programming algorithm to solve the problem referred to as the dynamic facility layout problem (DFLP) (i.e., the problem of assigning facilities to locations during a multi-period planning horizon such that sum of the transportation and rearrangement costs is minimized). This problem differs from the DGQAP, since each location is assigned to only one facility during each period as opposed to assigning multiple facilities to a location without exceeding its capacity, as in the DGQAP. For examples of SA algorithms used to solve the DFLP, see McKendall et al. [23] and Khajemahalle et al. [24]. In 1997, Kogan et al. [25] extended the GAP to the dynamic GAP (DGAP), which considers continuous-time. In 2012, Mazzola and Neebe [26] considered an extension of the GAP to multiple discrete time periods. This problem is called MultiGAP, which considers assigning multiple tasks to agents, without exceeding the capacities of the agents, for each period in the planning horizon such that the sum of the assignment and reassignment costs is minimized. The authors presented two mathematical models, a branch-and-bound algorithm, and a Lagrangian heuristic for the problem. It is important to note that MultiGAP is a special case of the DGQAP where the facilities are the tasks and the locations are the agents. In other words, the

main difference between MultiGAP and DGQAP is that the latter problem also considers the sum of the transportation costs in the objective function, which adds a quadratic term to the formulation, making the mathematical model nonlinear and more difficult to solve. Moccia et al. [27] present a different DGAP which extends the GAP by considering discrete periods and by associating a start and finish time to each task. Also, they consider additional constraints related to warehouse and yard management applications. In 2005, McKendall et al. [28] presented a mathematical model and SA algorithms for assigning activities to workspaces and resources to either work or storage locations during a multi-period planning horizon while minimizing total distance resources travel during planned outages at electric power plants. The problem is defined as the dynamic space allocation problem and is a generalization of the DGQAP. Notice, there are two types of locations: workspaces and storage spaces where activities (groups of resources required to perform activities) are assigned to workspaces and individual resources are assigned to storage locations. Although there are many other dynamic COPs such as the dynamic vehicle routing problem (Wang et al., [29]) and dynamic facility location problem (Zhang and Li [30]), here are just a few. This shows the importance and relevance for considering the dynamic extensions of several of the well-known COPs.

As mentioned above, SA is used to solve dynamic COPs such as DFLP (e.g., McKendall et al. [23] and Khajemahalle et al. [24]) and the dynamic space allocation problem (McKendall et al. [28]). However, Kirkpatrick et al. [31] was the first to use SA to solve a COP called the Ising spin glass problem. More recently, Huo et al. [32] presented an SA heuristic for a task assignment and path planning problem which is viewed as a three-dimensional vehicle routing problem. This problem is a generalization of the generalized assignment problem combined with multiple traveling salesperson problems. Dai et al. [33] developed an SA heuristic for a home healthcare location-routing problem with a mixed fleet of electric and conventional vehicles that consider battery swapping stations. Also, Leelertkij et al. [34] presented an SA algorithm for a multi-objective vehicle routing problem with time windows. Tole et al. [35] used an SA heuristic to solve a new variant of the bin packing problem called the circular bin packing problem with rectangular items. Lin et al. [36] presented an SA algorithm for the set orienteering problem with mandatory visits. This shows the diversity of problems that are solved using SA. Recently, researchers have used multi-neighborhood SA algorithms to solve COPs. Rosati and Schaerf [37] used a multi-neighborhood SA algorithm to solve the capacitated dispersion problem. The authors stated that their algorithm improved the state-of-the-art methods on almost all instances from public benchmarks. The multi-neighborhood SA algorithm developed for a capacitated facility location problem with incompatibilities presented in Ceschia and Schaerf [38] was able to outperform all previous methods on the publicly available data set. Da Ros et al. [39] proposed a multi-neighborhood SA algorithm for an oven scheduling problem. The authors stated that their algorithm was able to find new best solutions for many instances. Van Bulck et al. [40] presented a multi-neighborhood SA algorithm for a capacitated examination time-tabling problem. The authors stated that the proposed algorithm was able to produce new best solutions for a variety of well-known instances. As stated previously, McKendall and Dhungel [17] presented an SA heuristic for the GQAP, which considers multiple neighborhoods. Compared to the state-of-the-art methods, the proposed SA algorithm was able to produce the best-known solutions using very little computational time on a publicly available data set. Since multi-neighborhood SA was successfully applied to several related COPs (i.e., obtain very good solutions for many complex problems) and is easy to implement, it is used to solve the proposed DGQAP.

In this paper, mathematical models, construction algorithms, and simulated annealing (SA) heuristics are presented for the DGQAP problem. In Section 2, the mathematical

models are presented for the DGQAP, and an illustrative example is solved using one of the proposed models. The construction algorithms and the SA heuristics for the DGQAP are presented in Section 3. In Section 4, some computational results of the proposed techniques on a set of randomly generated test problems are given. Finally, Section 5 provides conclusions.

2. Mathematical Models for the DGQAP

2.1. Mathematical Programming Models

The DGQAP is the problem of assigning a set of M facilities to a set of N locations ($M > N$) for T periods in the planning horizon such that the sum of the transportation, assignment, and reassignment costs is minimized. The assumptions of the DGQAP are as follows.

(1) The planning horizon is divided into multiple periods. Periods may be defined as weeks, months, quarters, or years.
(2) The input data is deterministic and known. That is, for each period t, the number of units of materials transported between facilities i and j (f_{tij}), the distances between locations k and l (d_{tkl}), the space requirement of each facility i (r_{ti}), the capacity of each location k (C_{tk}), the costs of assigning and reassigning each facility i to each location k (a_{tik}), and the unit cost per distance unit of transporting materials between each pair of facility i at location k and facility j at location l (c_{tijkl}) are deterministic and known.
(3) One or more facilities may be assigned to each location during each period such that the capacity of the location is not exceeded.
(4) The objective is to minimize the sum of the transportation, assignment, and reassignment costs. The assignment costs are the costs of initially assigning (or installing) facilities to locations during the first period. Reassignment costs are the costs of reassigning facilities to different locations (uninstalling and reinstalling facilities) after the first period. More specifically, reassignment cost of a facility i assigned to location l in period t (a_{til}) consists of the costs of uninstalling facility i assigned to location k at end of period $t-1$ plus the cost of reinstalling facility i at a new location l at the beginning of the next period t.

Next, a mathematical formulation is presented for the DGQAP, which is an extension of the GQAP model, but first the decision variables are defined as follows. The decision variable $x_{tik} = 1$ indicate that facility i is assigned to location k at period t. When $x_{tik} = 0$, facility i is not assigned to location k at period t.

$$\text{Minimize } z = \sum_{i=1}^{M}\sum_{k=1}^{N} a_{1ik}x_{1ik} + \sum_{i=1}^{M}\sum_{k=1}^{N}\sum_{\substack{l=1\\ l\neq k}}^{N}\sum_{t=2}^{T} a_{til}x_{(t-1)ik}x_{til} + \sum_{i=1}^{M}\sum_{\substack{j=1\\ j\neq i}}^{M}\sum_{k=1}^{N}\sum_{\substack{l=1\\ l\neq k}}^{N}\sum_{t=1}^{T} c_{tijkl}f_{tij}d_{tkl}x_{tik}x_{tjl} \tag{8}$$

$$\sum_{k=1}^{N} x_{tik} = 1 \quad i = 1, \ldots, M,\ t = 1, \ldots, T \tag{9}$$

$$\sum_{i=1}^{M} r_{ti}x_{tik} \leq C_{tk} \quad k = 1, \ldots, N,\ t = 1, \ldots, T \tag{10}$$

$$x_{tik} = 0 \text{ or } 1 \quad \forall\, i,\ k,\ t \tag{11}$$

Objective function (8) minimizes the sum of the assignment, reassignment, and transportation costs. Constraint (9) ensures that each facility is assigned to only one location in each period. Constraint (10) ensures that the capacity of each location is not exceeded in each period. The restrictions on the decision variables are given in (11).

The mathematical formulation presented above for the DGQAP is nonlinear due to the nonlinear terms in Objective function (8) and is categorized as a BINLP model. In order to obtain the global optimal for the DGQAP, the nonlinear model above is linearized by using the simple standard transformation by substituting y_{tikl} for the term $x_{(t-1)ik}x_{til}$ and w_{tijkl} for the term $x_{tik}x_{tjl}$. This yields the following linearized objective function.

$$\text{Minimize } z = \sum_{i=1}^{M}\sum_{k=1}^{N} a_{1ik}x_{1ik} + \sum_{i=1}^{M}\sum_{k=1}^{N}\sum_{\substack{l=1\\l\neq k}}^{N}\sum_{t=2}^{T} a_{til}y_{tikl} + \sum_{i=1}^{M}\sum_{\substack{j=1\\j\neq i}}^{M}\sum_{k=1}^{N}\sum_{\substack{l=1\\l\neq k}}^{N}\sum_{t=1}^{T} c_{tijkl}f_{tij}d_{tkl}w_{tijkl} \tag{12}$$

and the following constraints.

$$x_{(t-1)ik} + x_{til} - 1 \leq y_{tikl} \quad i = 1,\ldots,M,\ k,l = 1,\ldots,N(l \neq k),\ t = 1,\ldots,T \tag{13}$$

$$x_{tik} + x_{tjl} - 1 \leq w_{tijkl} \quad i,\ j = 1,\ldots,M\ (j \neq i),\ k,l = 1,\ldots,N(l \neq k),\ t = 1,\ldots,T \tag{14}$$

$$y_{tikl} \geq 0 \quad \forall\ i,\ k,l \neq k,t \tag{15}$$

$$w_{tijkl} \geq 0 \quad \forall\ i,\ j \neq i,\ k,l \neq k,t \tag{16}$$

As a result, the linearized mathematical model for the DGQAP, which is categorized as a MILP model, consists of Objective function (12) subject to Constraints (9)–(11) and (13)–(16). Next, a DGQAP instance is presented and solved optimally using the MILP model presented above and the commercial solver CPLEX.

Consider a DGQAP instance in which six facilities are assigned to three locations in a 2-period planning horizon (i.e., M = 6, N = 3, T = 2). The unit cost per distance unit of transporting materials between each pair of the facilities during each period is 2 (i.e., c_{tijkl} = 2). Note, all costs are in USD 1000s. See Figure 1 below for the input data for a small DGQAP instance. For example, the cost of assigning facility 2 to location 3 in period 1 is 12 (i.e., a_{123} = USD 12,000), and the cost of reassigning facility 2 (located at either location 1 or 2 in period 1) to location 3 in period 2 is also 12 (i.e., a_{223} = USD 12,000). Furthermore, the distance from location 3 to location 2 for both periods 1 and 2 are same, which is 3 distance units (i.e., $d_{132} = d_{232} = 3$). However, the space requirements of a facility during different periods may change due to either an increase or decrease in production capacity, for example. In the data below, the space requirement for facility 1 increases for period 2 from period 1 (i.e., r_{11} = 2, r_{21} = 3), but for facility 3 for each period, the facility requirement is the same (i.e., $r_{13} = r_{23} = 2$). Furthermore, there is a possibility that the space capacity of a location may change during different planning periods, as discussed above for the application of assigning facilities to locations during planned outages at electric power plants. For example, additional space may become available at a location when a maintenance activity is completed. Notice the space capacity of location 1 during period 1 is 3 units (i.e., C_{11} = 3), but in period 2 it increased to 5 units (i.e., C_{21} = 5).

Using the data given in Figure 1, the MILP model presented for the DGQAP (i.e., objective function (12) subject to constraints (9)–(11) and (13)–(16)) and the commercial solver CPLEX (version 12.4), the following optimal solution is obtained. The nonzero values of the decision variable x_{tik} are as follows.

$$x_{113} = x_{123} = x_{133} = x_{141} = x_{151} = x_{162} = x_{213} = x_{221} = x_{233} = x_{241} = x_{251} = x_{262} = 1$$

a_{tik} =

	1	2	3
1	10	18	14
2	16	10	12
3	12	18	20
4	20	12	14
5	14	20	10
6	16	12	20
1	10	18	14
2	16	10	12
3	12	18	20
4	20	12	14
5	14	20	10
6	16	12	20

f_{tij} =

	1	2	3	4	5	6
1	0	5	10	2	0	0
2	0	0	10	0	0	0
3	0	0	0	5	0	0
4	0	0	0	0	10	0
5	0	0	0	0	0	5
6	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	10	10	0
3	0	0	0	0	5	0
4	0	0	0	0	0	10
5	0	0	0	0	0	5
6	0	0	0	0	0	0

d_{tkl} =

	1	2	3
1	0	2	5
2	2	0	3
3	5	3	0
1	0	2	5
2	2	0	3
3	5	3	0

r_{ti} =

	1	2	3	4	5	6
1	2	1	2	1	2	3
2	3	2	2	1	2	3

C_{tk} =

	1	2	3
1	3	3	6
2	5	3	6

Figure 1. Input data for a small DGQAP instance.

In other words, in period 1 assign facilities 4 and 5 to location 1, facility 6 to location 2, and facilities 1, 2, and 3 to location 3. In period 2, assign facilities 2, 4, and 5 to location 1, facility 6 to location 2, and facilities 1 and 3 to location 3. See Figure 2 for a display of the solution. Notice at the beginning of period 2, facility 2 (assigned to location 3 in period 1) is reassigned to location 1. Total assignment cost (AC) and transportation cost (TC) for period 1 is USD 92,000 and USD 90,000, respectively, and total reassignment cost (RC) and transportation cost (TC) for period 2 is USD 16,000 and USD 110,000. Thus, the total cost of the optimal solution is USD 308,000.

AC = USD 92,000, TC = USD 90,000

Loc 1	Loc 2	Loc 3
4, 5	6	1, 2, 3

Period 1

RC = USD 16,000, TC = USD 110,000

Loc 1	Loc 2	Loc 3
2, 4, 5	6	1, 3

Period 2

Figure 2. Optimal solution for DGQAP model for small DGQAP instance with total cost of USD 308,000.

Since this is the first paper known to the authors, which consider the DGQAP, the DGQAP may be solved (or is oftentimes solved) sequentially as multiple GQAPs. That is, for period 1 the GQAP may be solved, then the GQAP may be solved for period 2, and so on. As stated earlier, it would be better to solve the resulting problem as a DGQAP as opposed to solving multiple GQAPs, since finding the optimal (or good) solution for each subproblem (or period) oftentimes does not yield the optimal (or good) solution for the problem as a whole. Consider the data given in Figure 1, the MILP model presented for the GQAP, objective function (5) subject to constraints (2)–(4), (6), (7) and the commercial solver CPLEX. The GQAP is optimally solved for periods 1 and 2 separately, and the following optimal solution is obtained for both problems where the nonzero values of the decision variables x_{ik} are as follows.

$$x_{13} = x_{23} = x_{33} = x_{42} = x_{52} = x_{61} = 1 \text{ for } T = 1 \text{ and } x_{11} = x_{23} = x_{31} = x_{43} = x_{52} = x_{63} = 1 \text{ for } T = 2$$

In other words, assign facility 6 to location 1, facilities 4 and 5 to location 2, and facilities 1, 2, and 3 to location 3 in period 1. In period 2, assign facilities 1 and 3 to location 1, facility 5 to location 2, and facilities 2, 4, and 6 to location 3. For a graphical display of the solution, see Figure 3 below. Notice facilities 1, 3, 4, and 6 are reassigned to different locations in period 2. Total AC and TC for period 1 is USD 94,000 and USD 62,000, respectively, and total RC and TC for period 2 is USD 56,000 and USD 110,000, respectively. Thus, the total cost of the solution is USD 322,000, which gives a higher total cost solution (i.e., USD 14,000 higher) than solving the problem as a DGQAP.

AC = USD 94,000, TC = USD 62,000

Loc 1	Loc 2	Loc 3
6	4, 5	1, 2, 3

Period 1

RC = USD 56,000, TC = USD 110,000

Loc 1	Loc 2	Loc 3
1, 3	5	2, 4, 6

Period 2

Figure 3. Solution for DGQAP instance where each period is solved as an individual GQAP model with total cost of USD 322,000.

In contrast, if the DGQAP instance is solved for period 1 using the GQAP model, the above solution is obtained for period 1 (i.e., $x_{13} = x_{23} = x_{33} = x_{42} = x_{52} = x_{61} = 1$). If this solution for period 1 is put into the DGQAP model, then the solution for period 2 is $x_{13} = x_{22} = x_{33} = x_{42} = x_{51} = x_{61} = 1$. In other words, rearrangement costs are considered, unlike the previous scenario. Nevertheless, see Figure 4 below for a graphical display of the solution. Notice facilities 2 and 5 are reassigned to different locations in period 2, and the total cost of the solution is USD 310,000. Therefore, the savings for solving the complete problem as a DGQAP is USD 2000. Although this may not seem like a lot, it adds up as the number of facilities and planning periods increases.

AC = USD 94,000, TC = USD 62,000

Loc 1	Loc 2	Loc 3
6	4, 5	1, 2, 3

Period 1

RC = USD 24,000, TC = USD 130,000

Loc 1	Loc 2	Loc 3
5, 6	2, 4	1, 3

Period 2

Figure 4. Solution for DGQAP instance where period 1 is solved as a GQAP model, and assignment for period 1 is put into the DGQAP model to obtain the assignment for period 2. Here the total cost of the assignments is USD 310,000.

It is important to note that the DGQAP instance has given above in Figure 1 is a tightly constrained problem, which means that it is more difficult to find feasible solutions for this type of problem compared to loosely constrained problems. Consider the total amount of space required for the facilities in periods 1 and 2 are 11 (i.e., $r_{11} + r_{12} + \ldots + r_{16} = 2 + 1 + 2 + 1 + 2 + 3 = 11$) and 13, respectively, and the total capacity of the locations for periods 1 and 2 are 12 (i.e., $C_{11} + C_{12} + C_{13} = 3 + 3 + 6 = 12$) and 14, respectively. Thus, the amount of space utilized in periods 1 and 2 are 91.7% and 92.9%, respectively, which averages to 92.3%. Therefore, this is a tightly constrained problem and obtaining feasible solutions are oftentimes difficult, which will be discussed in the next section. However, since this DGQAP instance has 2 periods ($T = 2$), 6 facilities ($M = 6$), 3 locations ($N = 3$), and average space utilization is 92.3%, the instance code for this problem is 2-6-3-92. Note in Cordeau et al. [7] a similar notation is used to identify the complexity (i.e., size of an instance and whether the instance is loosely or tightly constrained) of GQAP instances. This notation will be used later when describing the set of test problems generated to test the performances of the proposed solution techniques. Nevertheless, since only small-size problems can be solved within a reasonable timeframe using the MILP model presented for the proposed problem, which will be shown below in Section 4.3, another formulation is given for the proposed problem.

2.2. Combinatorial Optimization Problem (COP) Model

To overcome the limitation of the mathematical model (i.e., unable to solve large-size problems in reasonable time), heuristics are developed for the DGQAP. As a result, additional notation is used to give another formulation of the DGQAP. Using a combinatorial optimization problem (COP) formulation, the solution for the DGQAP is represented as $\boldsymbol{S} = \{\boldsymbol{S_1}, \boldsymbol{S_2}, \ldots, \boldsymbol{S_T}\}$ where $\boldsymbol{S_t}$ is used to represent the assignment of facilities to locations in period t where $t = 1, \ldots, T$, $\boldsymbol{S_t} = (s_t(1), s_t(2), \ldots, s_t(M))$ and $s_t(i) = k$ indicates that facility i is assigned to location k in period t. Therefore, the solution $\boldsymbol{S}$ is represented as follows.

$$\boldsymbol{S} = \{(s_1(1), s_1(2), \ldots, s_1(M)), (s_2(1), s_2(2), \ldots, s_2(M)), \ldots, (s_T(1), s_T(2), \ldots, s_T(M))\} \quad (17)$$

Considering the small DGQAP instance 2-6-3-92 given earlier in Figure 1, the optimal solution given in Figure 2 is represented as $S = \{S_1, S_2\} = \{(3, 3, 3, 1, 1, 2), (3, 1, 3, 1, 1, 2)\}$. For example, $s_1(2) = 3$ and $s_2(2) = 1$, which means in period 1 facility 2 is assigned to location 3 and is reassigned to location 1 in period 2. Using the notation defined earlier for the mathematical models, the index representing location k (l) of facilities i (j) in period t is replaced with $s_t(i)$ ($s_t(j)$), and the COP model for the DGQAP is given as follows.

$$TC(S) = \sum_{i=1}^{M} a_{1is_1(i)} + \sum_{t=2}^{T}\sum_{i=1}^{M} a_{tis_t(i)} ra_{ti} + \sum_{t=1}^{T}\sum_{i=1}^{M}\sum_{\substack{j=1 \\ j\neq i}}^{M} c_{tijs_t(i)s_t(j)} f_{tij} d_{ts_t(i)s_t(j)} \quad (18)$$

$$\sum_{\forall i \text{ s.t. } s_t(i)=k} r_{ti} \leq C_{tk} \quad k = 1, \ldots, N, \; t = 1, \ldots, T \quad (19)$$

Notice an additional variable ra_{ti} is introduced into the second term of (18) to determine the reassignment cost of each facility i in period $t > 1$. That is, if facility i assigned to location k in period $t - 1$ (i.e., $s_{t-1}(i) = k$) and is reassigned to location l ($l \neq k$) in period t (i.e., $s_t(i) = l$), then reassignment cost is incurred, and the variable $ra_{ti} = 1$. Otherwise, $ra_{ti} = 0$, if facility i is not reassigned to a different location l in period t (i.e., $s_{t-1}(i) = s_t(i) = k$). Regardless, given the solution (17), Equation (18) gives the sum of the assignment, reassignment, and transportation costs (i.e., total cost of solution $\boldsymbol{S}$). Constraint (19) ensures that the capacity of each location is not exceeded during each period. Next, construction algorithms are presented for the DGQAP.

3. Heuristic Algorithms for the DGQAP

3.1. Construction Algorithms

Two construction algorithms are presented for the proposed DGQAP. The sole purpose of the construction algorithms is to generate solutions for the improvement algorithms (i.e., the proposed SA heuristics) presented later. The first construction algorithm (CAI) begins by assigning facilities to locations for the first time period ($t = 1$) and then moves to the next time period until all time periods have been considered. Facilities requiring more space are assigned to locations first (i.e., facilities are sorted and assigned based on space requirements). However, if there is not enough capacity at a location for a facility, the next facility in the sorted list is considered. The purpose of this algorithm is to ensure that feasible solutions are obtained for the tightly constrained problems without regards to transportation and assignment/reassignment costs. On the other hand, the second construction algorithm (CAII), attempts to assign facilities to the same location in multiple periods in order to construct solutions with lower reassignment costs. This algorithm performs best on loosely constrained problems.

3.1.1. Construction Algorithm I (CAI)

Algorithm 1 shows the steps of the pseudo-code for the first construction algorithm (CAI) used to generate a solution for the DGQAP. It is similar to the first of three construction algorithms presented in McKendall and Li [14] for the static GQAP. However, it has been extended to include a multi-period planning horizon. This algorithm works well for obtaining feasible solutions for tightly constrained DGQAPs by attempting to assign as many facilities (priority is given to facilities which require the most space) as possible to the first location, then the second location, and so on such that location capacities are not exceeded. This method ensures that feasible solutions are obtained regardless of the total costs of the solutions, since feasible solutions may be very difficult to obtain for tightly constrained problems.

To illustrate the construction algorithm CAI in Algorithm 1 for the small DGQAP instance 2-6-3-92 given in Figure 1, for period 1 ($t = 1$) the input arguments (capacities of the locations for period 1, C_1, and the facility requirements for period 1, r_1) are initialized (step 2), where $C_1 = \{3, 3, 6\}$ and $r_1 = \{2, 1, 2, 1, 2, 3\}$. Next, the set $EFS_1 = \{6, 1, 3, 5, 2, 4\}$ is generated (step 3). Notice facility 6 has the largest facility requirement (i.e., $r_1(6) = 3$) and is listed first in EFS_1. However, since $r_1(1) = r_1(3) = r_1(5) = 2$ and $1 < 3 < 5$, facilities 1, 3, and 5 are placed in set EFS_1 in that order (break ties by putting lower facility number first). Once set EFS_1 is obtained, facilities are assigned to locations based on their order in this set (steps 4–18). More specifically, first the assignment of the first location ($k = 1$) to facility i in the first position ($p = 1$) in the vector EFS_1 is considered, where $i = EFS_1(1) = 6$ (steps 4–8). Since facility 6 does not exceed the capacity of location 1 (i.e., $r_1(6) = 3 \leq C_1(1) = 3$ holds), assign facility 6 to location 1 (i.e., set $s_1(6) = 1 = k$) and update the capacity of location 1 (i.e., $C_1(1) = C_1(1) - r_1(6) = 0$) as well as remove facility 6 and update EFS_1 which is now {1, 3, 5, 2, 4} (steps 9–12). Since the remaining capacity is zero for location 1 (i.e., $C_1(1) = 0$) which is less than the facility requirement of the last facility in EFS_1 (i.e., $C_1(1) = 0 < r_1(4) = 1$), exit loop (in steps 7–16) and set $k = 2$ (step 17) and continue second iteration (steps 5–18). Next, assign facility i (i.e., $i = EFS_1(1) = 1$) to location 2 and update the capacity of location 2 (i.e., set $s_1(1) = 2 = k$ and $C_1(2) = C_1(2) - r_1(1) = 3 - 2 = 1$), since facility 1 does not exceed the capacity of location 2 (steps 6–11). Also, remove facility 1 and update EFS_1 which is now {3, 5, 2, 4} (step 12). Since $C_1(2) = 1 = r_1(4)$, there is still enough capacity in location 2 to assign facility 4 (step 16). However, facility 4 cannot be assigned to location 2, since higher priority is given to other facilities in the set EFS_1. So next facility 3 is considered. In the third iteration, facility 3 cannot be assigned to location 2 since there is not enough capacity, i.e., $r_1(3) = 2 > 1 = C_1(2)$ (steps 9). Therefore, update p such that $p = p + 1 = 2$ (step 14) and continue the fourth iteration (steps 7–16). Since there is not enough capacity, in iteration 5 ($p = 3$), facility $i = 2$ is assigned to location 2 (i.e., set $s_1(2) = 2 = k$ and $C_1(2) = C_1(2) - r_1(2) = 1 - 1 = 0$). Furthermore, update EFS_1 which is now {3, 5, 4} (step 12). Since there is no capacity remaining at location 2 (step 16), set $k = 3$ (step 17) and continue iteration 6 (steps 6–18). In iteration 6, 7, and 8, facilities 3, 5, and 4, respectively, are assigned to location 3. As a result, EFS_1 is empty (i.e., $EFS_1 = \{\}$) and the solution for period 1 is obtained (i.e., $\boldsymbol{S_1} = (2, 2, 3, 3, 3, 1)$). In other words, all the facilities are assigned to locations in period 1. Therefore, set $t = t + 1 = 2$ and continue for loop (steps 1–19) to obtain solution $\boldsymbol{S}$ (step 20).

Algorithm 1: CAI(C_{tk}, r_{ti})

1: **for** $t \leftarrow 1$ **to** T **do**
2: Initialize C_t and r_t
3: Construct EFS_t by sorting i in descending order based on $r_t(i)$ (break ties by lower i)
4: $k \leftarrow 1$ % k is the location index in array C_t

```
5:    while k ≤ N and EFS_t ≠ {} do % {} is the empty set
6:        p ← 1                    % p is the position index in array EFS_t
7:        repeat
8:            i ← EFS_t(p)         % i is the facility number
9:            if r_t(i) ≤ C_t(k) then
10:               s_t(i) ← k
11:               C_t(k) ← C_t(k) − r_t(i)
12:               Remove(EFS_t(p))   % Remove element EFS_t(p) from set EFS_t
13:           else
14:               p ← p + 1
15:           end if
16:       until C_t(k) < r_t(EFS_t(Last)) do   % EFS_t(Last) is the last facility in array EFS_t
17:       k ← k + 1
18:   end while
19: end for
20: return S
```

The construction algorithm CAI generates the solution $\boldsymbol{S}$ = {(2, 2, 3, 3, 3, 1), (1, 1, 3, 3, 3, 2)} which gives a total cost of 598 (i.e., $TC(\boldsymbol{S}) = 598$). As stated earlier, the objective of CAI is to ensure that feasible solutions are generated especially for tightly constrained problems regardless of the total cost of the solutions. Next, the second construction algorithm (CAII) is presented for the DGQAP which attempts to produce feasible solutions with lower reassignment costs.

3.1.2. Construction Algorithm II (CAII)

Algorithm 2 shows the steps of the pseudo-code for the second construction algorithm (CAII) used to construct a solution for the DGQAP. After initializing the parameters (step 1), first assign the facilities to locations in time period 1 using CAI (step 2). Next, construct the set EFS_t for each time period t for $2 \leq t \leq T$ (step 3). Then facilities in the current period t ($t > 1$) are assigned to the same locations as in the previous period $t - 1$ (e.g., $s_t(i) = s_{t-1}(i)$), based on their order in the set ESF_t, if there is enough capacity at the locations in period t (e.g., $r_t(i) \leq C_t(s_{t-1}(i)) = C_t(k)$ if $s_{t-1}(i) = k$) (steps 4–15). More specifically, if there is enough capacity at period t to assign the facility in the first position ($p = 1$) in the set ESF_t to the same location it is assigned to in period $t - 1$, assign facility to location, update the location capacity and remove facility from set ESF_t (steps 7–11). Now, if the facility in the first position in the set ESF_t cannot be assigned to the location it was assigned to in the previous period, move to the facility in the next position ($p = p + 1$) in the set ESF_t (step 13) and continue steps 7–14 until all facilities in set ESF_t has been considered. The facilities remaining in the set ESF_t at step 15 (i.e., the facilities that cannot be assigned to their same locations as in the previous period) are assigned to the first available locations with enough capacity, starting with location 1 then 2 and so on (step 17). For instance, if facility i in period 1 was assigned to location 3 ($s_1(i) = k = 3$), but there is not enough capacity in period 2 to assign facility i to location 3 ($r_2(i) > C_2(s_1(i)) = C_2(3)$), then facility i is assigned to the location k with enough capacity in period 2, starting with location 1. In our problem instance location $k = 1$ would be considered first, then location $k = 2$. Note that if it is not possible to assign all the facilities to locations in a period, use the CAI to assign all the facilities to locations for that period (steps 18–20). Repeat this process (continue for loop) for all time periods (steps 4–22) and obtain solution S (step 23). Again, CAII attempts to assign facilities to the same location in multiple periods to construct solutions with lower reassignment costs. This algorithm performs best on loosely constrained problems

and on problems with relatively high reassignment cost compared to transportation cost.

Algorithm 2: CAII(C_{tk}, r_{ti})

```
1: Initialize C_t and r_t for 1 ≤ t ≤ T
2: S_1 ← CAI(C_1, r_1)
3: Construct EFS_2, ..., EFS_T as discussed in CAI
4: for t ← 2 to T do
5:     p ← 1                % p is the position index in array EFS_t
6:     while p ≤ Last do    % Last is the position of the last facility in array EFS_t
7:         i ← EFS_t(p)     % i is the facility number
8:         if r_t(i) ≤ C_t(s_{t-1}(i)) then
9:             s_t(i) ← s_{t-1}(i)
10:            C_t(s_t(i)) ← C_t(s_t(i)) − r_t(i)
11:            Remove(EFS_t(p))   % Remove element EFS_t(p) from set EFS_t
12:        else
13:            p ← p + 1
14:        end if
15:    end while
16:    if EFS_t ≠ {} then
17:        Assign facilities in EFS_t to location with enough capacity
18:        if insufficient capacity then
19:            S_t ← CAI(C_t, r_t)
20:        end if
21:    end if
22: end for
23: return S
```

CAI will be used to construct starting solutions for the proposed SA heuristic (called SAI) which is a direct adaptation of SA to the DGQAP. However, the CAII will be used to produce starting solutions for the proposed SA heuristic (called SAII) which uses SAI with a look-ahead/look-back search strategy. This strategy works well for DGQAP instances where reassignment cost is relatively high. Therefore, starting with a solution with low rearrangement cost as with the CAII algorithm would aid in converging to a near optimal solution more quickly. On the other hand, the SAI heuristic may work better for DGQAP instances where reassignment cost is relatively low. As a result, the starting solutions produced by the CAI is sufficient. In summary, once a solution is generated using one of the proposed construction algorithms, either CAI or CAII, improvement algorithms which are discussed below are used to obtain "good" local optimums. The proposed improvement algorithms consist of an SA heuristic and a steepest descent heuristic. Next, the SA heuristics are presented for the DGQAP.

3.2. Simulated Annealing (SA) Heuristic

Simulated annealing (SA) is a metaheuristic used to solve many different types of COPs. As mentioned earlier, McKendall et al. [28] presented two SA heuristics to solve the dynamic space allocation problem, and McKendall et al. [23] presented two SA heuristics to solve a dynamic QAP (i.e., DFLP). The SA heuristics presented in the latter paper were modified and extended to solve the proposed problem. The first SA heuristic (SAI) is a direct adaptation of SA to the DGQAP, and the second SA heuristic (SAII) is the same as SAI with a look-ahead/look-back search strategy. Algorithm 3 shows the pseudo-code for

the SAI and SAII heuristics. Next, the two SA heuristics are presented and discussed for the proposed problem.

3.2.1. Simulated Annealing I (SAI)

As stated previously, the proposed SAI heuristic is a direct adaptation of SA to the DGQAP. Algorithm 3 shows the steps of the pseudo-code for the SAI heuristic. It starts with an initial solution S_0 (also called the current solution) obtained by the proposed construction algorithm CAI (step 2), and Equation (18) is used to obtain the total cost of the solution (step 3). This solution and its cost are defined as the best solution and best cost (step 4). After initializing the current temperature and the iteration counter n, which keeps track of the number of iterations performed at each temperature (step 5), a neighboring solution S is generated randomly using either a drop/add or pairwise exchange operation on the solution S_0 (step 7). The details of these operations are given below. Next, constraints (19) of the COP model is used to check the feasibility of the neighboring solution S (step 8). If the neighboring solution S is infeasible, the solution S_0 remains as the current solution (return to step 7). Otherwise, the cost of the neighboring solution, $TC(S)$, is obtained, and it is compared with the cost of the current solution, $TC(S_0)$, by calculating ΔTC, where $\Delta TC = TC(S_0) - TC(S)$ (steps 9–10). If the cost of neighboring solution S is better than the cost of the initial solution S_0 (i.e., $\Delta TC > 0$), the neighboring solution S becomes the current solution, i.e., set $S_0 = S$ (step 11). In addition, if the cost of the neighboring solution S is worse than the cost of the current solution S_0 (i.e., $\Delta TC < 0$), then the neighboring solution S is selected as the current solution S_0 (i.e., set $S_0 = S$) with respect to an acceptance probability p (steps 10–11). In either of the last two cases mentioned above, the current solution S_0 is updated, and if this solution gives the best cost, the best solution and its cost are updated (steps 12). Since we are describing the SAI heuristic, the look-ahead/look-back strategy is not considered (bypass step 13). Next, the iteration counter at the current temperature, n, is updated (step 15). After N_{temp} iterations at each temperature *Temp*, the temperature is reduced, and n is updated (step 17). This process continues until a stopping criterion is met (i.e., $Temp \leq min_Temp$). It is important to note that sometimes SA heuristics may not converge to a local optimum. In other words, there may be a neighboring solution S, in the neighborhood of *best_sol*, that gives a lower total cost (i.e., $TC(S) < TC(best_sol) = best_cost$). As a result, a steepest descent (local search) heuristic is implemented to ensure that the best solution obtained from the SAI heuristic (S^*) is a local optimum (steps 21–22). The details of the local search heuristic and the parameter setting are given below, but first the operations for obtaining a neighboring solution is defined.

Algorithm 3: SA($Temp_0$, *min_Temp*, N_{Temp}, α, p)

1: Initialize $Temp_0$, *min_Temp*, N_{Temp}, α, and p
2: $S_0 \leftarrow$ *CAI*(C_{tk}, r_{ti}) in SAI and $S_0 \leftarrow$ *CAII*(C_{tk}, r_{ti}) in SAII
3: Compute $TC(S_0)$ using Equation (18)
4: $best_sol \leftarrow S_0$, $best_cost \leftarrow TC(S_0)$
5: $n \leftarrow 0$, $Temp \leftarrow Temp_0$ % n is the iteration counter at current temperature *Temp*
6: **while** $Temp > min_Temp$ **do**
7: Randomly generate period t, ***move***, and obtain neighboring solution S
8: **if** S is feasible **then** % Use Equation (19) to check if S is feasible
9: $\Delta TC \leftarrow TC(S_0) - TC(S)$
10: **if** $\Delta TC > 0$ or $p = exp(-\Delta TC/Temp) >$ rand(0, 1) **then**
11: $S_0 \leftarrow S$
12: *Update(best_sol, best_cost)*

13: $S_0 \leftarrow$ *Look-Ahead/Look-Back Strategy*(S_0, t, ***move***) //use in SAII only
14: **end if**
15: $n \leftarrow n + 1$
16: **if** $n = N_{temp}$ **then**
17: $n \leftarrow 0$, $Temp \leftarrow \alpha$*$Temp$
18: **end if**
19: **end if**
20: **end while**
21: $S^* \leftarrow$ *LocalSearch*(*best_sol*, *best_cost*)
22: return S^*

In step 7 of the proposed SAI heuristic, there are two types of operations used to obtain a neighboring solution $\boldsymbol{S}$ from the current solution $\boldsymbol{S_0}$: drop/add and pairwise exchange. The drop/add operation removes one location assigned to a facility and reassigns a different location to it in one of the periods. However, the pairwise exchange operation interchanges the locations of two facilities (assigned to different locations) in one of the periods. For example, consider the solution $\boldsymbol{S_0}$ = {(2, 2, 3, 3, 3, 1), (1, 1, 3, 3, 3, 2)} obtained from the proposed construction algorithm CAI discussed earlier. The proposed SAI heuristic randomly selects an operation, say pairwise exchange. Next, a period is randomly selected, say period 2, and two facilities assigned to different locations are selected, say facilities 2 and 6. As a result, a neighboring solution $\boldsymbol{S}$ is obtained as follows by interchanging the locations of facilities 2 and 6 in period 2, and constraint (19) of the COP model is used to check the feasibility of the new solution $\boldsymbol{S}$ (step 8).

$$\boldsymbol{S_0} = \{(2, 2, 3, 3, 3, 1), (1, \underline{\mathbf{1}}, 3, 3, 3, \underline{\mathbf{2}})\} \quad \rightarrow \quad \boldsymbol{S} = \{(2, 2, 3, 3, 3, 1), (1, \underline{\mathbf{2}}, 3, 3, 3, \underline{\mathbf{1}})\}$$

As mentioned earlier, if the solution $\boldsymbol{S}$ is infeasible, the solution $\boldsymbol{S_0}$ remains the current solution. Otherwise, the total cost of the solution $\boldsymbol{S}$ (i.e., $TC(\boldsymbol{S})$) is obtained (step 9), and either $\boldsymbol{S_0}$ remains the current solution or $\boldsymbol{S}$ is selected as the current solution (i.e., set $\boldsymbol{S_0} = \boldsymbol{S}$) if either $\Delta TC > 0$ or $\Delta TC < 0$ and the acceptance probability $p = exp(-\Delta TC/Temp)$ is more than a randomly generated number between zero and one (i.e., rand(0, 1)) (step 10–11). In contrast, if the drop/add operation is selected randomly as well as a period, say period 1, then a facility is randomly selected, say facility 4 (assigned to location 3) and a different location is randomly selected between location 1 or 2, say 1. Therefore, a neighboring solution $\boldsymbol{S}$ is obtained as follows by replacing location 3 with location 1 for facility 4 in period 1, and constraint (19) of the COP model is used to check the feasibility of the new solution $\boldsymbol{S}$, as discussed earlier.

$$\boldsymbol{S_0} = \{(2, 2, 3, \underline{\mathbf{3}}, 3, 1), (1, 1, 3, 3, 3, 2)\} \quad \rightarrow \quad \boldsymbol{S} = \{(2, 2, 3, \underline{\mathbf{1}}, 3, 1), (1, 1, 3, 3, 3, 2)\}$$

Since there are M facilities and T periods where $M = 6$ and $T = 2$ for the above example, there are $T*M(M-1)/2 = 30$ possible pairwise exchanges. In contrast, since there are N locations where $N = 3$ in the example above, there are $T*M(N-1) = 24$ possible drop/add operations. It is important to note that for tightly constrained DGQAP instances, many of the neighboring solutions generated using either the pairwise exchange or drop/add operation may be infeasible. However, for loosely constrained DGQAP instances, many of the neighboring solutions generated from one of the operations may be feasible.

As mentioned earlier, SA heuristics may not converge to local optimums. In other words, the *best_sol* obtained after step 20 of Algorithm 3 may not be a local optimum. That is, there may be a neighboring solution S, in the neighborhood of *best_sol*, that gives a lower total cost (i.e., $TC(S) < TC(best_sol) = best_cost$). Therefore, a steepest descent (local search)

heuristic is implemented (step 21) to ensure the proposed SAI heuristic converges to a local optimum. The steps for the local search heuristic are as follows.

1. Given best_sol and best_cost.
2. Find all feasible neighboring solutions by considering all possible drop/add and pairwise exchange operations on best_sol.
3. Pick the best feasible neighboring solution, **S**, with respect to total cost, TC(**S**). If TC(**S**) < best_cost, set best_sol = **S**, best_cost = TC(**S**), and go to step 2. Otherwise, terminate heuristic and return local optimum S* = best_sol.

At the initial temperature, $Temp = Temp_0$, the probability of accepting non-improving solutions is high. This allows the heuristic to search the solution space without quickly converging to a poor local optimum. After a number of iterations, N_{Temp}, at the current temperature *Temp*, the temperature is reduced (i.e., set $Temp = \alpha * Temp$) where $0 < \alpha < 1$. As the temperature *Temp* reduces, the probability of accepting non-improving solutions is lower, allowing the heuristic to possibly converge to a good local optimum. The initial temperature ($Temp_0$) is determined using the acceptance probability equation $p = \exp(-\Delta TC/Temp_0)$ where $\Delta TC = TC(S_0) - TC(S)$, and p is the probability of accepting a non-improving neighboring solution S as the current solution. If the probability of accepting a neighboring solution with a cost of 10% above the cost of the initial solution S_0 is p, then $\Delta TC = 0.10TC(S_0)$. When solving the probability equation for $Temp_0$, the following equation is obtained.

$$Temp_0 = -\Delta TC/\ln(p) = -0.10TC(S_0)/\ln(p) \quad (20)$$

Before using Equation (20) to obtain the value for the parameter $Temp_0$, S_0 is obtained using CAI, and the parameter p is obtained experimentally as with the parameters α, N_{Temp}, and *min_Temp* which will be discussed in the computational results section of this paper.

It is believed that the SAI heuristic should work well for DGQAP instances with relatively low reassignment costs for each period, allowing for more reassignment of facilities in each period ($t = 2, \ldots, T$) while attempting to reduce total transportation cost. As a result, the starting solutions produced by the CAI is sufficient. In contrast, the starting solution produced by the CAII oftentimes reduce reassignment cost for each period by attempting to assign facilities to the same location for each period based on location capacities and facility requirements. Therefore, the proposed SAII heuristic, which will be discussed next and uses the CAII to produce a starting solution, works well for DGQAP instances with a relatively high reassignment cost during each period. The heuristic attempts to keep reassignment cost low by keeping facilities assigned to the same locations for each period.

3.2.2. Simulated Annealing II (SAII)

The SAII heuristic is the adaptation of the SAI heuristic with a look-ahead/look-back strategy. The only changes in the SAI heuristic (in Algorithm 3) are in steps 2 and 13. In step 2, the CAII heuristic is used to obtain an initial solution S_0, as opposed to CAI, as mentioned above. The only other change is the addition of the look-ahead/look-back strategy (step 13). Algorithm 4 shows the steps of the pseudo-code for the look-ahead/look-back strategy. Recall that step 7 in SAI (Algorithm 3) considers finding a neighboring solution using either the drop/add operation or the pairwise exchange operation. Instead of performing only one operation (i.e., move) at one randomly selected period t to produce a neighboring solution as in the SAI heuristic, in SAII, the same operation, move, is considered in previous and succeeding periods which may result in the same operation being performed in multiple periods. More specifically, if a period t ($t = 1, 2, \ldots, T$) is randomly selected, and an operation, move, is randomly selected to produce a neighboring

solution S (step 7), then if the neighboring solution S is feasible and either better than the current solution S_0 (i.e., $\Delta TC > 0$) or worse than the current solution S_0 (i.e., $\Delta TC < 0$) and $p = \exp(-\Delta TC/Temp) > \text{rand}(0, 1)$, set current solution $S_0 = S$ (steps 8–11), as in SAI. Update *best_sol* and *best_cost*, if necessary (step 12). However, in SAII the look-ahead/look-back strategy is implemented (step 13). That is, the operation, move, (in step 7 of Algorithm 3) is considered in the preceding period $t_p = t - 1$ and succeeding period $t_s = t + 1$ in Algorithm 4. As above, the operation, *move*, is accepted in periods $t_p = t - 1$ and/or $t_s = t + 1$ if it improves the total cost of the current solution or does not improve the total cost of the current solution but $p > \text{rand}(0, 1)$ (steps 1–7). If necessary, consider $t_p = t_p - 1$ where $t_p \geq 1$ and $t_s = t_s + 1$ where $t_s \leq T$ (step 9). Continue process until either $t_p < 1$ and $t_s > T$ (steps 2–16). In summary, the operation, *move*, is considered first in period t_p (step 3) until either

(i) ***S*** is infeasible (steps 13–14),
(ii) ***S*** is feasible, but $\Delta TC < 0$ and $p < rand(0, 1)$ (steps 10–11), or
(iii) $t_p < 1$ (step 16).

Algorithm 4: Look-Ahead/Look-Back Strategy(S_0, *t*, *move*)

```
1:  t_p ← t − 1, t_s ← t + 1
2:  repeat
3:      Perform move in period t_p (or t_s) and obtain solution S
4:      if S is feasible then      % Use Equation (19) to check if S is feasible
5:          ΔTC ← TC(S_0) − TC(S)
6:          if ΔTC > 0 or p = exp(−ΔTC/Temp) > rand(0, 1) then
7:              S_0 ← S
8:              Update(best_sol, best_cost)
9:              t_p ← t_p − 1 (t_s ← t_s + 1)
10:         else
11:             break
12:         end if
13:     else
14:         break
15:     end if
16: until t_p < 1 (t_s > T) do
17: return S_0
```

Similarly, the operation, move, is considered in period t_s (steps 2–16) in a similar fashion, except condition iii above is $t_s > T$ (Step 16). This strategy attempts to keep reassignment cost low by reassigning facilities to the same locations in multiple periods. Consider the case where a neighboring solution is obtained by randomly exchanging the locations of two facilities in a period t, and the optimal solution can be obtained by randomly exchanging the same locations of the facilities in the preceding and/or succeeding periods (i.e., periods $t - 1$ and $t + 1$). The probability of randomly selecting these facilities to exchange locations in periods $t - 1$ and/or $t + 1$ in consecutive iterations in the SAI heuristic is extremely small because of the randomness of the heuristic. However, the SAII heuristic has the capability of performing these operations in only one iteration. As a result, the additional operations implemented by the look-ahead/look-back search strategy are compensated for by possibly being able to perform more exchanges during an iteration, resulting in better solutions. In other words, since the SAII heuristic may perform more than one move during a single iteration, requiring more computational time per iteration, compared to the SAI heuristic, it is capable of producing higher quality solutions. Therefore,

the number of runs for SAI is either two or three times more than for the SAII heuristic, which will be illustrated below.

4. Computational Results

This section provides the results obtained from computational experiments performed to test the performances of the proposed SA heuristics. First, the proposed SA heuristics results are compared against the results obtained from solving the MILP model using CPLEX 12.4 for seven test problems (data set 1). This experiment was only performed to show that only very small-size problems can be solved optimally within a reasonable timeframe using the MILP model with CPLEX, and our proposed SA heuristics can perform well for solving small-size problems. In addition, the computational results of the proposed SA heuristics applied to 24 test problems (data set 2) are presented. Data set 1 consists of small-size DGQAP instances (where T = 2, 3, 4) whereas data set 2 consists of larger-size instances (where T = 5, 10). In all experiments, a PC (equipped with Microsoft Windows 10, an Intel Core processor i7 with a CPU speed of 2.90 GHz and 16 GB of RAM) was used to solve the test problems, and the proposed SA heuristics were coded using Matlab R2024a.

4.1. Set of Test Problems

Two data sets (31 test problems total) were developed for the DGQAP to test the performances of the SA heuristics. The test problems are generated following the procedure available in Cordeau et al. [7] for the GQAP and modified for the DGQAP. The authors labeled the test problems using the notation M-N-U where M = number of facilities, N = number of locations, and U = total percent of location space utilized by the facilities (a value between 1 and 100 used to control the tightness of the capacity constraints). Since the DGQAP has more than one period in the planning horizon, in this paper, U = average percent of location space utilized considering all periods. As a result, in this paper the test problems are labeled using the notation T-M-N-U where T = number of periods in the planning horizon. For instance, the small DGQAP instance given in Figure 1 is labeled as 2-6-3-92. Furthermore, the input data are generated for given values for T, M, N, and U as follows.

a_{tik} is generated according to a discrete uniform distribution in [1, . . ., 10^4 N].

f_{tij} and d_{tkl} are generated according to discrete uniform distribution in [1, . . ., 100].

r_{ti} is generated according to a discrete uniform distribution in [1, . . ., 100], as discussed below for the different cases (e.g., the r_{ti} remains the same for each instance for all periods).

C_{tk} is generated according to a discrete uniform distribution in [1, . . ., $\sum_{t\in T}\sum_{i\in N}\frac{r_{ti}}{U}$].

The seven test problems in data set 1 are generated as mentioned above where the space utilization (U) and the distance matrices are the same for each period (i.e., $d_{1kl} = d_{2kl} = \ldots = d_{Tkl}$). Also, the amount of space required for each facility i is the same for each period (i.e., $r_{1i} = r_{2i} = \ldots = r_{Ti}$). Note that the small DGQAP instance given in Figure 1, labeled as 2-6-3-92, does not consider the last restriction. That is, the amount of space required for a facility may change from one period to another (e.g., r_{11} = 2 but r_{21} = 3).

The 24 test problems in data set 2 are generated as mentioned above and are categorized into three different cases considering the different applications in managing facilities in container yards/warehouses, at construction sites and electric power plants during planned outages. Each case consists of eight test problems having either 5 or 10 periods. The first case considers the test problems in which the distances (d_{tkl}) between locations during each period are the same. Also, the facility requirements (r_{ti}) and space utilization (U) remain the same for each period. However, capacities of the locations (C_{tk}), the number of units of materials transported between facilities (f_{tij}), and the costs of assigning and reassigning facilities to locations changes for each period. The second case is like the first

case except that the distances (d_{tkl}) between locations changes in each period. In contrast, the third case considers the most general test problems where all the input data changes from one period to the next. Recall that these cases were discussed at the end of Section 1. See Table 1 below for the 24 test problems generated for the different types of cases for data set 2.

Table 1. Instances of the 24 test problems for data set 2.

Number of Periods	Case 1	Case 2	Case 3
5	5-15-5-35	5-15-5-55	5-15-5-75
	5-20-9-55	5-20-8-95	5-20-8-45
	5-30-10-65	5-30-20-75	5-30-20-80
	5-40-15-75	5-40-20-55	5-40-15-65
10	10-15-5-55	10-15-5-35	10-15-5-75
	10-20-8-95	10-20-10-75	10-20-9-55
	10-30-20-75	10-30-10-65	10-30-15-85
	10-40-15-65	10-40-20-75	10-40-15-65

4.2. Heuristic Parameter Settings

As discussed earlier, the values of the proposed SA heuristic parameters for SAI and SAII are obtained either theoretically or experimentally. More specifically, based on preliminary experiments, set cooling ratio, $\alpha = 0.99$, and minimum allowable temperature before terminating heuristic, $min_Temp = 0.0001$ for data set 1 for SAI and 0.001 for SAII. However, set $min_Temp = 0.001$ for data set 2 for both the SAI and SAII heuristics. The initial temperature, $Temp_0$, is obtained as discussed earlier using Equation (20). The number of iterations to be performed at each temperature (i.e., epoch length), N_{Temp}, is based on the number of possible neighboring solutions obtained using both drop/add and pairwise exchange operations, which is $T{*}M(N-1)$ and $T{*}M(M-1)/2$, respectively. After careful experimentation, it was decided to set $N_{Temp} = \lceil q[T*M(N-1)+T*M(M-1)/2] \rceil$. That is, N_{Temp} is q% of the maximum size of the neighborhood of $\boldsymbol{S_0}$, rounded above to the nearest integer. During the experimentation process, q = 40%, 50%, 60%, 70%, 80%, and 90% (and in between) was considered. It was decided for data set 1 to use q = 40% for SAI and q = 70% for SAII. However, the parameter used to determine the probability of accepting a non-improving solution $\boldsymbol{S}$ at the initial temperature $Temp_0$, p, was considered to take on the values p = 0.6, 0.7, 0.8, or 0.9. For data set 1, p = 0.7 (0.9) was used for SAI (II) heuristic. Recall that data set 1 considers only the simplest type of the DGQAP (i.e., case 1), but data set 2 considers all three cases. Therefore, there are nine different parameter settings used for both SAI and SAII for data set 2. For SAI, it was decided to use p = 0.6, 0.7, 0.8 where q = 40%, 50% and p = 0.9 where q = 40%, 50%, 60%. For SAII, it was decided to use p = 0.6, 0.7 where q = 50%, p = 0.8 where q = 60%, and p = 0.9 where q = 40%, 50%, 55%, 60%, 70%, 80%.

4.3. Computational Results for Data Set 1

The results for the proposed SA heuristics for data set 1 are compared against the results obtained from solving the proposed MILP model using the commercial solver CPLEX 12.4. The results of the computational experiments are given in Table 2 below. Notice the small DGQAP instance given in Figure 1 (i.e., 2-6-3-92) is listed first followed by the six test problems randomly generated as discussed above. Since SAII is able to produce higher quality solutions than SAI in a single run but requires more computational time, the number of runs for SAI is twice as long for SAII, which gives a fairer comparison and is discussed below. Each test problem was solved 20 (10) times, since the SAI (II) heuristics

are stochastic, and the outcomes may be different for different runs. Thus, the minimum (Min) and average (Avg) best total cost of the best solutions, as well as average computation time (Avg Time) in seconds are given for each problem. Furthermore, the optimal solutions (Opt TC) with run times were obtained for some of the problems using CPLEX (using the default settings), which are bolded, without asterisks. Notice the optimal solutions were obtained for four of the seven problems. However, CPLEX was unable to verify optimality for the other three problems, since CPLEX stopped due to insufficient memory (see Opt TC column with asterisks). In other words, the branch-and-bound search tree became very large and used up most of the RAM and terminated before obtaining or verifying optimality. Notice these problems ran for at least 10,916 s (or 3.03 h). Last, the percent deviation (% Dev) the best total cost obtained from the proposed SA heuristics is from the best total cost obtained from CPLEX is given in the last column for each problem. For instance, considering problem 3-15-4-85, CPLEX obtained its best solution where its total cost is 1,872,699, and the proposed SA heuristics obtained a solution with its Min total cost of 1,652,317, which is 11.8% below the Min total cost obtained from CPLEX. However, the SAII heuristic obtained a Min total cost which was 5.5% above the Min total cost obtained from CPLEX (and SAI) for problem 4-10-5-75. This is the only problem where CPLEX obtained a better solution than the SAII heuristic but notice the run time is a lot longer for CPLEX on average (i.e., 5.47 h = 19,674 s > 130.6 s = 13.06 (10 runs)). Hence, the proposed SA heuristics clearly outperformed CPLEX with respect to solution quality and computation time. However, the Friedman test reveals that there is no significant difference between the three methods when comparing Min values (p = 0.3679).

Table 2. Computational results for data set 1.

	Proposed SA Heuristics						**CPLEX**		
Problem	**SAI (20 runs)**			**SAII (10 runs)**					
	Min	**Avg**	**Avg Time (s)**	**Min**	**Avg**	**Avg Time (s)**	**Opt TC Min**	**Time (s)**	**% Dev**
2-6-3-92	**308**	309	0.95	**308**	**308**	1.38	**308**	0.05	0
2-15-4-85	**1,204,837**	1,231,778.95	6.28	**1,204,837**	**1,209,350**	10.40	1,248,242 *	19,644	−3.5
3-10-5-75	**710,036**	747,393.55	6.02	**710,036**	**739,104.8**	10.63	**710,036**	10,494	0
3-15-4-85	**1,652,317**	1,702,007.75	9.41	**1,652,317**	**1,679,416.7**	16.10	1,872,699 *	10,916	−11.8
3-9-4-55	**630,095**	636,850.75	2.71	**630,095**	**630,095**	4.70	**630,095**	661	0
4-10-5-75	**822,861**	907,919.3	8.01	865,255	**893,277**	13.06	**822,861 ***	19,674	0
4-12-5-35	**266,907**	340,770.7	5.63	**266,907**	**316,014.9**	10.96	**266,907**	24.84	0

* Ran out of memory (did not verify optimality).

When comparing the performances of SAI against SAII on data set 1, notice that the heuristics obtained the same minimum total cost (Min) for six of the seven problems. However, for problem 4-10-5-75 SAII did not obtain the Min of 822,861 but a total cost which is 5.5% higher. Although SAI average run time (Avg Time) is shorter, the average total run time was longer by approximately 30 s (i.e., 8.01 (20 runs) = 160.2 > 130.6 = 13.06 (10 runs)). As mentioned earlier, the look-ahead/look-back strategy of SAII performs multiple moves and requires more computation time per iteration. Therefore, the number of runs for SAI is doubled in order to fairly compete with SAII. Although it seems to make a difference for this problem, it does not for the other problem or for the problems of data set 2. More specifically, for data set 1, which is the simplest case (i.e., case 1 where the space utilization and the distance matrices are the same in each period) of the three cases mentioned in Section 4.1, SAI performed very well. However, for data set 2, which considers all three

cases with larger-size problems, SAI heuristic does not perform as well on the more difficult problems (i.e., cases 2 and 3) which will be discussed below. Next, notice that the average total cost (Avg) for SAII is lower for all the problems (lowest Avgs are bolded). Therefore, one can argue that SAII outperformed SAI. This was confirmed by the Wilcoxon signed-rank test, when comparing Avg for SAI and SAII (p = 0.018, r = 0.89). In other words, there is a significant difference between the Avg values for SAI and SAII with a large effect size.

4.4. Computational Results for Data Set 2

The proposed heuristics SAI and SAII was used to solve data set 2 using the parameter settings previously discussed in Section 4.1. Each test problem was solved 30 times and 10 times using the SAI and SAII heuristics, respectively. This will be discussed in detail below. Nevertheless, considering all the possible combinations of setting the parameters (for parameters p and q) given in Section 4.1 and solving the problems using the proposed SA heuristics, the best total costs and computational times obtained by each were recorded. See the results of the computational experiments in Table 3 where the first eight problems represent case 1-type problems, the second eight problems represent case 2-type problems, and the third eight problems represent case 3-type problems. When comparing the minimum best total cost (Min) obtained for SAI and SAII, SAI and SAII obtained best Min 6 and 19 times (best Mins are bolded), respectively. As the problems become more difficult (e.g., Case 3 problems 17-24), SAI obtained only one solution which gives the best Min. See problem 17. When comparing the average total cost (Avg) for SAI and SAII, SAII obtained lowest Avg (lowest Avgs are bolded) for all problems except for four problems (i.e., problems 8, 11, 12, and 16). As in data set 1, SAI average run time (Time) is shorter, but average total run time is longer, approximately twice as long as SAII. Notice times are given in seconds. For example, the largest discrepancy between average total run times is with problem 10 where SAI ran 2.146 times as long as SAII (i.e., 3573.3 s = 119.11 (30 runs) > 1664.7 s = 166.47 (10 runs)). Nevertheless, SAII outperformed SAI for data set 2. This was confirmed by the Wilcoxon signed-rank test. When comparing Min for SAI and SAII (p = 0.0285, r = 0.45) and Avg for SAI and SAII (p = 0.0397, r = 0.42), both cases represent a medium effect. Last, the percent deviation (% Dev), i.e., the best minimum total cost obtained from SAII is from the best minimum total cost obtained from SAI, is given in the last column. Notice the largest differences (i.e., negative and positive) in % Dev are for problems 13 and 16 of case 2. That is, for problem 13 the best Min obtained from SAII is 1.44% below best Min obtained from SAI, but 1.08% higher for problem 16. In summary, although SAII outperformed SAI with respect to solution quality and computation time, SAII solutions were only better by at most 1.44% Dev.

Table 3. Computational results for data set 2.

		SAI (30 Runs)			**SAII (10 Runs)**			
#	**Problem**	**Min**	**Avg**	**Avg Time (s)**	**Min**	**Avg**	**Avg Time (s)**	**% Dev**
1	5-15-5-35	**922,200**	945,672	9.89	**922,200**	**936,132.889**	16.66	0
2	5-20-9-55	5,379,995	5,460,632.22	41.16	**5,341,049**	**5,434,662.78**	68.18	−0.72
3	5-30-10-65	13,431,191	13,520,605.6	106.01	**13,317,816**	**13,453,558.1**	173.9	−0.84
4	5-40-15-75	27,560,439	27,745,720	328.05	**27,530,778**	**27,702,223.9**	506.7	−0.11
5	10-15-5-55	1,623,750	1,657,796.44	27.67	**1,600,610**	**1,646,887.11**	49.38	−1.43
6	10-20-8-95	**18,193,911**	18,364,936.2	252	18,213,573	**18,339,336.4**	359.68	0.11
7	10-30-20-75	39,161,467	39,414,923.1	679.79	**39,010,900**	**39,250,770**	1098.28	−0.38
8	10-40-15-65	**50,169,966**	**50,506,268**	551.35	50,262,851	50,644,072.67	1063.97	0.19

Table 3. *Cont.*

		SAI (30 Runs)			SAII (10 Runs)			
#	**Problem**	**Min**	**Avg**	**Avg Time (s)**	**Min**	**Avg**	**Avg Time (s)**	**% Dev**
9	5-15-5-55	2,286,825	2,321,354.56	13.27	**2,283,483**	**2,319,463.89**	20.82	−0.15
10	5-20-8-95	8,641,452	8,686,293.67	119.11	**8,588,225**	**8,669,305.67**	166.47	−0.62
11	5-30-20-75	**19,541,977**	**19,739,468**	323.11	19,692,003	19,788,634.78	486.99	0.77
12	5-40-20-55	27,392,715	**27,527,123**	337.82	**27,271,823**	27,571,778.44	583.04	−0.44
13	10-15-5-35	1,646,841	1,660,313.22	20.01	**1,623,122**	**1,633,250.44**	41.23	−1.44
14	10-20-10-75	15,341,257	15,405,018.7	122.99	**15,300,098**	**15,364,792.4**	201.64	−0.27
15	10-30-10-65	28,288,641	28,429,625.9	207.94	**28,234,507**	**28,385,578.8**	407.16	−0.19
16	10-40-20-75	**65,959,299**	**66,672,030**	965.77	66,668,768	**66,793,629.44**	1630.85	1.08
17	5-15-5-75	**2,904,174**	2,944,380.56	18.16	2,907,949	**2,941,021**	27.36	0.13
18	5-20-8-45	4,095,000	4,123,576.33	30.99	**4,063,326**	**4,115,795.22**	53.1	−0.77
19	5-30-20-80	20,518,420	20,649,265.9	374.43	**20,461,707**	**20,624,964.8**	549.39	−0.28
20	5-40-15-65	28,921,669	29,019,295	280.66	**28,865,557**	**28,958,645.3**	469.37	−0.19
21	10-15-5-75	5,763,799	5,790,489.56	36.99	**5,697,149**	**5,781,182**	59.05	−1.16
22	10-20-9-55	11,120,614	11,226,135.4	86.38	**11,114,752**	**11,207,295.2**	168.23	−0.05
23	10-30-15-85	38,612,373	38,841,027.4	560	**38,394,484**	**38,751,672.9**	873.47	−0.56
24	10-40-15-65	57,216,830	57,423,855	554.89	**57,150,774**	**57,365,444.9**	1070.64	−0.12
	# Best	6	4		19	20		

5. Conclusions

In this paper, the GQAP is extended to include multiple time periods and is called the DGQAP. Mathematical models as well as construction algorithms (i.e., CAI and CAII) and improvement algorithms were developed and presented for the proposed problem. The improvement algorithms consist of an SA and a steepest descent heuristic called SAI and SAII. The SAI heuristic is a direct adaptation of SA to the DGQAP, and SAII is like SAI except that it includes a look-ahead/look-back strategy. Computational experiments were conducted on a small data set (i.e., data set 1) and a large data set (i.e., data set 2), generated randomly, and it showed promising results for the proposed SA heuristics. However, SAII clearly outperformed SAI for data set 2. It is believed that the SAI heuristic did not perform well because of its randomness, and its inability to change locations of either one or two facilities in multiple periods simultaneously, especially for tightly constrained problems ($U > 70$). More specifically, for tightly constrained problems where rearrangement cost is relatively high, changing the location of either one or two facilities in a period (i.e., performing either a drop/add or pairwise exchange operation during one iteration) may require changing the location of one or two of those facilities in either preceding or succeeding periods (i.e., at the next several consecutive iterations) to keep total cost low. However, due to the randomness of the SAI heuristic, the chances of changing the locations of one or both of those facilities in the necessary preceding or succeeding periods in the next several consecutive iterations are very low. In contrast, since the SAII heuristic considers changing the locations of one or both facilities in preceding and succeeding periods during the same iteration (i.e., look-ahead/look-back strategy), it outperformed the SAI heuristic.

The following recommendations are given for future research. Consider extending DGQAP by considering the stochastic nature of the problem (e.g., units of materials flowing between facilities are stochastic) and develop solution techniques for the problems of the proposed algorithms. Adapt other metaheuristics (e.g., tabu search, variable neighborhood search, or genetic algorithm) for solving the DGQAP for a more robust performance com-

parison. Develop hybrid heuristics for the DGQAP which considers the combination of two metaheuristics to overcome the drawbacks of purely stochastic and deterministic heuristics.

Author Contributions: Conceptualization, A.M.; methodology, A.M. and Y.D.; software, Y.D.; validation, Y.D. and A.M.; formal analysis, A.M.; investigation, A.M. and Y.D.; resources, A.M.; data curation, A.M.; writing—original draft preparation, Y.D. and A.M.; writing—review and editing, Y.D. and A.M.; visualization, Y.D. and A.M.; supervision, A.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors on request.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Koopmans, T.C.; Beckmann, M.J. Assignment problems and the location of economic activities. *Econometrica* **1957**, *25*, 53–76. [CrossRef]
2. Sahni, S.; Gonzales, T. P-complete Approximation Problems. *J. ACM* **1976**, *23*, 555–565. [CrossRef]
3. Burkard, R.E.; Cela, E.; Pardalos, P.M.; Pitsoulis, L.S. The Quadratic Assignment Problem. In *Handbook of Combinatorial Optimization*; Du, D.-Z., Pardalos, P.M., Eds.; Kluwer: Boston, MA, USA, 1998; pp. 1713–1809. [CrossRef]
4. Loiola, E.M.; de Abreu, N.M.M.; Boaventura-Netto, P.O.; Hahn, P.; Querido, T. A survey for the quadratic assignment problem. *Eur. J. Oper. Res.* **2007**, *176*, 657–690. [CrossRef]
5. Silva, A.; Coelho, L.C.; Darvish, M. Quadratic assignment problem variants: A survey and an effective parallel memetic iterated tabu search. *Eur. J. Oper. Res.* **2021**, *292*, 1066–1084. [CrossRef]
6. Lee, C.-G.; Ma, Z. *The Generalized Quadratic Assignment Problem*; Research Report; Department of Mechanical and Industrial Engineering, University of Toronto: Toronto, ON, Canada, 2004.
7. Cordeau, J.-F.; Gaudioso, M.; Laporte, G.; Moccia, L. A memetic heuristic for the generalized quadratic assignment problem. *INFORMS J. Comput.* **2006**, *18*, 433–443. [CrossRef]
8. Hahn, P.M.; Kim, B.-J.; Guignard, M.; MacGregor Smith, J.; Zhu, Y.-R. An algorithm for the generalized quadratic assignment problem. *Comput. Optim. Appl.* **2008**, *40*, 351–372. [CrossRef]
9. Pessoa, A.A.; Hahn, P.M.; Guignard, M.; Zhu, Y.-R. Algorithms for the generalized quadratic assignment problem combining lagrangean decomposition and the reformulation-linearization technique. *Eur. J. Oper. Res.* **2010**, *206*, 54–63. [CrossRef]
10. Guignard, M. Strong RLT1 bounds from decomposable Lagrangean relaxation for some quadratic 0–1 optimization problems with linear constraints. *Ann. Oper. Res.* **2020**, *286*, 173–200. [CrossRef]
11. Holland, J.H. *Adaptation in Natural and Artificial Systems*; University of Michigan Press: Ann Arbor, MI, USA, 1975.
12. Glover, F. Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.* **1986**, *13*, 533–549. [CrossRef]
13. Mateus, G.R.; Resende, M.G.C.; Silva, R.M.A. GRASP with Path-relinking for the generalized quadratic assignment problem. *J. Heuristics* **2011**, *17*, 527–565. [CrossRef]
14. McKendall, A.; Li, C. A tabu search heuristic for a generalized quadratic assignment problem. *J. Ind. Prod. Eng.* **2017**, *34*, 221–231. [CrossRef]
15. Greistorfer, P.; Staněk, R.; Maniezzo, V. A tabu search matheuristic for the generalized quadratic assignment problem. In *Metaheuristics*; MIC 2022; Lecture Notes in Computer Science; Di Gaspero, L., Festa, P., Nakib, A., Pavone, M., Eds.; Springer: Cham, Switzerland, 2023; Volume 13838, pp. 544–553. [CrossRef]
16. Fathollahi-Fard, A.M.; Wong, K.Y.; Aljuaid, M. An efficient adaptive large neighborhood search algorithm based on heuristics and reformulations for the generalized quadratic assignment problem. *Eng. Appl. Artif. Intell.* **2023**, *126*, 106802. [CrossRef]
17. McKendall, A.; Dhungel, Y. A simulated annealing algorithm for the generalized quadratic assignment problem. *Algorithms* **2024**, *17*, 540. [CrossRef]
18. Sohrabi, M.; Fathollahi-Fard, A.M.; Gromov, V.A.; Dulebenets, M.A. A genetic engineering algorithm for the generalized quadratic assignment problem. *Neur. Comput. Appl.* **2025**, *37*, 12253–12279. [CrossRef]
19. Zou, D.; Gao, L.; Li, S.; Wu, J.; Wang, X. A novel global harmony search algorithm for task assignment problem. *J. Syst. Soft.* **2010**, *83*, 1678–1688. [CrossRef]
20. Unal, Y.Z.; Uysal, O. A new mixed integer programming model for curriculum balancing: Application to a Turkish university. *Eur. J. Oper. Res.* **2014**, *238*, 339–347. [CrossRef]

21. Elbeltagi, E.; Hegazy, T.; Eldosouky, A. Dynamic layout of construction temporary facilities considering safety. *J. Constr. Eng. Manag.* **2004**, *130*, 534–541. [CrossRef]
22. Rosenblatt, M.J. The dynamics of plant layout. *Manag. Sci.* **1986**, *32*, 76–86. [CrossRef]
23. McKendall, A.R.; Shang, J.; Kuppusamy, S. Simulated annealing heuristics for the dynamic facility layout problem. *Comput. Oper. Res.* **2006**, *33*, 2431–2444. [CrossRef]
24. Khajemahalle, L.; Emami, S.; Keshteli, R.N. A hybrid nested partitions and simulated annealing algorithm for dynamic facility layout problem: A robust optimization approach. *INFOR* **2021**, *59*, 74–101. [CrossRef]
25. Kogan, K.; Shtub, A.; Levit, V. DGAP-The dynamic generalized assignment problem. *Ann. Oper. Res.* **1997**, *69*, 227–239. [CrossRef]
26. Mazzola, J.B.; Neebe, A.W. A generalized assignment model for dynamic supply chain capacity planning. *Nav. Res. Logist.* **2012**, *59*, 470–485. [CrossRef]
27. Moccia, L.; Cordeau, J.-F.; Monaco, M.F.; Sammarra, M. A column generation heuristic for a dynamic generalized assignment problem. *Comput. Oper. Res.* **2009**, *36*, 2670–2681. [CrossRef]
28. McKendall, A.R.; Noble, J.S.; Klein, C.M. Simulated annealing heuristics for managing resources during planned outages at electric power plants. *Comput. Oper. Res.* **2005**, *32*, 107–125. [CrossRef]
29. Wang, S.; Sun, W.; Huang, M. An adaptive large neighborhood search for the multi-depot dynamic vehicle routing problem with time windows. *Comput. Ind. Eng.* **2024**, *191*, 110122. [CrossRef]
30. Zhang, L.; Li, Q. Approximation algorithm for dynamic facility location problem. *J. Comb. Optim.* **2025**, *49*, 48. [CrossRef]
31. Kirpatrick, S.; Gelatt, C.D.; Vecchi, M.P. Optimization by simulated annealing. *Science* **1983**, *220*, 671–680. [CrossRef]
32. Huo, L.; Zhu, J.; Wu, G.; Li, Z. A novel simulated annealing based strategy for balanced UAV task assignment and path planning. *Sensors* **2020**, *20*, 4769. [CrossRef]
33. Dai, Z.; Zhang, Z.; Chen, M. The home health care location-routing problem with a mixed fleet and battery swapping stations using a competitive simulated annealing algorithm. *Expert Syst. Appl.* **2023**, *228*, 120374. [CrossRef]
34. Leelertkij, T.; Buddhakulsomsiri, J.; Huynh, V.-N. A multi-thread simulated annealing for multi-objective vehicle routing problem with time windows and demand priority. *Comput. Ind. Eng.* **2025**, *207*, 111253. [CrossRef]
35. Tole, K.; Moqa, R.; Zheng, J.; He, K. A simulated annealing approach for the circle bin packing problem with rectangular items. *Comput. Ind. Eng.* **2023**, *176*, 109004. [CrossRef]
36. Lin, S.-W.; Guo, S.; Wu, W.-J. Applying the simulated annealing algorithm to the set orienteering problem with mandatory visits. *Mathematics* **2024**, *12*, 3089. [CrossRef]
37. Rosati, R.M.; Schaerf, A. Multi-neighborhood simulated annealing for the capacitated dispersion problem. *Expert Syst. Appl.* **2024**, *255*, 124484. [CrossRef]
38. Ceschia, S.; Schaerf, A. Multi-neighborhood simulated annealing for the capacitated facility location problem with customer incompatibilities. *Comput. Ind. Eng.* **2024**, *188*, 109858. [CrossRef]
39. Da Ros, F.; Di Gaspero, L.; Lackner, M.-L.; Musliu, N.; Winter, F. Multi-neighborhood simulated annealing for the oven scheduling problem. *Comput. Oper. Res.* **2025**, *177*, 106999. [CrossRef]
40. Van Bulck, D.; Goossens, D.; Schaerf, A. Multi-neighbourhood simulated annealing for the ITC-2007 capacitated examination timetabling problem. *J. Sched.* **2025**, *28*, 217–232. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.