Memoirs
of the National Defense Academy,
Japan, Vol. 62, No. 2,
(2022), pp. 1– 10

< Paper >

Dan-Kou
– A Grouping and Load Distributing Problem –

By Syotaro KOIDE*, Yuzo SAKAI** and Seiji KATAOKA***

Abstract

Dan-Kou is a type of cross-country race that is held at the National Defense Academy. Around 100 cadets are divided into 12 squads, and each cadet must run the race with a loaded pack. During the race, the more vigorous cadets are allowed to aid their exhausted squadmates by carrying extra loads. The recorded time for a squad is marked at the moment when its last member arrives at the goal. Hence, the key to winning Dan-Kou is in strategic squad grouping and load distribution. This paper proposes a new lower bound strategy and develops a branch-and-bound algorithm. The proposed algorithm succeeds in solving modeled instances thousands of times faster than a cutting-edge solver, but the solvable sizes are too small. In order to solve life-size instances, we apply a matheuristic strategy in which the proposed algorithm is used as an optimizing tool in neighborhoods.

Keywords: New lower bounds, Branch-and-bound, Matheuristic

1. Introduction — What is "Dan-Kou"?

This study originates from the competitive sport called *Dan-Kou* held at the National Defense Academy. In Japanese, "Dan" means "passing through" and "Kou" means "outskirts", so Dan-Kou is a type of cross-country race. In our academy, every cadet belongs to one of four battalions, and the battalions compete in various competitive sports throught the year. Dan-Kou is available for junior cadets and, in each battalion, all the approximately 100 cadets, 10% of which are women, are divided into 12 squads, so each squad consists of 8 or 9 cadets. Each cadet must run a course around 7 km long while carrying a loaded pack weighing around 9 kg that includes items such as some blankets, a water bottle, and several tools. During the race, the more vigorous cadets are allowed to aid their exhausted squadmates by carrying extra loads. The partial giving and receiving of loads—just a blanket, for example—is permitted. The recorded time for a squad, called 'squad time', is marked at the moment when its last member arrives at the goal. The total of the 12 squad times is the final score for their belonging battalion. Hence, the key to winning Dan-Kou is in strategic squad grouping and load distribution.

As an optimization problem, Dan-Kou can be categorized as either a grouping problem or an assignment problem. There are a rich variety of grouping or assignment problems in the litera-

* Undergraduate Student(66th term)
** Undergraduate Student(66th term)
*** Professor

ture.[7] However, our somewhat time-intensive search could find hardly any absolutely referable studies that simultaneously consider load distribution. Hence, in Section 2, we introduce two cadets' graduation theses that take mathematical programming approaches. These studies were not able to fully obtain an optimal squad grouping and load distribution for an actual Dan-Kou race, but they inspired some remarkable ideas for this paper. One important idea centers around a new lower bound, which is a major contribution to this paper and is discussed in Section 3. Using the proposed lower bound, Section 4 discusses a branch-and-bound algorithm. In the case of smaller instances, the algorithm succeeds in obtaining optimal solutions thousands of times faster than Gurobi, a cutting-edge solver. For larger, life-size instances, Section 5 utilizes a matheuristic approach, which makes use of the proposed algorithm as an optimizing tool in neighborhoods. Concluding remarks are made in Section 6.

2. Past Researches

2.1 Hamamoto's study

Hamamoto[4] presented the following mathematical programming model (Formulation H).

Set and suffix

$i \in C = \{1, 2, \ldots, n\}$: the set of cadets,

$j \in S = \{1, 2, \ldots, m\}$: the set of squads,

S_j: the set of cadets in squad j, ($S_{j_1} \cap S_{j_2} = \emptyset, j_1 \neq j_2$, $|S_1| + |S_2| + \cdots + |S_m| = n$, these sets are as evenly sized as possible.)

$k \in M = \{0, 1, 2\}$: the set of modes (0: no load; 1: normal 9 kg load; 2: twice as heavy 18 kg load). Hamamoto limited the modes to 0, 1, and 2. Hereinafter, we refer to these as 0-load, 1-load, and 2-load, respectively.

Given data

t_{ik}: the running time of cadet i with k-load.

Formulation H

$$\text{min.} \quad \sum_{j \in S} T_j \ , \tag{1}$$

$$\text{s.t.} \quad \sum_{j \in S} \sum_{k \in M} x_{ijk} = 1, \qquad \forall i \in C, \tag{2}$$

$$\sum_{i \in C} \sum_{k \in M} x_{ijk} = |S_j|, \quad \forall j \in S, \tag{3}$$

$$\sum_{i \in C} \sum_{k \in M} k x_{ijk} = |S_j|, \quad \forall j \in S, \tag{4}$$

$$\sum_{k \in M} t_{ik} x_{ijk} \leq T_j, \ \forall i \in C, j \in S, \tag{5}$$

$$T_{j_1} \leq T_{j_2}, \qquad \forall |S_{j_1}| = |S_{j_2}|, \ j_1 < j_2, \tag{6}$$

$$x_{ijk} \in \{0, 1\}, T_j \geq 0. \tag{7}$$

Decision variables

x_{ijk}: it takes 1 if cadet i runs in squad j with k-load, and 0 if otherwise.

T_j: the squad time for squad j.

The constraints (6) are for non-decreasingly ordering the squad times with equal size. These constraints are useful for avoiding wasteful enumeration, and afterwards, they play a role in cutting branches in the search tree.

Hamamoto tried to solve Formulation H using NUOPT, now called "Numetical Optimizer".[8] He reported that the solver can solve Formulation H for only up to two squads. Based on this ability, Hamamoto proposed an algorithm that divides the set of cadets into two, then recursively divides each half-set into two until each divided set consists of 8 or 9 cadets.

His algorithm gives solutions where some cadets carry 0-load and other cadets carry 2-load, even in the top squad. In an actual Dan-Kou race, however, the cadets in the top squad usually carry their own 1-load. Hamamoto himself then concluded that the proposed algorithm failed to reflect an actual Dan-Kou situation.

2.2 Kozuka's study

Solving Formulation H normally results in obtaining solutions where each cadet keeps running from start to the goal with just one discrete load, either a 0-, 1-, or 2-load. This means that throughout the race, cadets who run in the front tend to stay in the front, and likewise cadets who run in the back tend to stay in the back. Hence, the distance between those in front and those in back among squad members gets to be larger and larger, so in practice it is impossible to mutually give and receive any loads.

The second thesis, Kozuka,[6] considered strategically adjusting loads. The expected solutions would make it possible to align squad members throughout the race. The resultant loads would inform each squad member whether he or she was on the supporting side or on the supported side, and the extent of their loads. This knowledge could be reflected in an actual Dan-Kou race.

Kozuka called Dan-Kou "a min-max grouping and load distributing problem"[6] as a purely mathematical programming model. So in this paper, we use the abbreviation "MMGLD" when considering a purely mathematical programming model and not an actual Dan-Kou race.

In order to express the partial load distribution and easily analyze the mathematical model, Kozuka assumed there to be piecewise linear functions between the [0,1]-load, the [1,2]-load, and the running time. Additionally, as a natural assumption, the running time for each cadet is expressed as a monotonically increasing function as the load increases from 0 to 2. Most of the parameters follow Formulation H, but as for the decision variables, Kozuka distinguishes them between the [0,1]-load and the [1,2]-load as follows:

x_{ij}: it takes 1 if cadet i runs in squad j with a [0,1]-load and 0 otherwise.

y_{ij}: it takes 1 if cadet i runs in squad j with a [1,2]-load and 0 otherwise.

Formulation K

$$\text{min.} \quad \sum_{j\in S} T_j, \tag{8}$$

$$\text{s.t.} \quad \sum_{j\in S}(x_{ij}+y_{ij}) = 1, \qquad \forall i, \tag{9}$$

$$x_{ij}+y_{ij} \le 1, \qquad \forall i,j, \tag{10}$$

$$\sum_{i\in C}(x_{ij}+y_{ij}) = |S_j|, \qquad \forall j, \tag{11}$$

$$x_{ij}-\alpha_{ij}-\beta_{ij} = 0, \qquad \forall i,j, \tag{12}$$

$$y_{ij}-\gamma_{ij}-\delta_{ij} = 0, \qquad \forall i,j, \tag{13}$$

$$\sum_{i\in C}(\beta_{ij}+(y_{ij}+\delta_{ij})) = |S_j|, \quad \forall j, \tag{14}$$

$$(t_{i0}\alpha_{ij}+t_{i1}\beta_{ij}) + (t_{i1}\gamma_{ij}+t_{i2}\delta_{ij}) \le T_j \qquad \forall i,j, \tag{15}$$

$$T_{j_1} \le T_{j_2}, \qquad \forall |S_{j_1}| = |S_{j_2}|, \; j_1 < j_2, \tag{16}$$

$$x_{ij}, y_{ij} \in \{0,1\},$$
$$0 \le \alpha_{ij}, \beta_{ij}, \gamma_{ij}, \delta_{ij} \le 1. \tag{17}$$

In Formulatoin K, the constraints (12) and (13) force the activation of the conditions for convex combination with parameters $\alpha_{ij}, \beta_{ij}, \gamma_{ij}$, and δ_{ij}.

Kozuka tried to solve MMGLD using Formulation K and Gurobi, but the linearly relaxed lower bounds were so small that even toy instances could not be solved practically. For example, when $n = 16$, $m = 2$, and $|S_1| = |S_2| = 8$, the mixed-integer optimal value is 4985.75 ($T_1 = 2226.71$, $T_2 = 2759.04$) but the linearly-relaxed lower bound is 2759.04 ($T_1 = 1379.52$, $T_2 = 1379.52$), about half of the optimal value. Observing each decision variable x_{ij} and y_{ij} in linearly-relaxed solutions, they are fragmentary scattered all over the cadets and squads. Small x_{ij} and y_{ij} also make the parameters small by (12) and (13). Therefore, the left-hand side of (15) also becomes small. If the number of squads becomes large, the linearly-relaxed lower bounds would take hopeless values.

2.3 Feasible load-distribution

Although Kozuka improved the defects in Formulation H, he also could not succeed in solving the actual Dan-Kou problem or even a small-sized MMGLD to optimality. However, for a given assignment, Kozuka proposed an easy way for obtaining the optimal load distribution that does not depend on mathematical programming.

Figure 1 shows an example; for simplicity, four cadets, A, B, C, and D, are to be assigned to a squad j. Given a target time, the dashed line in the figure, the corresponding loads that are to arrive at the target time are easily calculated from each piecewise linear and monotonically increasing function. Hence, if we can find the time at which the total loads become just four, then these loads will give a feasible, optimal load distribution. The searched time is exactly the optimal squad time T_j for squad j. In the exceptional case where a cadet whose 0-load running time is later than the searched time—see cadet D and $T_j < t_{D0}$ in the figure for example—then T_j is replaced with t_{D0}.

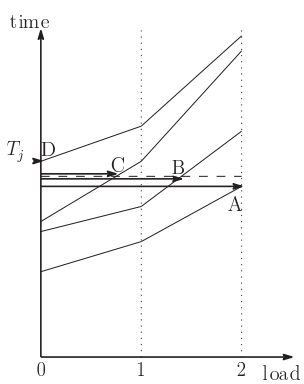

Fig. 1 Loads, running times, and squad time

3. Lower Bound not Depending on Mathematical Programming

The proposed upper bound procedure discussed in Section 2.3 also gives an idea for how to obtain the lower bounds for MMGLD. The proposed lower bound relaxes the number of cadets in each squad.

3.1 Fixing/unfixing no cadet to squads

As for the relationship between the load and the running time for each cadet i, we assume a piecewise linear and monotonically increasing function that passes through $(0, t_{i0})$, $(1, t_{i1})$, and $(2, t_{i2})$, $(t_{i0} < t_{i1} < t_{i2})$. Hence, considering its inverse function, for a given time $t (t_{i0} \le t \le t_{i2})$ we can calculate the load $\ell_i(t)$ for cadet i that is needed in order to arrive at the goal at time t. As exceptional cases, if $t < t_{i0}$, then cadet i cannot arrive at the goal at time t even with 0-load, so we set $\ell_i(t) := 0$, and if $t_{i2} < t$, cadet i can carry a 2-load before time t, so we set $\ell_i(t) := 2$. See (18).

$$\ell_i(t) := \begin{cases} 0, & t < t_{i0}, \\ \dfrac{t - t_{i0}}{t_{i1} - t_{i0}}, & t_{i0} \le t < t_{i1}, \\ \dfrac{t - t_{i1}}{t_{i2} - t_{i1}} + 1, & t_{i1} \le t < t_{i2}, \\ 2, & t_{i2} \le t. \end{cases} \tag{18}$$

In MMGLD, there exist n such functions and the total amount of load that is carried by all cadets at time t is expressed as follows:

$$L(t) := \sum_{i \in C} \ell_i(t). \tag{19}$$

Since each $\ell_i(t)$ is a piecewise linear and monotonically increasing function between t_{i0} and t_{i2}, and constant for $t < t_{i0}$ and $t_{i2} < t$, then the sum of these functions $L(t)$ is also a piecewise linear and monotonically increasing function between the fastest running time for the 0-load cadets ($\min_i\{t_{i0}\}$) and the slowest running time for the

Table 1 Gurobi results, LP, and proposed lower bounds: 16;(8)2, $\rho = 0.5$

	Formulation K & Gurobi				Proposed LB	
seed	CPU	OptVal	(T_1, T_2)	LPLB	LB	(T_1^L, T_2^L)
111	2.17	4985.75	(2226.71,2759.04)	2759.04	4977.82	(2218.78,2759.04)
222	0.93	4810.42	(2191.94,2618.49)	2618.49	4809.22	(2190.74,2618.48)
333	0.42	4761.47	(2254.67,2506.80)	2506.80	4746.69	(2239.89,2506.80)
444	0.54	4635.45	(2150.09,2485.36)	2404.87	4551.24	(2146.36,2404.87)
555	1.01	4870.49	(2237.60,2632.89)	2632.89	4870.49	(2237.60,2632.89)

Table 2 Gurobi results, LP, and proposed lower bounds: 24;(8)3, $\rho = 0.5$

	Formulation K & Gurobi				Proposed LB	
seed	CPU	OptVal	(T_1, T_2, T_3)	LPLB	LB	(T_1^L, T_2^L, T_3^L)
111	27.61	7282.27	(2165.69,2313.84,2803.20)	2803.20	7224.50	(2148.08,2273.22,2803.20)
222	15.93	7000.15	(2133.11,2290.58,2576.46)	2569.53	6880.68	(2077.44,2233.71,2569.53)
333	241.29	7253.65	(2235.75,2323.46,2694.44)	2694.45	7174.82	(2186.75,2293.62,2694.44)
444	161.93	6944.11	(2146.60,2295.73,2501.78)	2498.18	6838.90	(2106.44,2234.27,2498.18)
555	16.87	6971.48	(2133.94,2290.72,2546.82)	2498.60	6866.86	(2117.82,2250.43,2498.60)

2-load cadets ($\max_i\{t_{i2}\}$). The turning points that shape the piecewise function are t_{ik} ($i = 1, \ldots, n; k = 0, 1, 2$). We sort these $3n$ t_{ik}'s in increasing order and rename them as τ_p ($p = 1, \ldots, 3n$). If λ is a desired load that we want to carry, then by searching for an r that satisfies

$$L(\tau_{r-1}) \leq \lambda < L(\tau_r), \tag{20}$$

the minimum time for which all cadets can carry the load λ to the goal is calculated as follows:

$$\frac{\lambda - L(\tau_{r-1})}{L(\tau_r) - L(\tau_{r-1})} \cdot (\tau_r - \tau_{r-1}) + \tau_{r-1}. \tag{21}$$

If we set λ to $|S_1|$, then (21) gives a lower bound T_1^L for T_1, the squad time for cadets assigned to squad S_1. Since T_1^L is the squad time for carrying the $|S_1|$-load by all cadets, it gives a lower bound for the squad time when only the cadets in S_1 carry the same amount of load.

As for S_2, by setting λ to $|S_1| + |S_2|$ and assuming that the load is carried by all cadets, (21) gives a lower bound T_2^L of T_2, where cadets are restricted in S_2. Similarly, to obtain a lower bound T_j^L of T_j, we may set λ to $|S_1| + |S_2| + \cdots + |S_j|$. However, in the case of the last squad m, if $T_m^L < \max_i\{t_{i0}\}$, considering the influence of the slowest running time with 0-load, we set $T_m^L := \max_i\{t_{i0}\}$.

The proposed lower bound is 4977.82 ($T_1^L = 2218.78$, $T_2^L = 2759.04$) for the instance previously introduced in Section 2.2. Additional results for two sizes are shown in Table 1 and Table 2. As for the notation for instance size, 16;(8,8) denotes an instance of $n = 16$, $m = 2$, and $|S_1| = |S_2| = 8$. When all squad sizes are equal, we also use 16;(8)2. The parameter ρ is a correlation coefficient between t_{i0} and t_{i1}, and, t_{i1} and t_{i2}, which we explain more precisely in Section 4.2. Also, computational environments are discussed in Section 4.3.

Seeing even these subtle results, we can expect to use the proposed lower bounds in the framework for a branch-and-bound method. Hence, it is necessary to consider revising the lower bound strategies for the conditions where some cadets are fixed or unfixed to some squads.

3.2 Fixing/unfixing cadets to squads

The proposed lower bound is chosen so as to relax the restriction of having a limited number of cadets carry a given load by having all cadets

be able to assist in carrying that load. That is, each cadet is allowed to be involved in several squads and to carry partial loads between these squads. Hence, we reconsider the proposed relaxation from the viewpoint of load quantity for a given cadet i for each squad involved.

In Figure 2, let $T_j^L (j = 1, 2, 3)$ be the lower bounds obtained from the proposed procedure in Section 3.1. Observing cadet i, who has a piecewise linear load-time function like that in Figure 2, we can see that the cadet contributes 0.5-load to squad 1, 1.1-load to squad 2, and 0.4-load to squad 3. More precisely, the contribution to the net load of squad 2 by cadet i at time t $(T_1^L < t \leq T_2^L)$ can be evaluated as $\ell_i(t) - \ell_i(T_1^L)$. Similarly in the following squads, the contribution to the net load of squad j by cadet i at time t is evaluated by subtracting the cumulative load that has been carried in all previous squads $1, 2, \ldots, j-1$. Defining $T_0^L := 0$, we can generally express the net load quantity for each squad j by a cadet i at time t as

$$\ell_i(t) - \ell_i(T_{j-1}^L), \quad T_{j-1}^L < t \leq T_j^L. \tag{22}$$

Next, we consider the case where some cadets are fixed and some are unfixed to particular squads. Figure 3 is an example of this; cadet

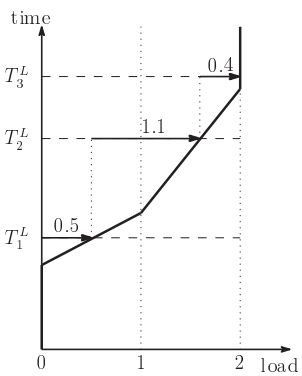

Fig. 2 Net loads for squads involved

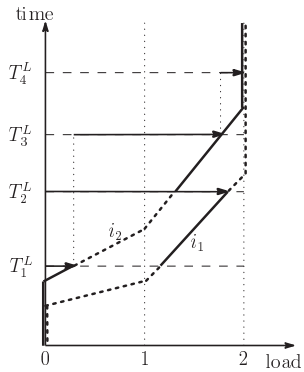

Fig. 3 Net loads when fixing/unfixing some cadets to squads

i_1 is fixed to squad 2 and cadet i_2 is unfixed to squad 2. Obviously, fixing cadet i_1 to squad 2 results in unfixing cadet i_1 to the other squads.

Since cadet i_1 is considered only in the calculation of the net load at t for squad 2, the contribution of cadet i_1 to squad 2 is fully evaluated as $\ell_{i_1}(t)(T_1^L < t \leq T_2^L)$ without subtracting the cumulative load from previous squads. On the other hand, since cadet i_2 does not contribute to squad 2 but to squad 1, the net load for squad 3 should be $\ell_{i_2}(t) - \ell_{i_2}(T_1^L)(T_2^L < t \leq T_3^L)$. Additionally, the net load for squad 4 should be $\ell_{i_2}(t) - \ell_{i_2}(T_3^L)(T_3^L < t \leq T_4^L)$. Therefore, whether cadet i is fixed or unfixed to squad j, the contribution to the net load of squad j can be evaluated by subtracting the cumulative load in previous squads.

Lastly, consider the case where cadet i is fixed to squad j_1 but does not contribute any load to it. Such a case occurs when $t_{i0} > T_{j_1}^L$. Then the relaxed squad time is set to $T_{j_1}^L := t_{i0}$. As a result, if $|S_{j_1}| = |S_{j_2}|(j_1 < j_2)$ and the relaxed squad time $T_{j_2}^L$ becomes smaller than $T_{j_1}^L$, then we can consider the branch-and-bound search tree to be wasteful fixing because of the assumption $T_{j_1}^L \leq T_{j_2}^L(|S_{j_1}| = |S_{j2}|, j_1 < j_2)$ from (6) or (16).

4. Branch-and-Bound Algorithm

4.1 Upper bounds and the branching rule

As an initial feasible solution, we sort cadets along the 1-load running time $t_{i1}(i \in C)$, assign the first $|S_1|$ cadets to squad 1, and sequentially assign the following $|S_j|$ cadets to squad j ($j = 2, \ldots, m$).

Another important matter in developing a branch-and-bound algorithm is the branching rule. Our branching rule depends on the rule normally used in the multiple knapsack problem.[5] Here, cadet i is assigned sequentially to squad 1, squad 2, ..., squad m, and as a result the search tree forms an m-nary tree. The cadets are numbered along the 0-load running time, so better solutions are expected to appear at a relatively early stage. Afterwards, if cadets with a fast running time are assigned to squad 2 or latter squads, then the lower bound of squad 1, that is, T_1^L, is likely to increase, so it can be expected that the search tree will be cut. On the other hand, if cadets with a slow running time are assigned to former squads, then their running times influence the squad times for the squads and may break the constraints $T_{j_1}^L \leq T_{j_2}^L (|S_{j_1}| = |S_{j_2}|, j_1 < j_2)$. For this too we may also be expected to cut the search tree.

4.2 Generating instances

This study was inspired by Dan-Kou, a competitive sport held at our academy. All of the data we have are only a single set that consists of 101 cadets and is lacking in many ways, particularly in the data for the 2-load running times. In order to evaluate the proposed algorithm, it is necessary to generate trial data based on variables such as estimated mean and variance. As an additional consideration, about 10 % of cadets are women.

First, we assume that each t_{i0}, t_{i1}, and t_{i2} follows a normal distribution $N(\mu, \sigma^2)$ among men and women, where μ is the mean and σ^2 is the variance. We give the distributions of the running times in Table 3, estimated from the real data set.

Secondly, we consider introducing correlation. It may be natural to assume that a person who is able to run fast can also run fast even when the person is carrying a load. On the other hand, a fast but lightweight vehicle often worsens its performance when it loads heavy weight while a slow but heavy vehicle is not affected by it. Accordingly, in our computational experiences, we also introduce some negative correlation when evaluating our algorithm for a mathematical model, namely, MMGLD.

It is known that two normal distributions X and Y along $N(0.0, 1.0^2)$ having the correlation coefficient ρ can be generated by using another normal distribution Z along $N(0.0, 1.0^2)$ using

$$Y = \rho X + \sqrt{1 - \rho^2} Z. \tag{23}$$

Hence, by using the normal distributions X, Z_0, and Z_2 along $N(0.0, 1.0^2)$, for the case of men, we set t_{i1} as $200X + 2300$, t_{i0} as $200Y_0 + 2200$, and t_{i2} as $300Y_2 + 2500$ and make them satisfy $t_{i0} < t_{i1} < t_{i2}$.

4.3 Computational experiments

The computational experiments were performed on a personal computer with a 3.3 GHz Intel Core i7-5820K CPU running the CentOS 6.8 operating system. The primary purpose of the experiments in this section is to gauge the ability of the branch-and-bound algorithm. As stated by Hamamoto and Kozuka the number of

Table 3 Distributions of running times for load and gender

	Men(90%)	Women(10%)
0-load: t_{i0}	$N(2200, 200^2)$	$N(2500, 200^2)$
1-load: t_{i1}	$N(2300, 200^2)$	$N(2700, 200^2)$
2-load: t_{i2}	$N(2500, 300^2)$	$N(3100, 300^2)$

Table 4 Results for the proposed branch-and-bound algorithm

instance type	correlation coefficient ρ					
$n;(\frac{n}{m})m$	-0.6	-0.3	0.0	0.3	0.6	0.9
	3.3354	2.0603	1.7654	1.6411	1.4300	1.1420
16;(8)2	0.0050	0.0035	0.0022	0.0013	0.0007	0.0007
	(167)	(59)	(802)	(1262)	(2043)	(1631)
	22.5240	20.2723	16.7761	14.6891	12.0873	8.9068
20;(10)2	0.0456	0.0242	0.0125	0.0052	0.0026	0.0017
	(494)	(838)	(1342)	(2825)	(4649)	(5239)
	185.3960	152.9805	134.1564	107.9700	77.2425	42.0382
24;(12)2	0.2077	0.1069	0.0338	0.0208	0.0066	0.0086
	(893)	(1431)	(3969)	(5191)	(11703)	(4888)
	[11] *2686.9762*	[11] *2737.3445*	[2] *1252.7871*	[6] *1481.9765*	[4] *960.6046*	[1] *388.9298*
28;(14)2	*1.4065*	*0.4464*	*0.1692*	*0.0306*	*0.0146*	*0.0390*
	(1019)	(6132)	(7404)	(48431)	(65795)	(9972)
	0.8994	0.9115	0.7636	0.6299	0.4788	0.3571
12;(4)3	0.0075	0.0067	0.0050	0.0039	0.0029	0.0029
	(119)	(136)	(153)	(162)	(165)	(123)
	169.3254	114.4436	57.9433	30.1712	10.4648	3.0221
18;(6)3	1.0183	0.6161	0.2813	0.1629	0.0873	0.0911
	(166)	(186)	(206)	(185)	(120)	(33)
	[6] *1427.2486*	[7] *1389.5969*	[4] *1069.0050*	[1.9] 752.0816	[0.4] 250.1126	36.7846
24;(8)3	*55.6485*	*31.6295*	*8.1460*	6.0882	2.3840	2.7931
	(26)	(44)	(131)	(124)	(105)	(13)
	4.0563	4.1001	3.1175	2.5065	1.7615	1.1992
12;(3)4	0.0499	0.0397	0.0286	0.0218	0.0172	0.0189
	(81)	(103)	(109)	(115)	(102)	(63)
	212.8648	[0.1] 207.1138	153.5388	80.6396	39.1376	15.4401
16;(4)4	2.0730	1.4722	0.9161	0.5679	0.3144	0.4099
	(103)	(141)	(168)	(142)	(124)	(38)

squads seems to be the crucial parameter for the algorithm. So in Table 4, the first four types are for two-squad models, the next three types are for three-squad models, and the last two types are for four-squad models. Each squad size is set to be equal. The results for each pair of instance type and correlation coefficient consist of three values. The upper value is the CPU time for Formulation K and Gurobi 9.1,[3] the middle value is that for the proposed branch-and-bound algorithm, and the last value in parentheses is the quotient of the two CPU times. Most values in the table are the average of 1000 trials, but values in italic font represent the average of 100 trials as a result of heavy computational burden. Leading superscript values — [0.1] 207.1138 for example—show that 0.1 percent of instances exceed the time limit of 10000 sec. The calculation of the average CPU time includes these instances as 10000 sec, so the true average time will be larger than 207.1138.

As a whole, the proposed branch-and-bound algorithm solves MMGLD much faster, sometimes thousands of times faster, than using Formulation K and Gurobi. As for the instance type, a subtle increasing of the number of cadets n or the number of squads m affects the computational burden. This observation indicates the necessity of another heuristic algorithm for solving larger and more realistic Dan-Kou size instances, which

we discuss in the next section. As for the correlation coefficient, instances with higher correlation are easier to solve. This would be preferable because real Dan-Kou data are expected to have a higher correlation coefficient.

5. Large, Realistic Size Instances

5.1 Matheuristic

The proposed branch-and-bound algorithm succeeds in solving MMGLD more effectively than Formulation K and Gurobi, but in practice its solvable sizes are too small to apply the algorithm to larger, more realistic Dan-Kou size instances. Thus, we consider applying the matheuristic strategy.[1,2] The idea behind the matheuristic approach appeared around 2010; it is an expansion of a type of local search. The idea is about taking a somewhat larger neighborhood and searching for an optimal solution by using a mathematical programming solver, such as Gurobi. This idea depends on the development of high computational performance and on the development of cutting-edge mathematical programming solvers.

5.2 Selecting optimizing tools

In MMGLD, exchanging only a few cadets would not be expected to deeply improve the feasible solutions. In introducing the matheuristic algorithm, we take a few squads and within them solve MMGLD to optimality using the proposed branch-and-bound algorithm.

First, we consider the selection of optimizing tools that are to be used in our matheuristic algorithm. Keeping in mind actual Dan-Kou conditions, we try to solve for 96 cadets and 12 squads, that is, a total of 96;(8)12 instances. In order to decide which optimizing tools or neighborhood to use, we observe the performances for solving MMGLD with a multiple of 8 cadets in a squad. For the correlation coefficient, the value for the actual Dan-Kou scenario is assumed to be 0.85–0.92, so we set $\rho := 0.9$. Table 5 shows the computational results for these instances given by the proposed branch-and-bound algorithm. Values are the average CPU time from 1000 trials.

Table 5 Results for a multiple of 8 cadets in squad instances

16;(8)2	32;(16)2	48;(24)2	64;(32)2	24;(8)3
0.0007	0.0685	9.8886	[5] *1057.7342*	2.7931

Limiting the average CPU time to 1.0 sec or less, we can take up 16;(8)2 or 32;(16)2 instances. But the performance of 32;(16)2 is about 100 times slower than 16;(8)2. Hence, for the matheuristic approach, we decide to repeatedly use only 16;(8)2.

5.3 Strategy and resuls

The proposed matheuristic algorithm involves choosing two squads randomly from the current feasible solution and solving the induced 16;(8)2 MMGLD to optimality without considering the other 10 squads. It would be possible to iterate the above procedure 120–150 times per second. The algorithm is similar to what we call '2-opt'.

Figure 4 shows the improving process for an instance. The bold line shows the result of Formulation K and Gurobi, which gradually improves incumbent solutions over 3600 sec. On the other hand, the ten horizontal lines, the differences of which depend on random seeds, show the results of the proposed matheuristic algorithm. Note that, strictly speaking, these are not simply horizontal lines. The proposed matheuristic algorithm attains these stable values within the first few seconds. Figure 5 shows an expansion of the first 10 sec of Figure 4. The bold horizontal line is the final value of Formulation K and Gurobi after a 3600-sec runtime. As shown Figure 4 and Figure 5, we find that some results (dotted lines) lie in local optimal solutions that are larger than

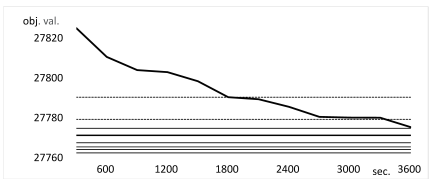

Fig. 4 Improving processes from 0 to 3600 sec for an instance

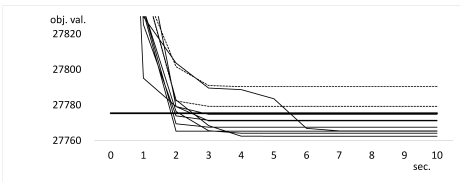

Fig. 5 Improving processes from 0 to 10 sec for an instance

the final result from Formulation K and Gurobi, but it takes 1800 or 3300 sec for Gurobi to catch up with these two inferior results.

Finally, Figure 6 shows the average behavior of 10 trials of 96;(8)12 instances. The horizontal line shows the average value obtained after a 3600-sec runtime of Gurobi and is set to 1.00 as a standard. According to the standard, the results of the matheuristic algorithm are shown as relative values. The upper line is the average of the 10 worst cases among the 10 different random seeds, and the lower line is that of the 10 best cases. The middle line is the average of all 10 instances and all 10 random seeds. The proposed matheuristic algorithm performs so quickly that the multi-start strategy will be expected to resolve large MMGLD instances, such as for 96 cadets and 12 squads, or actual Dan-Kou races.

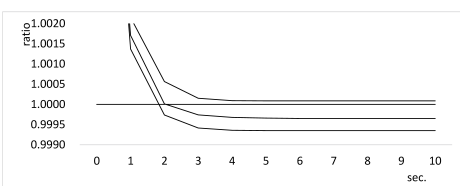

Fig. 6 Average behavior of 10 trials

6. Concluding remarks

This study was inspired by the competitive sport Dan-Kou. Based on two cadets' studies, we proposed a new lower bounding strategy. By applying the proposed lower bounds, the developed branch-and-bound algorithm succeeds in solving small mathematical models thousands of times faster than using Gurobi. In order to solve larger or real-life Dan-Kou size instances, we apply the matheuristic idea. The proposed matheuristic outperforms previous best values by a few seconds.

References

1) C. Archetti and M.G. Speranza, "A Survey on matheuristics for routing problems," *EURO Journal on Computational Optimization* **2** (2014), pp.223–246.
2) M.O. Ball, "Heuristics based on mathematical programming," *Surveys in Operations Research and Management Science* **16** (2011), pp.21–38.
3) Gurobi Optimizer: `http://www.gurobi.com` (2022-8-22).
4) Y. Hamamoto, *Optimal Squad Grouping for Dan-Kou*, Bachelor's thesis (50th term), Dept. Computer Science (2006) (in Japanese).
5) H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack problems*, Springer, Berlin (2004).
6) H. Kozuka, *Min-Max Grouping and Load-Distributing Problem*, Bachelor's thesis (65 term), Dept. Computer Science (2021) (in Japanese).
7) M. Mutingi and C. Mbohwa, *Grouping Genetic Algorithms — Advances and Applications*, Springer, Cham (2017).
8) NTT DATA Mathematical Systems Inc: `http://www.msi.co.jp/nuopt`(2022-8-22).