Yokohama Publishers
ISSN 1349-8169 ONLINE JOURNAL

Pacific Journal of Optimization
Vol. 1, No. 3, pp. 465–490, September 2005

HEURISTIC APPROACHES TO THE CAPACITATED SQUARE COVERING PROBLEM

Endre Boros, Toshihide Ibaraki, Hiroya Ichikawa,
Koji Nonobe, Takeaki Uno and Mutsunori Yagiura

Dedicated to Professor Toshihide Ibaraki on the Occasion of His 65th Birthday.

Abstract: We consider the capacitated square covering problem (CSCP), which is described as follows. Given a set of points, each having a demand, in the two-dimensional Euclidean plane, we find the minimum number of squares with the same size and capacity to cover all the points under the capacity constraint. As CSCP is NP-hard, we focus on heuristic algorithms in this paper. We first test a set covering approach, in which a CSCP is solved as a set covering problem. Since its performance is not always satisfactory, though it works quite well for a certain type of instances, we then propose a more robust metaheuristic approach. In this approach, starting with a feasible solution using a rather large number of squares, we remove squares one by one. After each removal of a square, the resulting infeasible solution is repaired by a local search method so that it becomes feasible. To increase the performance, we incorporate the idea of tabu search and scatter search as well as an adaptive control mechanism of penalty weights. Computational results on randomly generated instances with up to 1600 points indicate the effectiveness of our approaches.

Key words: *capacitated square covering problem, geometric covering problem, set covering, metaheuristics, approximation algorithm*

Mathematics Subject Classification: *90C59, 52C15*

1 Introduction

Given a set of points in the two-dimensional Euclidean plane, *the square covering problem* asks to find the minimum number of squares of the given size that cover all the points, where each square is placed in the plane with its sides parallel to axes. The problem has applications in such fields as VLSI design, location of emergency facilities, image processing and others, and theoretical results have been achieved in literature. In [4] and [12], the problem of asking to cover points in the plane by a given number of squares or disks of prescribed size is proved to be NP-hard. In [7], a polynomial time approximation scheme (PTAS), which uses a unified technique called the shifting strategy, is presented. A similar technique was independently proposed in [3], and has often been used to develop PTASs for covering and packing problems in the plane and geometric location problems [8].

In this paper, we consider the *capacitated square covering problem* (CSCP), in which each point has its demand and squares have a common capacity, and focus on developing heuristic algorithms, mainly from a practical viewpoint. We first test a set covering approach, which is based on the transformation of CSCP into the set covering problem (SCP) by enumerating all sets of points that can be covered by a square, as will be described in Section 3. This approach solves the resulting SCP instance by an existing SCP algorithm.

Since the set covering approach becomes inapplicable when the number of sets to enumerate explodes, we propose, as a more robust one, a metaheuristic approach in Section 4. This is a two-phase heuristic algorithm; we construct a feasible solution in the first phase (construction phase), and reduce the number of squares step by step in the second phase (improvement phase). For the first phase, we present a constructive method and show that it has an approximation ratio of four. (By using a PTAS developed for the (uncapacitated) square covering problem, we can improve the approximation ratio to $3 + \epsilon$ for any $\epsilon > 0$.) In the improvement phase, we first remove one square from the best feasible solution obtained so far. Since the removal makes the solution infeasible in general, we repair it by local search, without changing the number of squares. In our local search, the quality of solutions are measured by the weighted sum of penalties, where the penalty represents how much constraints are violated. If the locally optimal solution obtained is still infeasible, we return to the best feasible solution, remove another square instead, and apply a local search again; this process may be considered as a multi-start local search (MLS). If a feasible solution is found by the MLS, we repeat the removal of one more square and its repair by MLS to find a better feasible solution.

In Section 5, we try to improve our metaheuristic approach. For this end, we extend the local search to *tabu search* [5] and use it in the framework of *scatter search* [6]. Furthermore, we introduce an adaptive control mechanism of the penalty weights. To compare effectiveness of our approaches, we solve randomly generated instances with up to 1600 points. The experimental results are reported in Section 6.

$\boxed{2}$ Problem Definition

Let $P = \{1, 2, \ldots, n\}$ stand for a given set of points (x_i, y_i) $(i \in P)$ in the two-dimensional Euclidean plane, each having a demand $d_i \geq 0$. The squares to cover these points have the same side length $l > 0$ and capacity $c > 0$. The *capacitated square covering problem* (CSCP) seeks to find the minimum number of squares that cover all the points under the capacity constraint that the total demand of the points covered by a square does not exceed c, where each square must be placed in the plane with its sides parallel to x- or y-axis. Throughout this paper, we consider the x-axis as a horizontal line from left to right, and the y-axis as a vertical line from bottom to top. To ensure feasibility of the problem, we assume $d_i \leq c$ for all $i \in P$.

CSCP can be formulated as a problem of finding a collection $\mathcal{S} = \{S_1, S_2, \ldots, S_{|\mathcal{S}|}\}$ of disjoint subsets of P such that its union is P (i.e., $\mathcal{S}$ is a partition of P) and each $S \in \mathcal{S}$ $(S \subseteq P)$ satisfies the following inequalities (stating that the points in S can be covered by a single square and satisfy the capacity constraints):

$$\max\{|x_{i_1} - x_{i_2}|, |y_{i_1} - y_{i_2}|\} \leq l, \quad \forall i_1, i_2 \in S, \tag{1}$$

$$\sum_{i \in S} d_i \leq c. \tag{2}$$

The objective is to minimize the number of subsets in $\mathcal{S}$. In this formulation, each square is identified with a subset $S \in \mathcal{S}$, and square S is said to *cover* point i if $i \in S$ holds. To cover point i by a square S, the point i must be contained in the region of square S; in this situation, we say that S *geometrically covers* point i without referring to constraint

(2). In other words, the set of points covered by a square S are chosen from those points geometrically covered by S so that the capacity constraint (2) is satisfied.

Although the position of square S is not explicitly given in the formulation, it can be determined from constraint (1) by putting its bottom-left corner at $(x^{\min}, y^{\min})$, where $x^{\min} = \min\{x_i \mid i \in S\}$ and $y^{\min} = \min\{y_i \mid i \in S\}$.

A subset $S \subseteq P$ is called *valid* if it satisfies constraints (1) and (2). We call a partition $\mathcal{S} = \{S_1, S_2, \ldots, S_{|\mathcal{S}|}\}$ of P a *solution*, and call it *feasible* if all $S \in \mathcal{S}$ are valid.

$\boxed{3}$ Set Covering Approach

For a given CSCP instance, let $\mathcal{F}$ denote the collection of all valid subsets $S \subseteq P$. By introducing a 0-1 variable $z(S)$ for each $S \in \mathcal{F}$ such that $z(S)$ represents whether S is contained in the solution $\mathcal{S}$ or not, we can formulate CSCP as follows:

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{S \in \mathcal{F}} z(S) & \\ \text{subject to} & \displaystyle\sum_{S:\ i \in S} z(S) = 1, & i \in P, \\ & z(S) \in \{0,1\}, & S \in \mathcal{F}. \end{array}$$

Since, for any $S \in \mathcal{F}$, any of its subset $S' \subset S$ is also valid, we can replace the equality constraints by inequalities

$$\sum_{S:\ i \in S} z(S) \geq 1, \qquad i \in P$$

without changing the optimal value. For the same reason, $\mathcal{F}$ may be defined as the collection of all valid subsets maximal with respect to set-inclusion. The resulting problem is in the form of the *set covering problem* (SCP), and we can solve it by using an existing SCP algorithm. We call this the *set covering approach.*

In order to enumerate all maximal valid subsets efficiently in $\mathcal{F}$, we can use a polynomial delay algorithm given in Appendix A. Given an undirected graph (V, E) and a weight $w(i)$ for each $i \in V$ as well as a constant α, this algorithm enumerates all maximal cliques K in (V, E) such that the total weight of the vertices in K does not exceed α. For a given CSCP instance, we define graph (V, E) by letting $V = P$ and $E = \{(i_1, i_2) \in P \times P \mid \max\{|x_{i_1} - x_{i_2}|, |y_{i_1} - y_{i_2}|\} \leq l\}$. Then, a subset $S \subseteq V$ forms a clique in (V, E) if and only if it satisfies the geometric constraint (1). Therefore, by setting $w(i) = d_i$ $(i \in V)$ and $\alpha = c$, we can use this algorithm for our purpose.

For the uncapacitated case (i.e., the capacity c is infinite), the enumeration can be achieved more easily. Let $S \in \mathcal{F}$ be a maximal valid subset, and let $i_L = \arg\min\{x_i \mid i \in S\}$ and $i_B = \arg\min\{y_i \mid i \in S\}$. Then, S is equal to a set of points $i \in P$ such that $(x_i, y_i) \in [x_{i_L}, x_{i_L} + l] \times [y_{i_B}, y_{i_B} + l]$. This implies that we can enumerate all maximal valid subsets by considering, as such i_L and i_B, all pair of (possibly the same) points i_1 and i_2 in P such that $x_{i_1} \leq x_{i_2} \leq x_{i_1} + l$ and $y_{i_2} \leq y_{i_1} \leq y_{i_2} + l$. Let $\tilde{\mathcal{F}}$ be the collection of subsets enumerated in this way. Although $\tilde{\mathcal{F}}$ may contain subsets that are not maximal, their validity is guaranteed. Furthermore, the size of $\tilde{\mathcal{F}}$ is at most $\nu|P|$, where ν is the largest size of $S \in \tilde{\mathcal{F}}$. Since in many cases, ν is much smaller than $|P|$, we can practically use $\tilde{\mathcal{F}}$ in the set covering approach. This simple enumeration method runs in $\mathrm{O}(|P| \log |P| + \sum_{S \in \tilde{\mathcal{F}}} |S|)$ time, if we use appropriate data structures such as heap and balanced binary search tree. Since the output size is $\Omega(\sum_{S \in \tilde{\mathcal{F}}} |S|)$, only $\mathrm{O}(|P| \log |P|)$ time is additional.

As will be reported in Section 6, the performance of the set covering approach crucially depends on the number of subsets in $\mathcal{F}$. If points are densely located in the plane (which means that many points can be geometrically covered simultaneously by a square) and the capacity constraint is not very tight, the set covering approach becomes practically infeasible due to a huge number of maximal valid subsets. To overcome this difficulty, we next propose a metaheuristic approach.

4 Metaheuristic Approach: Basic Ideas

Our metaheuristic approach consists of two phases; we first construct a feasible solution (construction phase), and decrease the number of squares one by one (improvement phase). Its framework is described below.

CSCP algorithm

Step 0 (Construction).
Construct an initial feasible solution $\mathcal{S}^{(0)}$. Set $k := 1$.

Step k (Improvement).
Apply a metaheuristic algorithm to find a feasible solution $\mathcal{S}^{(k)}$ with $|\mathcal{S}^{(k-1)}| - 1$ squares. If we fail to find such a solution, then output the current best solution $\mathcal{S}^{(k-1)}$ and terminate; otherwise let $k := k + 1$, and return to Step k.

In this section, after explaining a method to construct an initial feasible solution and analyzing its approximation ratio, we will describe basic components of our metaheuristic algorithms. Other ingredients to improve the performance will be presented in the next section.

4.1 Construction Phase

4.1.1 Algorithm

Our construction algorithm starts with no square, and iteratively puts a new square S_j in the plane until all points are covered. After putting a new square S_j, we insert as many points as possible into S_j under constraints (1) and (2), and then adjust its position in the hope that it covers some more points.

The algorithm is summarized below. During the computation, set U stores the points not covered by any square yet.

CONSTRUCTION
Input: An instance of CSCP.
Output: A feasible solution $\mathcal{S}^{(0)}$.

Step 1 (Initialization)
Initialize a collection of subsets $\mathcal{S}^{(0)}$ by $\mathcal{S}^{(0)} := \emptyset$ and $U := P$. Set $j := 0$.

Step 2 (Determination of target points)
If $U = \emptyset$, then output $\mathcal{S}^{(0)}$ and terminate; $\mathcal{S}^{(0)}$ is a feasible solution. Otherwise, let i_L be the leftmost point in U. (If there is more than one such point, take the bottom one.) Let A be the set of all uncovered points i ($\in U$) such that i is within the closed rectangle $[x_{i_L}, x_{i_L} + l] \times [y_{i_L} - l, y_{i_L} + l]$ defined by i_L (see Figure 1 (1)(7), where white points denote uncovered points).

The points in A are called *target points*. In the following, Steps 3–5 will be repeated until every target point is covered by some square.

Step 3 (Covering by a new square)
Let i_B be the uncovered target point with the smallest y-coordinate. (If there is more than one such point, take the leftmost one.) Let $j := j + 1$ and introduce a new square S_j (which corresponds to adding an empty set S_j to $\mathcal{S}^{(0)}$), and place it in the plane so that its bottom-left corner is at (x_{i_L}, y_{i_B}) (Figure 1 (2)(3)(5)(8)). Consider those uncovered points i ($\in U$) now geometrically covered by square S_j; i.e., $(x_i, y_i) \in [x_{i_L}, x_{i_L} + l] \times [y_{i_B}, y_{i_B} + l]$, and add into S_j as many such points as possible under the capacity constraint (2) in non-decreasing order of y_i, breaking ties by preferring smaller x_i (black points in Figure 1 (2)(3)(5), where gray points denote those which have already been covered). Delete all points in S_j from set U.

Step 4 (Covering some more points)
Let $x^{\min} := \min\{x_i \,|\, i \in S_j\}$ and $y^{\max} := \max\{y_i \,|\, i \in S_j\}$. If $x^{\min} = x_{i_L}$ holds, go to Step 5 (Figure 1 (2)). Otherwise, change the position of square S_j so that its top-left corner is at $(x^{\min}, y^{\max})$ (Figure 1 (3)$\rightarrow$(4), (5)$\rightarrow$(6)). (This operation slides square S_j to the rightmost position, and then to the bottommost position, while keeping all points $i \in S_j$ geometrically covered by S_j.) Consider those points $i \in U$ which got geometrically covered by S_j as a result of the operation, and add as many such points as possible to S_j in the non-decreasing order of x_i. Delete from set U all points added to S_j.

Step 5 (Repetition of the process)
If $A \cap U = \emptyset$, return to Step 2; otherwise return to Step 3.

Figure 1 illustrates the process of CONSTRUCTION. In this example, the capacity c of a square is four, and the demand d_i of each point i is one or two as shown in the figure. White points represent those which have not been covered yet, while gray ones have already been covered, and black ones are being covered by the current square drawn with thick lines. At first, we apply Steps 1 and 2, and we get (1), where points inside the two adjacent squares with broken lines are target points A. (2) shows the result after applying Step 3. One square is introduced and it covers three points, while one point with a demand of two is left uncovered even though it is geometrically covered, because of the capacity constraint. Since we cannot slide the square in Step 4, we go to Step 5 and return to Step 3. (3) illustrates the result after one more square is introduced in the next iteration of Step 3, and (4) shows the result of Step 4. Since there still remains an uncovered target point, we return to Step 3, and we get (5), where (6) shows the result of Step 4. Since all target points are now covered, we return to Step 2, and determine the next target points as illustrated by two broken squares in (7). In (8), all points in P are covered, and CONSTRUCTION terminates. In this example, four squares in (2), (4), (6) and (8) constitute the output solution.

$\boxed{4.1.2}$ Approximation Ratio

We analyze algorithm CONSTRUCTION and show that it has an approximation ratio of four. We first give two lower bounds on the optimal value m^*. One is trivially derived from the capacity constraints:

$$m^* \geq \frac{\sum_{i \in P} d_i}{c}.$$

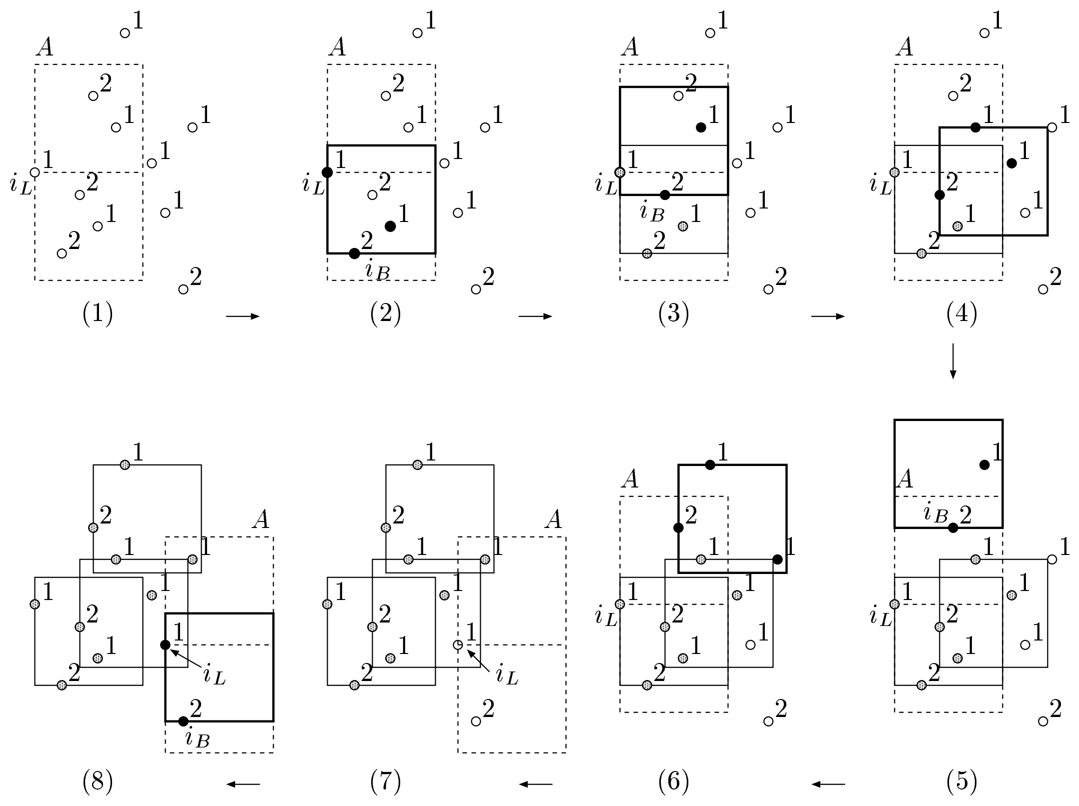

Figure 1: An illustrative example of CONSTRUCTION

The other is based on the geometric property. Let P_L be the set of all points chosen as i_L in some iteration of Step 2. For convenience, we consider the process of CONSTRUCTION as a series of $|P_L|$ rounds, each starting when i_L is selected and target points A are determined in Step 2.

We claim that no two different points i_1 and i_2 in P_L can be covered simultaneously by a square. Without loss of generality, assume that i_1 is selected as i_L in an earlier round than i_2. Then $x_{i_1} \leq x_{i_2}$ holds. Furthermore, i_2 is a target point in the round of i_1, implying $x_{i_2} - x_{i_1} > l$ or $|y_{i_1} - y_{i_2}| > l$; hence the claim is true. Therefore, at least $|P_L|$ squares are necessary to cover all points in $P_L \subseteq P$, and we have

$$m^* \geq |P_L|.$$

We note that squares S_j are introduced only in Step 3. We classify those squares S_j into two groups. If some target point i that is geometrically covered by S_j remains uncovered after the Step 3, then S_j is called *saturated* (because the point i is excluded due to the capacity constraint); otherwise called *unsaturated.*

Let $\mathcal{S}_{\text{sat}} \subset \mathcal{S}^{(0)}$ be the collection of saturated squares. For a square $S_j \in \mathcal{S}_{\text{sat}}$, consider the time when Step 3 is completed after S_j is introduced. Let i_j be the target point with the smallest y-coordinate among those that are left uncovered even though geometrically covered by S_j (if there is more than one such point, take the leftmost one). Obviously, the residual capacity $c - \sum_{i \in S_j} d_i$ of square S_j is less than d_{i_j}, and thus

$$\sum_{S_j \in \mathcal{S}_{\mathrm{sat}}} \left(c - \sum_{i \in S_j} d_i \right) < \sum_{S_j \in \mathcal{S}_{\mathrm{sat}}} d_{i_j}.$$

Since i_j will be selected as i_B at the next iteration of Step 3, and covered by the next square S_{j+1}, we have $i_j \neq i_{j'}$ for any two distinct saturated squares S_j and $S_{j'}$ $(j \neq j')$. Therefore, we obtain

$$|\mathcal{S}_{\mathrm{sat}}| c < \sum_{S_j \in \mathcal{S}_{\mathrm{sat}}} d_{i_j} + \sum_{S_j \in \mathcal{S}_{\mathrm{sat}}} \sum_{i \in S_j} d_i < 2 \sum_{i \in P} d_i,$$

and

$$|\mathcal{S}_{\mathrm{sat}}| < 2 \frac{\sum_{i \in P} d_i}{c} \leq 2m^*. \tag{3}$$

(The idea of this analysis is essentially the same as that of the Next-Fit bin packing algorithm[9].)

Next, we analyze the number of unsaturated squares. Once Step 3 ends with an unsaturated square S_j, no target point located below the upper side of S_j is left uncovered, and hence, at the subsequent iterations in the same round, new squares will be placed in the plane completely above S_j. Since all target points for the current round can be geometrically covered by two squares, the number of unsaturated squares is at most two per round, and thus at most $2|P_L|$ squares in total;

$$|\mathcal{S}| - |\mathcal{S}_{\mathrm{sat}}| \leq 2|P_L| \leq 2m^*. \tag{4}$$

Putting the results (3) and (4) together, we obtain the following theorem.

Theorem 1 *CONSTRUCTION has an approximation ratio of four. For the uncapacitated case, all the squares are unsaturated, and the approximation ratio becomes two.* $\square$

We note here that Step 4 has no effect to the analysis of approximation ratio, but is practically useful in reducing the number of squares.

As mentioned in Introduction, it is known that the uncapacitated square covering problem has a PTAS. By modifying CONSTRUCTION to utilize a $(1+\epsilon)$-approximation algorithm for the uncapacitated version, we can achieve $(3+\epsilon)$ approximation ratio for CSCP. For a given CSCP instance, ignoring the capacity constraints, we first obtain a $(1+\epsilon)$-approximation solution $\tilde{\mathcal{S}} = \{\tilde{S}_1, \tilde{S}_2, \ldots, \tilde{S}_{\tilde{m}}\}$ to the resultant uncapacitated problem instance. Then, we apply CONSTRUCTION while setting target points A to $\tilde{S}_\kappa$ at the κ-th iteration of Step 2. (The algorithm then terminates in $\tilde{m}$ rounds.) By the same argument above, the number of saturated squares is shown to be at most $2m^*$, where m^* is the optimal value. As for unsaturated squares, their number is at most $\tilde{m}$, which is the number of rounds, because whenever Step 3 ends with an unsaturated square, no uncovered target point is left and the current round is completed. Since $\tilde{m} \leq (1+\epsilon)m^*$ holds by assumption, the total number of squares is bounded by $2m^* + \tilde{m} \leq (3+\epsilon)m^*$.

Although this modification improves the approximation ratio, it makes CONSTRUCTION slower. Therefore, we adopt the original one in our implementation.

$\boxed{4.2}$ Improvement Phase

The objective of step k in the improvement phase is to find a feasible solution with $|\mathcal{S}^{(k-1)}|-1$ squares, where $\mathcal{S}^{(k-1)}$ is the best solution obtained so far. To achieve this, we employ metaheuristic algorithms, whose core part is local search. Local search is a method that iteratively replaces the current solution $\mathcal{S}$ by another one $\mathcal{S}'$ that can be obtained from $\mathcal{S}$ by a slight modification. Such a solution $\mathcal{S}'$ is called a *neighbor* of $\mathcal{S}$, and the set of neighbors is called the *neighborhood.* Local search terminates when the current solution is locally optimal (i.e., no better solution exists in its neighborhood).

In this subsection, we describe a multi-start local search algorithm (MLS), which repeatedly executes local search from different initial solutions. In the rest of this section, we explain the components of our local search and how to generate initial solutions.

$\boxed{4.2.1}$ Search Space and Penalty Function

Suppose that the current best solution uses $m+1$ squares. MLS operates on the set of all (feasible and infeasible) solutions $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ with m squares, which implies that the constraints (1) and (2) are not necessarily satisfied during the search. For convenience, we sometimes represent a solution $\mathcal{S}$ by a mapping $\sigma_{\mathcal{S}} : P \to \{1, 2, \ldots, m\}$ such that $\sigma_{\mathcal{S}}(i) = j \iff i \in S_j$, and we denote $\sigma_{\mathcal{S}}$ simply by σ, when $\mathcal{S}$ is clear from context.

To evaluate solutions, we introduce penalty function $p(\mathcal{S})$ to be minimized, which takes value 0 if $\mathcal{S}$ is feasible, and a positive value otherwise. More precisely, for each square $S_j \in \mathcal{S}$, we define penalty functions $p^{\text{geo}}(S_j)$ and $p^{\text{cap}}(S_j)$, which represent how much S_j violates constraints (1) and (2), respectively, by

$$\begin{aligned} p^{\text{geo}}(S_j) &= \max\{0, \Delta_x(S_j) - l\} + \max\{0, \Delta_y(S_j) - l\}, \\ p^{\text{cap}}(S_j) &= \max\left\{0, \sum_{i \in S_j} d_j - c\right\}, \end{aligned}$$

where

$$\Delta_x(S_j) = \max_{i_1, i_2 \in S_j} |x_{i_1} - x_{i_2}| \quad \text{and} \quad \Delta_y(S_j) = \max_{i_1, i_2 \in S_j} |y_{i_1} - y_{i_2}|.$$

(If S_j contains less than two points, $\Delta_x(S_j)$ and $\Delta_y(S_j)$ are defined as 0.) Then $p(\mathcal{S})$ is given by the weighted sum of these penalties

$$p(\mathcal{S}) = \sum_{S_j \in \mathcal{S}} \left(w_j^{\text{geo}} p^{\text{geo}}(S_j) + w_j^{\text{cap}} p^{\text{cap}}(S_j) \right), \tag{5}$$

where weights w_j^{geo} and w_j^{cap} are program parameters, which are adaptively controlled during search as will be explained in the next section. The objective of MLS is to find a solution $\mathcal{S}$ such that $p(\mathcal{S}) = 0$.

Our MLS adopts the best admissible move strategy; the current solution is replaced by the best neighbor in the sense of having the minimum penalty value. If there are more than one best solution, to break ties, we use a secondary criterion $g(\mathcal{S})$ (a smaller value is better) defined by

$$g(\mathcal{S}) = \sum_{S_j \in \mathcal{S}} \left(\Delta_x(S_j) + \Delta_y(S_j) \right).$$

The values of Δ_x and Δ_y affect the penalty value $p(\mathcal{S})$ only when they are larger than l. However, even if $\Delta_x \le l$ and $\Delta_y \le l$ hold, squares with smaller Δ_x and Δ_y may have higher potential for covering more points without violating the geometric constraint (1). The criterion $g(\mathcal{S})$ gives higher priority to such squares.

$\boxed{4.2.2}$ Shift and Swap Neighborhoods

To generate neighbors of the current solution $\mathcal{S}$, MLS uses two types of operations: Shift and Swap. A shift operation Shift(i, j) is defined for a pair of point i and square S_j such that $j \neq \sigma(i)$, and moves point i into S_j (i.e., $\sigma(i) := j$). A swap operation Swap(i_1, i_2) is defined for a pair of two points i_1 and i_2 with $\sigma(i_1) \neq \sigma(i_2)$ and switches them (i.e., $\sigma(i_1) \leftrightarrow \sigma(i_2)$). Let $\mathcal{N}_{\text{SHIFT}}(\mathcal{S})$ (resp., $\mathcal{N}_{\text{SWAP}}(\mathcal{S})$) denote the set of solutions obtained from $\mathcal{S}$ by applying a shift (resp., a swap) operation. The size of $\mathcal{N}_{\text{SHIFT}}(\mathcal{S})$ is $\mathrm{O}(mn)$ and that of $\mathcal{N}_{\text{SWAP}}(\mathcal{S})$ is $\mathrm{O}(n^2)$, where n and m are the numbers of points and squares, respectively. Since it is time-consuming to search the whole neighborhoods at each iteration of local search, we restrict them as follows:

- Shift(i, j) is applied to $\mathcal{S}$ only if both of the following conditions hold:
    1. (a) Square $S_{\sigma(i)}$, which currently covers i, has a positive penalty, or (b) point i has the minimum or maximum x- or y-coordinate in $S_{\sigma(i)}$ (which means that point i is on the boundary of the minimum rectangle containing all points in $S_{\sigma(i)}$).
    2. (a) Square S_j covers no point; i.e., $S_j = \emptyset$, or (b) S_j contains a point i' whose L_∞-distance from i, $\max\{|x_i - x_{i'}|, |y_i - y_{i'}|\}$, is less than or equal to max_dist_shift, where max_dist_shift is a program parameter.

- Swap(i_1, i_2) is applied to $\mathcal{S}$ only if both of the following conditions hold:
    1. Square $S_{\sigma(j_1)}$ or $S_{\sigma(j_2)}$ has a positive penalty.
    2. The L_∞-distance between i_1 and i_2 is less than or equal to max_dist_swap, which is another program parameter.

We denote by $\tilde{\mathcal{N}}_{\text{SHIFT}}(\mathcal{S})$ and $\tilde{\mathcal{N}}_{\text{SWAP}}(\mathcal{S})$, respectively, the shift and swap neighborhoods reduced in this way. These restrictions are expected to exclude shift and swap operations that are unlikely to improve the current solution $\mathcal{S}$. For example, in the case of Shift(i, j), conditions 1-(a) and 1-(b) assures the possibility to decrease the values of $p^{\text{geo}}(S_{\sigma(i)}), p^{\text{cap}}(S_{\sigma(i)})$ and $\Delta_x(S_{\sigma(i)}), \Delta_y(S_{\sigma(i)})$, respectively, while condition 2 is to keep the increment of the penalty $p^{\text{geo}}(S_j)$ below max_dist_shift.

In our current implementation, we set $max\_dist\_shift = l$ and $max\_dist\_swap = l$.

$\boxed{4.2.3}$ Initial Solutions of MLS

To launch local search, we need to prepare an initial solution. In our MLS, we generate it by removing one square S_j from the best feasible solution $\mathcal{S}^{(k-1)} = \{S_1, S_2, \ldots, S_{m+1}\}$. As the removal of S_j leaves the points in S_j uncovered, we cover them by the remaining m squares by applying the following step $|S_j|$ times in a greedy fashion; apply a shift operation Shift(i, j') with $i \in S_j$ and $j' \neq j$ such that the minimum penalty increase is attained. (In our implementation, to reduce the computational efforts, we check only those pairs i and j' satisfying the above condition 2-(b).) From the resultant solution, we then execute local search. If the local optimum obtained is infeasible, the same process is repeated by removing another square until a feasible solution is found, or $m+1$ times, whichever comes first. In the latter case, there is no square to remove any more, and we conclude that MLS has failed to improve the best solution and return to the CSCP algorithm, which immediately terminates after outputting the best solution $\mathcal{S}^{(k-1)}$.

Now we mention about the order of squares S_j to be removed from $\mathcal{S}^{(k-1)}$. To find a feasible solution (if any) in fewer runs of local search, we remove first a square $S_j \in \mathcal{S}^{(k-1)}$ that covers the smallest number of points, so that the total penalty increase in the resulting initial solution may be small.

$\boxed{4.2.4}$ Algorithm

The entire algorithm of MLS is summarized below.

Multi-Start Local Search (MLS)

Input: A feasible solution $\mathcal{S}^{(k-1)}$ with $m+1$ squares.
Output: A feasible solution $\mathcal{S}^{(k)}$ with m squares, or "failure."

Step 1 (Initial solution).
If all $m+1$ initial solutions have been tested without finding a feasible solution, return "failure." Otherwise, according to the rule described in Section 4.2.3, generate a solution $\mathcal{S}$ with m squares.

Step 2 (Local search).
If $\mathcal{S}$ is feasible, let $\mathcal{S}^{(k)} := \mathcal{S}$ and return $\mathcal{S}^{(k)}$. Otherwise, find the best solution $\mathcal{S}'$ in $\tilde{\mathcal{N}}_{\text{SHIFT}}(\mathcal{S}) \cup \tilde{\mathcal{N}}_{\text{SWAP}}(\mathcal{S})$. If $\mathcal{S}'$ is better than the current solution $\mathcal{S}$, let $\mathcal{S} := \mathcal{S}'$ and return to Step 2; otherwise return to Step 1.

$\boxed{5}$ Metaheuristic Approach: Improved Algorithms

In this section, we present some ideas for improving the performance of MLS, by incorporating tabu search and scatter search. The resulting two algorithms are denoted by MTS (Multi-start Tabu Search) and SSLS (Scatter Search with Local Search), respectively. We also tested a combination of tabu search and scatter search, denoted by SSTS (Scatter Search with Tabu Search). Finally, we propose a method of controlling the penalty weights w_j^{geo} and w_j^{cap} in penalty function (5).

$\boxed{5.1}$ Multi-Start Tabu Search

Local search terminates when a locally optimal solution is obtained. However, continuing the search beyond the local optimum appears effective for finding better solutions. To achieve this, we adopt tabu search [5]. In tabu search, the current solution $\mathcal{S}$ is always replaced with the best one $\mathcal{S}'$ in the neighborhood, even if $\mathcal{S}'$ is not better than $\mathcal{S}$. To realize an effective search while avoiding going back to solutions already visited, tabu search makes use of memory that stores the search history, which is typically embodied in *tabu lists.* In our tabu search, tabu list T stores the points $i \in P$ such that $\sigma(i)$ has recently been changed, and prohibits to change it again while i is contained in T. More precisely, T is initially set to the empty set, and updated whenever the current solution $\mathcal{S}$ is replaced by its neighbor $\mathcal{S}'$; if $\mathcal{S}'$ is obtained from $\mathcal{S}$ by $\text{SHIFT}(i,j)$, we insert point i to T, while if $\mathcal{S}'$ is obtained by $\text{SWAP}(i_1,i_2)$, we insert both points i_1 and i_2 to T. Those points in T will be maintained for *tabu_tenure* iterations, where *tabu_tenure* is a program parameter. In searching neighborhood, we call a neighbor $\mathcal{S}'$ *tabu* and exclude it from candidates, if it is obtained by applying $\text{SHIFT}(i,j)$ with $i \in T$ or $\text{SWAP}(i_1,i_2)$ with $i_1, i_2 \in T$. As in the standard implementation of tabu search, we introduce an *aspiration criterion*; the tabu status of a solution $\mathcal{S}'$ is overruled, if its penalty value is smaller than that of the best solution found in the current run of tabu search. (Recall that tabu search is repeatedly executed in MTS. The best solution is reset whenever tabu search restarts.)

Our tabu search terminates when one of the following conditions is met:

- A feasible solution is found.
- Neighborhood search has been repeated *max_iteration* times.

In the former case, the best solution to CSCP is improved and MTS terminates, while in the latter case, tabu search has failed to find a feasible solution, and restarts from another initial solution. (Initial solutions are generated in the same way as MLS.) The entire algorithm of MTS is described below.

Multi-Start Tabu Search (MTS)

Input: A feasible solution $\mathcal{S}^{(k-1)}$.

Output: A feasible solution $\mathcal{S}^{(k)}$ with one less square than $\mathcal{S}^{(k)}$ or "failure."

Step 1 (Initial solution).

As in MLS.

Step 2 (Tabu search).

If $\mathcal{S}$ is feasible, let $\mathcal{S}^{(k)} := \mathcal{S}$ and return $\mathcal{S}^{(k)}$. If the solution has been replaced *max_iteration* times, then return to Step 1. Otherwise, find the best neighbor $\mathcal{S}' \in \tilde{\mathcal{N}}_{\text{SHIFT}}(\mathcal{S}) \cup \tilde{\mathcal{N}}_{\text{SWAP}}(\mathcal{S})$ that is not tabu and let $\mathcal{S} := \mathcal{S}'$. Update the tabu list and return to Step 2.

In the computational experiments in Section 6, we set the parameters as *tabu_tenure* $= 0.2 \cdot |P|$ and *max_iteration* $= 5 \cdot |P|$ according to preliminary experiments. In the preliminary experiments, we observed that *max_iteration* had a larger impact on the results than *tabu_tenure*, and the best value of *max_iteration* depends on the maximal computation time specified by users. At the current stage, in order to find their appropriate values, preliminary experiments are necessary to a certain extent.

$\boxed{5.2}$ Scatter Search with Local Search

Scatter search is an evolutionary approach, which works on a set of solutions, called the *reference set*. (Solutions in the reference set are called *reference solutions*.) Scatter search repeats an operation of creating new solutions by combining two or more reference solutions and updating the reference set so that it contains good and diverse solutions. Scatter Search contrasts with other evolutionary procedures, such as genetic algorithms, by providing unifying principles for combining solutions based on generalized path constructions (in both Euclidean and neighborhood spaces) and by utilizing strategic designs where other approaches resort to randomization [6]. We incorporate an idea of scatter search into MLS for the purpose of generating initial solutions more effectively, and also for making it possible to execute local search more than $m + 1$ times, where $m + 1$ is the number of squares used in the current best solution. In SSLS, the reference set consists of some of local optima obtained so far. In this subsection, we first explain how to manage the reference set, and then how to use it to create initial solutions.

Reference set

In principle, we store solutions of high quality in the reference set. To achieve diversification, however, using a program parameter *min_dist_ref*, we keep the distance of any two reference solutions at least *min_dist_ref*, where the distance between $\mathcal{S}$ and $\mathcal{T}$ is defined as follows. Let π denote a one-to-one mapping $\pi : \{1, 2, \dots, m\} \to \{1, 2, \dots, m\}$. The distance is then given by $\min_\pi |\{i \in P \mid \pi(\sigma_{\mathcal{S}}(i)) \neq \sigma_{\mathcal{T}}(i)\}|$. This minimization problem is formulated as an assignment problem and solvable in $O(m(r + m \log m))$ time, where r is the number of pairs of subsets $(S, T) \in \mathcal{S} \times \mathcal{T}$ such that $S \cap T \neq \emptyset$, and r is at most $\min\{|P|, m^2\}$. For our purpose here, however, an optimal π is not necessary, but an approximate value is sufficient. Therefore, we obtain π simply by a greedy method; in the order of $j = 1, 2, \dots, m$, let $\pi(j) := j'$, where j' has the maximum value of

$$\frac{|S_j \cap T_{j'}|}{\max\{|S_j|, |T_{j'}|\}} \qquad \text{(defined as 1 if } |S_j| = |T_{j'}| = 0)$$

in $\{1, 2, \dots, m\} \setminus \{\pi(1), \pi(2), \dots, \pi(j-1)\}$.

The reference set is initially empty, and updated whenever local search terminates. Let $\mathcal{S}^*$ be the obtained locally optimal solution. If there is no reference solution whose distance from $\mathcal{S}^*$ is less than *min_dist_ref*, then we add $\mathcal{S}^*$ into the reference set, removing the worst one if the number of reference solutions exceeds *max_num_ref* as a result of this. On the other hand, suppose that some reference solution has a distance less than *min_dist_ref*. If $\mathcal{S}'$ has a penalty value smaller than any of such solutions, they are all removed from the reference set, and $\mathcal{S}'$ is added instead; otherwise, the reference set is unchanged.

Path-Relinking

In SSLS, initial solutions are generated in the same way as MLS for the first *num_initial* runs of local search, while for the subsequent runs, initial solutions are generated by a method called *path-relinking*. Path-relinking generates a sequence of solutions, called a *path*, from one solution (*initiating solution*) toward another one (*guiding solution*). (In general, it is possible to consider more than one solution as guiding solutions.) In our implementation, we first choose an initiating solution $\mathcal{S} = \{S_1, S_2, \dots, S_m\}$ and a guiding solution $\mathcal{T} = \{T_1, T_2, \dots, T_m\}$ randomly from the reference set. Then, we generate a path $(\mathcal{S}_0, \mathcal{S}_1, \dots, \mathcal{S}_q)$ by extending it from $\mathcal{S}_0 = \mathcal{S}$ to $\mathcal{S}_q = \mathcal{T}$, where $\mathcal{S}_{k+1}$ is obtained by applying a shift operation to $\mathcal{S}_k$ $(k = 0, 1, \dots, q-1)$. After the choice of $\mathcal{S}$ and $\mathcal{T}$, we find a mapping π to measure the distance from $\mathcal{S}$ to $\mathcal{T}$, by the greedy method used in the reference set management. For the current path $(\mathcal{S}_0, \mathcal{S}_1, \dots, \mathcal{S}_k)$ $(0 \leq k < q)$, let D_k be the set of all points $i \in P$ such that $\pi(\sigma_{\mathcal{S}_k}(i)) \neq \sigma_{\mathcal{T}}(i)$. To extend the path to $\mathcal{S}_{k+1}$, we apply a shift operation $\textsc{Shift}(i, j)$ with $i \in D_k$ and $j = \sigma_{\mathcal{T}}(i)$ so that $|D_{k+1}| = |D_k| - 1$ holds. (Therefore, the length of the path q is $|D_0|$.) From among such $|D_k|$ choices of $\textsc{Shift}(i, j)$, we choose and apply the best one in the sense of achieving the minimum penalty.

After the path is generated, (at most) *num_best* best solutions in $\{\mathcal{S}_1, \mathcal{S}_2, \dots, \mathcal{S}_{q-1}\}$ are selected as initial solutions of local search.

SSLS terminates when a feasible solution is found or the total computation time exceeds a prespecified bound. The entire algorithm of SSLS is summarized below.

Scatter Search with Local Search (SSLS)

Input: A feasible solution $\mathcal{S}^{(k-1)}$ with $m + 1$ squares.
Output: A feasible solution $\mathcal{S}^{(k)}$ with m squares, or "failure."

Step 0 (MLS).
Execute local search *num_initial* times in the framework of MLS, updating the reference set at the end of each local search. If a feasible solution is found during this process, output it as $\mathcal{S}^{(k)}$ and terminate.

Step 1 (Path-relinking).
If the predetermined computation time is reached, terminate after returning "failure." Otherwise, choose two reference solutions $\mathcal{S}$ and $\mathcal{T}$ randomly, and generate *num_best* initial solutions by the path-relinking.

Step 2 (Local search and update of reference set).
For each solution generated in Step 1, apply local search. If a feasible solution is found, output it as $\mathcal{S}^{(k)}$ and terminate; otherwise update the reference set and return to Step 1.

For the computational experiments in Section 6, we set *num_initial* $= 15$, *max_num_ref* $= 10$, *min_dist_ref* $= 3$ and *num_best* $= 3$. Although it might be possible to improve the performance of SSLS by tuning the parameters more carefully, on the other hand, it is reported in [14] that how to manage the reference set is not very critical in the use of path-relinking.

5.3 Scatter Search with Tabu Search

We also tested scatter search with tabu search (SSTS), which differs from SSLS only in using tabu search in place of local search in Step 0 and Step 2. In the experiments, we use the same parameter setting as MTS and SSLS except for letting *max_iteration* $= 3 \cdot |P|$.

5.4 Adaptive Control of Penalty Weights

In preliminary experiments, the performance of our metaheuristic algorithms was observed to depend on penalty weights w_j^{geo} and w_j^{cap}. Since their appropriate values may vary from instance to instance and hence it is difficult to find such values in advance, we incorporate an adaptive control mechanism of the penalty weights.

At the beginning of the improvement phase, we initialize w_j^{geo} and w_j^{cap} to one for all squares S_j, and update penalty weights when a locally optimal solution is obtained (i.e., when local search terminates in MLS or SSLS, and when the current solution is replaced by a neighbor with a larger penalty value in MTS or SSTS). Let $\mathcal{S}^*$ be the local optimum at hand. For each $S_j^* \in \mathcal{S}^*$, its weights are updated by

$$w_j^{\text{geo}} := w_j^{\text{geo}} \left(1 + \rho \cdot \frac{p^{\text{geo}}(S_j^*)}{p_{\max}^{\text{geo}}}\right) \quad \text{and} \quad w_j^{\text{cap}} := w_j^{\text{cap}} \left(1 + \rho \cdot \frac{p^{\text{cap}}(S_j^*)}{p_{\max}^{\text{cap}}}\right),$$

where

$$p_{\max}^{\text{geo}} := \max_{S_j^* \in \mathcal{S}^*} p^{\text{geo}}(S_j^*) \quad \text{and} \quad p_{\max}^{\text{cap}} := \max_{S_j^* \in \mathcal{S}^*} p^{\text{cap}}(S_j^*),$$

and $\rho > 0$ is a prespecified parameter. (If $p_{\max}^{\text{geo}} = 0$ (resp., $p_{\max}^{\text{cap}} = 0$) holds, no weight w_j^{geo} (resp., w_j^{cap}) is updated.) The above rule has the effect of making it easier for those constraints violated by $\mathcal{S}^*$ to be satisfied in the subsequent search.

The penalty weights are not reset even when local search or tabu search restarts from another initial solution. In MLS, MTS and in the first *num_initial* iterations of SSLS and SSTS, we generate initial solutions $\mathcal{S}$ by removing one square from the current best solution

$\mathcal{S}^{(k-1)}$. Penalty weights are defined for each square in $\mathcal{S}^{(k-1)}$, and each $S \in \mathcal{S}$ inherits the weights in $\mathcal{S}^{(k-1)}$. After local search or tabu search terminates and the weights are updated according to the above rule, the original weights defined for $\mathcal{S}^{(k-1)}$ are replaced by the updated ones, respectively. (The weights of the square that was removed to generate $\mathcal{S}$ are unchanged.)

To prevent the penalty weights from increasing unlimitedly, after local search or tabu search terminates, we normalize weights w_j^{geo} (resp., w_j^{cap}) by dividing by the maximum weight $\max_{j'} w_{j'}^{\text{geo}}$ (resp., $\max_{j'} w_{j'}^{\text{cap}}$). As an exceptional rule, if a weight becomes smaller than w_{LB} (resp., greater than w_{UB}) during the computation, we set it to w_{LB} (resp., w_{UB}). In our computational experiments, we use $\rho = 5.0\times10^{-2}$, $w_{\text{LB}} = 1.0\times10^{-6}$ and $w_{\text{UB}} = 1.0\times10^{7}$.

$\boxed{6}$ Computational Experiments

Our algorithms were coded in C language. The computational experiments were all conducted on a personal computer with Xeon 2.8 GHz.

For experimental purpose, we generated random instances in which x- and y-coordinates of points are all integers, chosen from $[0, L-1]$ uniformly, independently and at random, where integer L is a parameter*. A side length l of a square is fixed to 120. (We tested only those instances in which coordinates (x_i, y_i) and side length l are given by integers, but the integrality does not restrict generality of instances with real coordinates. See Appendix B for more details.) Demands d_i of points i are also integers, randomly chosen from $[1, 9]$. Each instance is defined by three parameters; density μ, capacity of a square c and number of points n, where the density $\mu = n(l/L)^2$ is the average number of points that fall into a square with side length l. We generated an instance for every combination of $\mu \in \{4, 9, 16\}$, $c \in \{\lceil 5\mu/2\rceil, 5\mu, \lceil 15\mu/2\rceil, \infty\}$ and $n \in \{100, 400, 900, 1600\}$. Therefore, there are 48 instances in total.

We solved the above 48 instances by the set covering and metaheuristic approaches. In the set covering approach of Section 3, we transformed each instance to the corresponding SCP instance by using the method given in Appendix A. For each instance, Table 1 shows the number of maximal valid subsets enumerated, and the computation time (in seconds) required for the enumeration. For two instances $(\mu, c, n) = (16, 80, 900)$ and $(16, 80, 1600)$, the enumeration algorithm did not terminate after 10 hours due to a huge number of maximal valid subsets. From the Table 1, we can confirm that a higher density μ increases the number of maximal valid subsets more significantly for capacitated cases. As for the tightness of the capacity constraints, we observed that in most cases the number of maximal valid subsets is largest if capacity c is set to 5μ. This phenomenon can intuitively be explained as follows. Let $\mathcal{G}_\kappa$ denote the collection of sets of κ points that satisfy the geometric constraint (1). (A subset in $\mathcal{G}_\kappa$ becomes valid if it satisfies the capacity constraint as well.) Since, in our instances, points are distributed uniformly and randomly with density μ, we can consider that the size of $\mathcal{G}_\kappa$ becomes maximum at $\kappa = \mu$, and decreases exponentially as $|\kappa - \mu|$ increases. First, let us compare the cases of $c = \lceil 5\mu/2\rceil$ and $\mu/2$. In the former case, only those subsets with less than around $\lceil \mu/2\rceil$ points can be valid (notice that the average demand d_i of point i is 5), while those with μ points can be valid in the latter case. Since $|\mathcal{G}_\kappa|$ is much smaller than $|\mathcal{G}_\mu|$ if κ is similar to $\lceil \mu/2\rceil$, the number of subsets enumerated is smaller if $c = \lceil 5\mu/2\rceil$. On the other hand, if capacity c gets much larger than μ, maximal valid subsets are likely to contain more than μ points. Since the number of such subsets, containing many points, are relatively few, the number of maximal valid subsets to

*These instances are available at `http://www-or.amp.i.kyoto-u.ac.jp/~yagiura/sqcov/`.

Table 1: Reduction of CSCP instances to SCP instances

instance			#subset	time
μ	c	n		
4	10	100	382	<1
4	10	400	1,398	<1
4	10	900	3,945	<1
4	10	1600	6,867	<1
4	20	100	1,138	<1
4	20	400	5,332	<1
4	20	900	15,295	<1
4	20	1600	25,537	<1
4	30	100	789	<1
4	30	400	6,973	<1
4	30	900	19,075	<1
4	30	1600	29,064	<1
4	∞	100	103	<1
4	∞	400	484	<1
4	∞	900	1,211	<1
4	∞	1600	2,177	<1

instance			#subset	time
μ	c	n		
9	23	100	25,087	<1
9	23	400	100,009	1
9	23	900	350,899	5
9	23	1600	667,104	10
9	45	100	36,789	<1
9	45	400	541,652	10
9	45	900	1,669,283	32
9	45	1600	3,621,057	75
9	68	100	748	<1
9	68	400	306,296	7
9	68	900	547,681	14
9	68	1600	1,608,035	45
9	∞	100	157	<1
9	∞	400	810	<1
9	∞	900	2,077	<1
9	∞	1600	3,683	<1

instance			#subset	time
μ	c	n		
16	40	100	632,678	167
16	40	400	23,674,855	533
16	40	900	128,706,774	3460
16	40	1600	256,311,677	7557
16	80	100	2,947,030	104
16	80	400	94,377,153	3039
16	80	900	—	—
16	80	1600	—	—
16	120	100	392	<1
16	120	400	2,324,877	102
16	120	900	98,355,528	5291
16	120	1600	441,429,970	24944
16	∞	100	195	<1
16	∞	400	1,099	<1
16	∞	900	3,056	<1
16	∞	1600	5,512	<1

enumerate decreases, compared to the case of $c = \mu$.

We solved the resultant SCP instances (except those two) by a heuristic algorithm proposed in [15] with the maximal time limit of 1800 seconds, excluding the enumeration time of maximal valid subsets. This algorithm also gives a lower bound based on the Lagrangian relaxation. In the metaheuristic approach of Section 4, we tested the four algorithms in Sections 4 and 5, MLS, MTS, SSLS and SSTS, for the improvement phase. These four algorithms are all equipped with the adaptive control mechanism of penalty weights (Section 5.4). For all CSCP algorithms, we also set the computation time bound to 1800 seconds. However, those with MLS and MTS terminate before the time limit expires, when no feasible solution is found in the last m runs of local search and tabu search, respectively, after the best solution with m squares is obtained (i.e., all m initial solutions have failed).

The results are summarized in Tables 2–4, corresponding to the instances with $\mu = 4$, 9 and 16, respectively. In these tables, for each instance, we show the results of the set covering approach (SC) and CSCP algorithms with four different metaheuristics (MLS, MTS, SSLS and SSTS), respectively. In each entry, the upper row gives the objective value, and the lower row gives the computation time required to find the best feasible solution (t_{best}) and the total computation time (t_{total}), both in seconds, in the format $t_{\text{best}}/t_{\text{total}}$. Objective values in bold face indicate the best result obtained by the five algorithms. The tables also show a lower bound obtained by the set covering algorithm (LB), and an upper bound obtained by CONSTRUCTION (CON) (construction phase of the CSCP algorithm). (We do not list the computation time of CONSTRUCTION, because it took less than 0.5 seconds for each instance.) If a lower bound is shown in bold face, it means that the value coincides with an upper bound, and hence, it gives the optimal value. In the columns of LB and SC, '—' means that we failed to execute the set covering approach because the enumeration algorithm did not stop within 10 hours, or the size of the input data exceeded the memory limit of the computer.

From the results, we can see that the performance of the set covering approach varies considerably with the type of instances. It works quite well for those instances with a lower density μ or no capacity constraint (i.e., $c = \infty$), while its performance deteriorates unacceptably as the density becomes higher for moderately tight capacity constraint; for such cases, it is often impossible even to execute the set covering algorithm.

Compared to the set covering approach, metaheuristic approaches are more robust. We observe that CONSTRUCTION generates a feasible solution with at most about 30% more squares than the lower bound. MLS is not powerful enough to find solutions of high quality, but it can be improved by incorporating the components of tabu search and scatter search. Although a fair comparison between MTS and SSLS is difficult because their termination criteria are different, we can observe that MTS finds better solutions in less computation time. If the computation time is of less concern, SSTS brings the best results among the tested algorithms.

Our CSCP algorithms use random bits in the improving phase. In order to investigate the reliability of the results, we solved each instance five times, using different random seeds, with SSTS. For 35 instances (including all with $n \leq 400$) out of 48, the number of squares in the output solution is the same for all five runs. For other 11 instances, the difference of the best and worst objective values of the five runs is one, and for the other two instances, the difference is two. This result indicates that our CSCP algorithm is not much affected by different random seeds.

Table 2: Computational results of the set covering approach and the metaheuristic approaches with four different algorithms: MLS, MTS, SSLS and SSTS (for instances with density 4)

instance			LB	SC	CON.	metaheuristic algorithms			
μ	c	n				MLS	MTS	SSLS	SSTS
4	10	100	**49**	**49** <0.1/1800	54	51 <0.1/0.1	**49** 0.1/3.3	**49** 0.6/1800	**49** 0.2/1800
4	10	400	**213**	**213** 919.5/1800	230	220 1.7/2.9	**213** 73.5/382.8	214 307.8/1800	**213** 46.3/1800
4	10	900	**478**	**478** 197.5/1800	514	491 37.7/45.5	**478** 1110.1/1800	480 1791.8/1800	**478** 981.5/1800
4	10	1600	**848**	**848** 19.6/1800	905	867 233.9/256.5	851 1665.6/1800	870 1711.4/1800	850 1604.5/1800
4	20	100	**25**	**25** 3.3/1800	28	26 <0.1/0.1	**25** <0.1/1.6	**25** 0.3/1800	**25** <0.1/1800
4	20	400	100	**103** 187.7/1800	115	110 0.4/1.2	104 6.9/104.7	104 199.3/1800	**103** 24.9/1800
4	20	900	224	**230** 686.8/1800	256	245 8.1/13.4	231 200.8/1137.2	233 518.9/1800	**230** 1152.4/1800
4	20	1600	398	**407** 1299.1/1800	459	438 40.7/59.0	409 1132.4/1800	422 1726.1/1800	411 389.7/1800
4	30	100	**19**	**19** 0.7/1800	23	20 <0.1/<0.1	**19** <0.1/1.6	**19** 0.5/1800	**19** <0.1/1800
4	30	400	72	77 53.5/1800	88	82 0.6/1.1	**76** 11.9/83.6	**76** 399.4/1800	**76** 21.8/1800
4	30	900	160	168 1087.8/1800	191	185 1.8/5.7	**167** 340.0/982.4	171 1318.9/1800	**167** 764.0/1800
4	30	1600	284	**299** 1422.6/1800	345	328 23.6/35.8	302 715.2/1800	312 907.8/1800	301 1071.3/1800
4	∞	100	**18**	**18** <0.1/1800	21	19 <0.1/<0.1	**18** <0.1/1.1	**18** 0.2/1800	**18** <0.1/1800
4	∞	400	**66**	**66** 0.1/1800	76	72 0.6/0.9	**66** 2.7/70.7	68 57.9/1800	**66** 4.2/1800
4	∞	900	**146**	147 0.4/1800	169	164 3.0/5.6	147 57.7/775.9	153 984.2/1800	**146** 600.9/1800
4	∞	1600	257	**262** 1.7/1800	293	285 9.8/17.8	263 1336.3/1800	273 1744.3/1800	**262** 755.6/1800

LB: Lower bound obtained by set covering approach

SC: Set covering approach using the algorithm in reference [15]

CON.: CONSTRUCTION in Section 4.1

MLS: CON. + Multi-start Local Search in Section 4.2

MTS: CON. + Multi-start Tabu Search in Section 5.1

SSLS: CON. + Scatter Search with Local Search in Section 5.2

SSTS: CON. + Scatter Search with Tabu Search in Section 5.3

Table 3: Computational results of the set covering approach and the metaheuristic approaches with four different algorithms: MLS, MTS, SSLS and SSTS (for instances with density 9)

instance			LB	SC	CON.	metaheuristic algorithms			
μ	c	n				MLS	MTS	SSLS	SSTS
9	23	100	**21**	**21** 6.1/1800	**21**	**21** <0.1/0.1	**21** <0.1/1.4	**21** <0.1/1800	**21** <0.1/1800
9	23	400	**86**	**86** 1506.5/1800	91	88 0.8/3.6	**86** 1.2/86.5	**86** 13.9/1800	**86** 1.7/1800
9	23	900	**193**	202 827.8/1800	205	200 6.8/22.2	194 32.3/949.1	**193** 404.5/1800	**193** 802.9/1800
9	23	1600	343	359 1679.3/1800	367	358 105.3/154.8	345 88.1/1800	345 1186.0/1800	**344** 502.4/1800
9	45	100	**11**	12 1.0/1800	14	13 <0.1/<0.1	**11** <0.1/1.8	**11** 0.4/1800	**11** <0.1/1800
9	45	400	44	47 1341.7/1800	54	50 0.5/1.9	**46** 19.1/102.7	**46** 1191.3/1800	**46** 2.3/1800
9	45	900	97	106 1788.8/1800	121	114 13.7/22.5	102 300.2/892.3	104 224.2/1800	**101** 1629.3/1800
9	45	1600	—	—	207	198 56.4/87.5	182 241.9/1800	183 1558.1/1800	**180** 1594.5/1800
9	68	100	**10**	**10** 0.1/1800	12	12 <0.1/<0.1	**10** 0.1/2.1	**10** 0.1/1800	**10** 0.1/1800
9	68	400	36	**39** 62.6/1800	46	43 1.0/2.0	**39** 3.8/91.9	40 97.0/1800	**39** 8.4/1800
9	68	900	78	86 1651.9/1800	103	96 6.3/11.7	85 401.7/1183.3	88 1445.4/1800	**84** 674.0/1800
9	68	1600	134	149 1502.4/1800	176	161 77.8/99.9	**148** 755.3/1800	152 1156.6/1800	149 69.8/1800
9	∞	100	**10**	**10** <0.1/1800	12	11 <0.1/<0.1	**10** <0.1/1.9	**10** 0.2/1800	**10** <0.1/1800
9	∞	400	**38**	**38** 0.1/1800	43	42 0.1/0.9	**38** 4.0/93.1	**38** 543.9/1800	**38** 4.1/1800
9	∞	900	**82**	**82** 104.0/1800	99	93 4.7/8.9	83 166.8/1042.4	87 64.1/1800	83 124.2/1800
9	∞	1600	137	**143** 166.8/1800	168	166 6.4/21.3	145 679.7/1800	155 831.0/1800	146 190.9/1800

LB: Lower bound obtained by set covering approach

SC: Set covering approach using the algorithm in reference [15]

CON.: CONSTRUCTION in Section 4.1

MLS: CON. + Multi-start Local Search in Section 4.2

MTS: CON. + Multi-start Tabu Search in Section 5.1

SSLS: CON. + Scatter Search with Local Search in Section 5.2

SSTS: CON. + Scatter Search with Tabu Search in Section 5.3

Table 4: Computational results of the set covering approach and the metaheuristic approaches with four different algorithms: MLS, MTS, SSLS and SSTS (for instances with density 16)

instance			LB	SC	CON.	metaheuristic algorithms			
μ	c	n				MLS	MTS	SSLS	SSTS
16	40	100	—	—	13	**12** <0.1/0.1	**12** <0.1/1.1	**12** <0.1/1800	**12** <0.1/1800
16	40	400	—	—	53	51 0.3/3.9	**50** 0.3/73.0	**50** 4.3/1800	**50** 0.3/1800
16	40	900	—	—	117	115 0.2/22.0	**111** 24.0/874.4	**111** 160.5/1800	**111** 48.1/1800
16	40	1600	—	—	208	205 2.3/77.5	**198** 27.4/1800	**198** 1035.2/1800	**198** 6.4/1800
16	80	100	—	—	9	8 <0.1/0.1	**7** <0.1/2.8	**7** 0.2/1800	**7** <0.1/1800
16	80	400	—	—	32	31 0.1/1.7	**26** 69.2/175.0	27 57.4/1800	**26** 55.5/1800
16	80	900	—	—	69	64 12.9/27.5	**58** 130.6/977.4	59 1673.2/1800	**58** 602.0/1800
16	80	1600	—	—	128	119 28.6/76.7	**103** 933.5/1800	107 796.8/1800	104 74.2/1800
16	120	100	**7**	**7** 0.1/1800	9	8 <0.1/0.1	**7** <0.1/3.4	**7** 2.4/1800	**7** <0.1/1800
16	120	400	—	—	30	30 <0.1/1.6	**25** 0.8/109.7	26 7.8/1800	**25** 0.8/1800
16	120	900	—	—	63	60 7.6/16.6	**52** 25.0/1354.5	55 869.7/1800	**52** 116.6/1800
16	120	1600	—	—	111	106 4.8/33.7	**94** 148.6/1800	105 39.6/1800	95 318.4/1800
16	∞	100	**7**	**7** <0.1/1800	9	8 <0.1/<0.1	**7** <0.1/3.3	**7** 0.1/1800	**7** <0.1/1800
16	∞	400	**25**	**25** 0.3/1800	28	27 <0.1/1.0	**25** 0.1/108.9	**25** 1.8/1800	**25** 0.1/1800
16	∞	900	49	**52** 16.7/1800	61	59 3.9/10.8	**52** 52.8/1521.0	57 54.5/1800	**52** 407.9/1800
16	∞	1600	86	**92** 556.1/1800	111	105 4.5/28.8	94 84.7/1800	102 859.1/1800	94 138.1/1800

LB: Lower bound obtained by set covering approach

SC: Set covering approach using the algorithm in reference [15]

CON.: CONSTRUCTION in Section 4.1

MLS: CON. + Multi-start Local Search in Section 4.2

MTS: CON. + Multi-start Tabu Search in Section 5.1

SSLS: CON. + Scatter Search with Local Search in Section 5.2

SSTS: CON. + Scatter Search with Tabu Search in Section 5.3

7 Conclusion

In this paper, we proposed the set covering and the metaheuristic approaches for the capacitated square covering problem. In the metaheuristic approach, we prepare an initial feasible solution by a constructive method, which has an approximation ratio of four in general, and two for the uncapacitated case. For the improvement phase, we tested four types of metaheuristics, MLS, MTS, SSLS, and SSTS, each equipped with the adaptive control mechanism of penalty weights. From the experimental results on randomly generated instances, we conclude as follows.

- If there is no capacity constraint, or the density of points is low, the set covering approach performs well.
- For other types of instances, the metaheuristic approach with MTS and SSTS give good solutions.
- If a relatively large amount of computation time is available, the metaheuristic approach with SSTS is the best choice among the tested algorithms.

As the best feasible solutions obtained by our algorithms often reach the lower bound, our algorithm appears to be practically effective.

Acknowledgments

This research was partially supported by Grants-in-Aid for Scientific Research from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

[1] R.K. Ahuja, T.L. Magnanti and J.B. Orlin, *Network flows: Theory, Algorithms, and Applications*, Prentice Hall, New Jersey, 1993.

[2] D. Avis and K. Fukuda, Reverse search for enumeration, *Discrete Appl. Math.* 65 (1996) 21–46.

[3] B.S. Baker, Approximation algorithms for NP-complete problems on planar graphs, *J. ACM* 41 (1994) 153–180.

[4] R.J. Fowler, M.S. Paterson and S.L. Tanimoto, Optimal packing and covering in the plane are NP-complete, *Inform. Process. Lett.* 12 (1981) 133–137.

[5] F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic Publishers, Boston, 1997.

[6] F. Glover, M. Laguna and R. Martí, Fundamentals of Scatter Search and Path Relinking, *Control Cybernet.* 39 (2000) 653–684.

[7] D.S. Hochbaum and W. Maass, Approximation schemes for covering and packing problems in image processing and VLSI, *J. ACM* 32 (1985) 130–136.

[8] H.B. Hunt III, M.V. Marathe, V. Radhakrishnan, S.S. Ravi, D.J. Rosenkrantz and R.E. Stearns, NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs, *J. Algorithms* 26 (1998) 238–274.

[9] D.S. Johnson, Fast algorithms for bin packing, *J. Comput. System Sci.* 8 (1974) 272–314.

[10] D.S. Johnson, M. Yanakakis and C.H. Papadimitriou, On generating all maximal independent sets, *Inform. Process. Lett.* 27 (1998) 119–123.

[11] K. Makino and T. Uno, New algorithms for enumerating all maximal cliques, *Algorithm Theory — SWAT2004: 9th Scandinavian Workshop on Algorithm Theory*; *Lecture Notes in Comput. Sci.* 3111 (2004) 260–272.

[12] S. Masuyama, T. Ibaraki and T. Hasegawa, The computational complexity of the m-center problems on the plane, *The Transactions of the IECE of Japan* E64 (1981) 57–64.

[13] S. Tsukiyama, M. Ide, H. Ariyoshi and I. Shirakawa, A new algorithm for generating all the maximal independent sets, *SIAM J. Comput.* 6 (1977) 505–517.

[14] M. Yagiura, T. Ibaraki and F. Glover, A path relinking approach with ejection chains for the generalized assignment problem, *European J. Oper. Res.* to appear.

[15] M. Yagiura, M. Kishida and T. Ibaraki, A 3-flip neighborhood local search for the set covering problem, *European J. Oper. Res.* to appear.

Appendices

A. Enumeration Algorithm for Maximal Weighted Cliques

In the set covering approach in Section 3, we need to enumerate all maximal valid subsets. To achieve this, we consider the problem of enumerating all maximal weighted cliques in a graph with vertex weights.

Let $G = (V, E)$ be a graph with vertex set $V = \{1, \ldots, n\}$ and an edge set $E = \{e_1, \ldots, e_m\}$. We assume without loss of generality that G is simple and connected. Each vertex $i \in V$ is given a weight $w(i)$, where we assume that $w(i) \geq w(j)$ holds for any $i < j$, i.e., the vertices are sorted in the decreasing order of their weights. We denote the sum of weights of vertices in a vertex set S by $w(S)$. A *clique* of G is a vertex set $K \subseteq V$ such that any two vertices in K are connected by an edge. For a given constant α, a clique K is called *light* if $w(K) \leq \alpha$. If a light clique is included in no other light clique, we call it a *maximal weighted clique*.

For a given CSCP instance, we define graph (V, E) by letting $V = P$ and $E = \{(i_1, i_2) \in P \times P \mid \max\{|x_{i_1} - x_{i_2}|, |y_{i_1} - y_{i_2}|\} \leq l\}$. A constant α is given by the capacity c. Then, a set of points $S \subseteq V$ forms a clique in G if and only if it satisfies the geometric constraint (1). If S is a light clique, it satisfies the capacity constraint (2) as well, and hence S is a valid subset. Therefore, we can enumerate all maximal valid subsets by enumerating all maximal weighted cliques in (V, E). In this appendix, we describe an algorithm for enumerating all maximal weighted cliques.

In the existing studies, no output linear time algorithm to enumerate all maximal weighted cliques is known, where output linear time means that the computation time is linear in the number of output, and the computation time for each output is polynomial on average in the input size. For the problem of enumerating maximal cliques (without weights), several algorithms have been proposed by Tsukiyama, Ide, Ariyoshi & Shirakawa[13], Johnson, Yanakakis & Papadimitriou[10], and Makino & and Uno[11]. The time complexities of the algorithms in [13, 10] are the same, but the algorithm in [10] enumerates maximal cliques in a lexicographic order with the use of memory up to the size of output. The algorithms in [11] is an improved version of the algorithms in [13, 10], so that it runs faster for dense graphs and sparse graphs.

Our algorithm to enumerate all maximal weighted cliques is motivated by the algorithm of Makino and Uno[11], and we describe only the points different from [11].

For a vertex set S and a vertex i, let $S_{\leq i} = S \cap \{1, \ldots, i\}$. For two vertex sets X and Y, we say X is *lexicographically larger than* Y if the smallest vertex in $(X \setminus Y) \cup (Y \setminus X)$ is contained in X. (Let $x(S)$ denote the characteristic vector of a vertex set S; i.e., the i-th element $x_i(S)$ of $x(S)$ is 1 if $i \in S$, and 0, otherwise. Then, X is lexicographically larger than Y if and only if $x(X)$ is lexicographically larger than $x(Y)$; i.e., there exists $i \in V$ such that $x_j(X) = x_j(Y)$ for all $j < i$, and $x_i(X) > x_i(Y)$.) For a light clique K, let $C(K)$ denote the lexicographically largest light clique containing K. We note that $C(K)$ is always a maximal weighted clique.

Let K_0 denote the lexicographically largest maximal weighted clique. For a maximal weighted clique K ($\neq K_0$), we define the *parent* $P(K)$ of K by $C(K_{\leq i-1})$ such that i is the minimum vertex satisfying $C(K_{\leq i}) = K$. Such a vertex i is called the *parent vertex*, and denoted by $i(K)$. We note that the parent vertex is always included in K. We can observe that $K_{\leq i(K)-1} \subset P(K)_{\leq i(K)-1}$, thus $P(K)$ is lexicographically larger than K. Therefore, this parent-child binary relation $(P(K), K)$ on maximal weighted cliques is acyclic, and forms a tree rooted at K_0. We call this tree the *enumeration tree* for maximal weighted cliques of a graph G, vertex weight w, and constant α.

By traversing the enumeration tree in a depth-first search manner starting from K_0, we can visit all maximal weighted cliques. To operate the depth-first search without storing the whole enumeration tree, we generate the children of a vertex only when it becomes necessary and eliminate the information of a vertex when depth-first search backtracks from the vertex. In the following, we explain the way to generate the children of the current maximal weighted clique.

Let K be a maximal weighted clique. For a vertex $i \notin K$ not adjacent to all vertices in $K_{\leq i}$, we define $K[i]$ by $C((K_{\leq i} \cap \Gamma(i)) \cup \{i\})$, where $\Gamma(i) = \{j \in V \mid (i,j) \in E\}$. Since there is a vertex $j \in K_{\leq i} \setminus \Gamma(i)$, which satisfies $w(i) \leq w(j)$ from $i > j$, we can see that $K_{\leq i} \cap \Gamma(i) \cup \{i\}$ is a light clique, and $K[i]$ is well defined.

For a vertex $i \notin K$ adjacent to all vertices in $K_{\leq i}$ and a vertex $j \in K; j < i$, we define $K[i,j]$ by $C(K_{\leq i} \cup \{i\} \setminus \{j\})$. Since $w(i) \leq w(j)$ holds, we can see that $K_{\leq i} \cup \{i\} \setminus \{j\}$ is a light clique, and $K[i,j]$ is well defined.

Lemma 1 *For two maximal weighted cliques K and K', K' is a child of K only if $K' = K[i(K')]$ holds or $K' = K[i(K'), j]$ holds for some j.*

Proof: Suppose that K' is a child of K. Then, $K = C(K'_{\leq i(K')-1})$ for which $K'_{\leq i(K')-1} \subset K_{\leq i(K')-1}$ holds. Let j be the minimum vertex in $K \setminus K'$. Then, we consider the following two cases.

(1) j is not adjacent to $i(K')$. In this case, we claim that no vertex in $K_{\leq i(K')} \setminus K'$ is adjacent to $i(K')$. Under this claim, $(K_{\leq i(K')} \cap \Gamma(i(K'))) \cup \{i(K')\}$ is equal to $K'_{\leq i(K')}$ since $K'_{\leq i(K')-1} \subset K_{\leq i(K')-1}$, hence $K' = K[i(K')]$.

To prove the claim by contradiction, suppose that a vertex $j' \in K_{\leq i(K')} \setminus K'$ is adjacent to $i(K')$. Then, we have $\alpha \geq w(K) \geq w(K'_{\leq i(K')}) - w(i(K')) + w(j) + w(j')$. This together with $w(i(K')) \leq w(j)$, we have $w(K'_{\leq i(K')} \cup \{j'\}) \leq \alpha$. Since j' is adjacent to all vertices in K, j' is adjacent to all vertices in $K'_{\leq i(K')}$ since $K'_{\leq i(K')-1} \subset K_{\leq i(K')-1}$. This implies that

$K'_{\leq i(K')} \cup \{j'\}$ is a light clique including $K'_{\leq i(K')}$. This contradicts that $C(K'_{\leq i(K')}) = K'$; i.e., K' is the lexicographically largest light clique including $K'_{\leq i(K')}$.

(2) j is adjacent to $i(K')$. Then, from $K'_{\leq i(K')-1} \subset K_{\leq i(K')-1}$, j is adjacent to all vertices in $K'_{\leq i(K')}$. Since $C(K'_{\leq i(K')}) = K'$ does not include j, we have $w(K'_{\leq i(K')}) + w(j) > \alpha$. Thus, for any vertex $j' < i$, $w(K'_{\leq i(K')-1}) + w(j) + w(j') = w(K'_{\leq i(K')}) - w(i(K')) + w(j) + w(j') > \alpha$. This means that $K_{\leq i(K')} \setminus K'$ includes no other vertex than j. Therefore, $K_{\leq i(K')} = K'_{\leq i(K')-1} \cup \{j\}$, and we have that $K' = K[i(K'), j]$. □

From the lemma, we can see that the following algorithm computes the children of a maximal weighted clique K by evaluating at most $|V|^2$ candidates.

```
1. For each $K' = K[i]$ or $K[i, j]$ do
2.     If $P(K') = K$ then output $K'$
```

Since, for any light clique S, $C(S)$ can be computed in $O(|V| + |E|)$ time by augmenting S repeatedly, each of $K[i]$ and $K[i, j]$ can be constructed in $O(|V| + |E|)$ time. Given the parent vertex of K', we can construct its parent $P(K')$ in $O(|V| + |E|)$ time, and hence we can check in $O(|V| + |E|)$ time whether $P(K') = K$ holds or not. The computation time for generating all children is then $O(|V|^2(|V| + |E|)) = O(|V|^2|E|)$. Therefore, we obtain the following theorem.

Theorem 2 *For a given graph $G = (V, E)$, a vertex weight w and a constant α, all maximal weighted cliques can be enumerated in $O(|V|^2|E|)$ time for each.* □

In practical computation, the input graph is often sparse. We here consider the time complexity with respect to the maximum degree Δ of G.

Suppose that a child K' of K satisfies $K' = K[i(K')]$. According to [11], if $K_{\leq i(K')} \cap \Gamma(i(K')) = \emptyset$, then $P(K') = K_0$ holds in the case that $K'_{\leq i(K')-1} = \emptyset$, and $K'_{\leq i(K')-1} \not\subseteq K_{\leq i(K')-1}$ holds otherwise. It implies that K is not the parent of K' if $K \neq K_0$. Thus, for a maximal weighted clique $K \neq K_0$, $K' = K[i(K')]$ is a child of K only if $K_{\leq i(K')} \cap \Gamma(i(K')) \neq \emptyset$. Since $|K| \leq \Delta + 1$, at most $\Delta(\Delta + 1)$ vertices satisfy $K_{\leq i(K')} \cap \Gamma(i(K')) \neq \emptyset$.

Suppose that a child K' of K does not satisfy $K' = K[i(K')]$. Then, from Lemma 1, $K' = K[i(K'), j]$ holds for some $j \in K$, and $i(K')$ is adjacent to all vertices in $K_{\leq i(K')-1}$. Hence, the number of such pairs of i and j is at most $\Delta(\Delta + 1)$.

Since we can construct $C(S)$ in $O(\Delta^2)$ time for any light clique S, we obtain the following lemma.

Lemma 2 *We can enumerate all children of a given maximal weighted clique other than K_0 in $O(\Delta^4)$ time.* □

From the lemma, we obtain the following theorem.

Theorem 3 *We can enumerate all maximal weighted cliques in $O(\Delta^4)$ time for each, with $O(|V|^2|E|)$ preprocessing time.* □

We note that the additional $O(|V|^2|E|)$ time is the computation time to find children of K_0.

B. Scaling Input Data to Integers

Given n points (x'_i, y'_i) $(i = 1, 2, \ldots, n)$ in the plane and a real number $l' > 0$, where x'_i and y'_i are real numbers, we consider the problem of mapping those points (x'_i, y'_i) and l' to nonnegative integer points (x_i, y_i) $(i = 1, 2, \ldots, n)$ and a positive integer l, respectively, so that

$$x'_j - x'_i \le l' \iff x_j - x_i \le l, \qquad \forall i, j, \tag{6}$$

$$y'_j - y'_i \le l' \iff y_j - y_i \le l, \qquad \forall i, j, \tag{7}$$

and l is the minimum.

Note that constraints (6) and (7) assure that

$$\max\{|x'_i - x'_j|, |y'_i - y'_j|\} \le l' \iff \max\{|x_i - x_j|, |y_i - y_j|\} \le l, \quad \forall i, j.$$

Therefore, the CSCP instance with the mapped points (x_i, y_i) and l is equivalent to the instance with the original data.

For a fixed integer $\lambda > 0$, we first consider the problem (P_λ) of finding an integer solution $\boldsymbol{x} = (x_i \mid 1 \le i \le n)$ that satisfies (6) with $l = \lambda$. Without loss of generality, we assume that $n \ge 2$ and $x'_1 \le x'_2 \le \cdots \le x'_n$ hold. In this case, we can also assume that $x_1 \le x_2 \le \cdots \le x_n$ holds (monotonicity), because if there is a solution to (P_λ), say $\tilde{\boldsymbol{x}} = (\tilde{x}_i \mid 1 \le i \le n)$, then so is $\boldsymbol{x}^* = (\max\{\tilde{x}_1, \tilde{x}_2, \ldots, \tilde{x}_i\} \mid 1 \le i \le n)$, which satisfies monotonicity. Suppose that $x'_j - x'_i \le l'$ holds. Let x^*_j be given by $\tilde{x}_k$, where $1 \le k \le j$. Then, $x'_k - x'_i \le l'$ and thus $\tilde{x}_k - \tilde{x}_i \le \lambda$ holds by assumption, which implies that $x^*_j - x^*_i \le \tilde{x}_k - \tilde{x}_i \le \lambda$. Similarly, we can show that $x'_j - x'_i > l' \Longrightarrow x^*_j - x^*_i > \lambda$ holds for any i and j. Therefore, $\boldsymbol{x}^*$ is a solution to (P_λ). Then, (P_λ) is formulated as the following integer linear system:

$$\begin{array}{lll}
(\mathrm{P}_\lambda) & x_{k(i)+1} - x_i \ge \lambda + 1, & \forall i \ \text{ with } \ k(i) + 1 \le n, \\
 & x_{k(i)} - x_i \le \lambda, & \forall i \ \text{ with } \ k(i) > i, \\
 & x_i \le x_{i+1}, & 1 \le i < n, \\
 & x_i : \text{nonnegative integer}, & 1 \le i \le n,
\end{array}$$

where $k(i) = \max\{j \mid x'_j \le x'_i + l'\}$. For this, we define a directed network $N(\lambda)$ as follows:

- Node set $V = \{1, \ldots, n\}$.
- Arc set $A = A_f \cup A_b \cup A_a$, where $A_f = \{(i, k(i) + 1) \mid 1 \le i < n, k(i) + 1 \le n\}$, $A_b = \{(k(i), i) \mid 1 \le i < n, k(i) > i\}$ and $A_a = \{(i, i + 1) \mid 1 \le i < n\}$.
- Length of arc $d(i,j) = \begin{cases} -(\lambda + 1), & \text{if } (i,j) \in A_f, \\ \lambda, & \text{if } (i,j) \in A_b, \\ 0, & \text{if } (i,j) \in A_a. \end{cases}$

It is easy to see that if $N(\lambda)$ has a cycle of negative length (called a negative cycle), then (P_λ) has no solution. On the other hand, suppose that $N(\lambda)$ has no negative cycle. Then, $N(\lambda)$ has a shortest path from node 1 to each i $(1 \le i \le n)$, whose length $\mathit{dist}(i)$ is a non-positive integer, and $\mathit{dist}(j) \le \mathit{dist}(i) + d(i,j)$ holds for any $(i,j) \in A$. This implies that $\boldsymbol{x} = (-\mathit{dist}(i) \mid 1 \le i \le n)$ gives a solution to (P_λ). Therefore, we can either conclude that (P_λ) is infeasible, or find a solution $\boldsymbol{x}$ to (P_λ) in $\mathrm{O}(n^2)$ time by applying an appropriate shortest path algorithm (e.g., the label-correcting algorithm [1]) to $N(\lambda)$.

Next, we show that (P_λ) is always feasible for $\lambda = \lfloor (n-1)/2 \rfloor$; i.e., $N(\lfloor (n-1)/2 \rfloor)$ has no negative cycle. To prove this, we first claim that, for a sufficiently large integer λ such that $\tilde{\lambda} \geq 3l'/\delta + 1$, where $\delta = \min\{x'_j - x'_i - l' \mid 1 \leq i < n, j = k(i) + 1 \leq n\}$, network $N(\tilde{\lambda})$ has no negative cycle. This claim is true because $\boldsymbol{x} = (\lfloor \tilde{\lambda} x'_i / l' \rfloor \mid 1 \leq i \leq n)$ is a solution to $(\mathrm{P}_{\tilde{\lambda}})$.

Now let C be any simple cycle in $N(\lambda)$ containing m_f (≥ 0) arcs in A_f and m_b (> 0) arcs in A_b. Then, m_f must be smaller than m_b, since otherwise the length of C, $m_b\lambda - (\lambda + 1)m_f = (m_b - m_f)\lambda - m_f$, becomes negative for any λ (in particular for $\lambda = \tilde{\lambda}$), which is a contradiction. Since the number of arcs in $A_f \cup A_b$ contained in C, $m_f + m_b$, is at most n, $m_f < m_b$ implies $m_f \leq \lfloor (n-1)/2 \rfloor$, and hence the length of C is at least $\lambda - \lfloor (n-1)/2 \rfloor$. (This bound is tight. Figure 2 shows a network having a cycle of length $\lambda - \lfloor (n-1)/2 \rfloor$.) Therefore, if we set $\lambda = \lfloor (n-1)/2 \rfloor$, network $N(\lambda)$ has no negative cycle, and hence (P_λ) has a solution. Furthermore, if $(\mathrm{P}_{\lambda'})$ has a solution, then so does $(\mathrm{P}_{\lambda''})$ for any $\lambda'' > \lambda'$, because the length of any simple cycle on $N(\lambda)$ is nondecreasing in λ.

Since the argument above applies to the problem of finding $\boldsymbol{y}$ for a fixed l, a solution $(\boldsymbol{x}, \boldsymbol{y}, l^*)$ with the minimum integer l^* can be obtained in $\mathrm{O}(n^2 \log n)$ time by using a binary search between $1 \leq l \leq \lfloor (n-1)/2 \rfloor$. (Recall that, for a fixed l, we can identify a solution $(\boldsymbol{x}, \boldsymbol{y}, l)$ in $\mathrm{O}(n^2)$ time.)

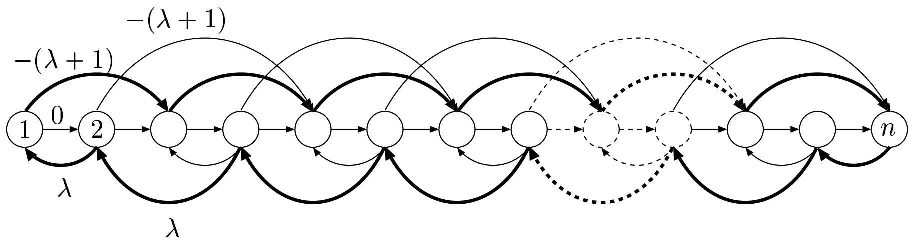

Figure 2: An example of network $N(\lambda)$ with a cycle of length $\lambda - \lfloor (n-1)/2 \rfloor$.

Manuscript received 28 October 2004
revised 12 April 2005
accepted for publication 16 April 2005

Endre Boros
Rutgers Center for Operations Research, Rutgers University, The State University of New Jersey,
640 Bartholomew Road, Piscataway, NJ 08854-8003, USA
E-mail address: `boros@rutcor.rutgers.edu`

Toshihide Ibaraki
Department of Informatics, School of Science and Technology, Kwansei Gakuin University,
2-1 Gakuen, Sanda, Hyogo 669-1337, Japan
E-mail address: `ibaraki@ksc.kwansei.ac.jp`

Hiroya Ichikawa
Sumisho Computer Systems Corporation, Harumi Island Triton Square Office Tower Z, 1-8-12 Harumi,
Chuo-ku, Tokyo 104-6241, Japan
E-mail address: `h.ichikawa@jpta.scs.co.jp`

Koji Nonobe
Department of Art and Technology, Faculty of Engineering, Hosei University,
3-7-2 Kajino-cho, Koganei, Tokyo 184-8584, Japan
E-mail address: `nonobe@k.hosei.ac.jp`

Takeaki Uno
National Institute of Informatics, 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
E-mail address: `uno@nii.jp`

Mutsunori Yagiura
Department of Applied Mathematics and Physics, Graduate School of Informatics,
Kyoto University, Kyoto 606-8501, Japan
E-mail address: `yagiura@i.kyoto-u.ac.jp`