数理解析研究所講究録
第1773巻 2012年 231-238

頂点容量付き有向全域木パッキング問題に対するラグランジュ緩和ヒューリスティック

田中 勇真[†]　今堀 慎治[‡]　柳浦 睦憲[†]
[†]名古屋大学大学院 情報科学研究科 計算機数理科学専攻
[‡]名古屋大学大学院 工学研究科 計算理工学専攻

概要

本研究では, 頂点容量付き有向全域木パッキング問題を扱う. この問題は入力として, 有向グラフ, ルート頂点, 頂点容量, 辺の始点側と終点側それぞれに消費量が与えられる. 目的はルート頂点に流入する有向全域木のパッキング回数を最大化することである. ただし, 有向全域木の各頂点に対する消費量の合計は, 与えられた頂点容量を超えてはいけない. この問題はNP困難であることが知られている. 以前, 我々はこの問題に対して2段階のアルゴリズムを提案した. このアルゴリズムは, 1段階目に有望であると考えられる木の候補を生成し, 2段階目に生成したそれぞれの木のパッキング回数を決定する. 本研究では, 線形緩和の代わりにラグランジュ緩和を用いることによって1段階目の改善を行った. ランダムに生成されたグラフに対して計算実験を行ったところ, 提案アルゴリズムは以前のアルゴリズムより速く木を生成でき, 少ない木の候補でも良い解を得ることを確認した.

1 Introduction

In this paper, we consider the *node capacitated in-tree packing problem* (abbreviated as NCIPP). The input consists of a directed graph, a root node, a node capacity function and edge consumption functions for heads and tails. The objective of the problem is to find the maximum number of rooted spanning in-trees such that the total consumption of the in-trees at each node does not exceed the capacity of the node.

Let $G = (V, E)$ be a directed graph, $r \in V$ be a root node and $\mathbb{R}_+$ be the set of nonnegative real numbers. In addition, let $t : E \to \mathbb{R}_+$ and $h : E \to \mathbb{R}_+$ be tail and head consumption functions on directed edges, respectively, and $b_i \in \mathbb{R}_+$ be the capacity of a node $i \in V$. For convenience, we define T_{all} as the set of all spanning in-trees rooted at the given root $r \in V$ in the graph G. Let $\delta_j^+(i)$ (resp., $\delta_j^-(i)$) be the set of edges in an in-tree $j \in T_{\mathrm{all}}$ leaving (resp., entering) a node $i \in V$. The consumption a_{ij} of an in-tree $j \in T_{\mathrm{all}}$ at a node $i \in V$ is defined as

$$a_{ij} = \sum_{e \in \delta_j^+(i)} t(e) + \sum_{e \in \delta_j^-(i)} h(e). \tag{1}$$

We call the first term of this equation (1) *tail consumption*, and the second term *head consumption*. The node capacitated in-tree packing problem is to find a subset $T \subseteq T_{\mathrm{all}}$ of spanning in-trees and the packing number x_j of each in-tree $j \in T$ subject to the node capacity restriction

$$\sum_{j \in T} a_{ij} x_j \leq b_i, \quad \forall i \in V, \tag{2}$$

so as to maximize the total number of packed in-trees $\sum_{j \in T} x_j$. Throughout this paper, an in-tree means a spanning in-tree even if we do not clearly state spanning.

This problem is known to be NP-hard [10]. Furthermore, it is still NP-hard even if instances are restricted to complete graphs embedded in a space with tail consumptions depending only on the distance between end nodes.

This problem is studied in the context of sensor networks. Recently, several kinds of graph packing problems are studied in the context of ad hoc wireless networks and sensor networks. These problems are called *network lifetime problems*. The important problems included among this category are the node

capacitated spanning subgraph packing problems [3, 8, 11]. For sensor networks, for example, a spanning subgraph corresponds to a communication network topology for collecting information from all nodes (sensors) to the root (base station) or for sending information from the root to all other nodes. Sending a message along an edge consumes energy at end nodes, usually depending on the distance between them. The use of energy for each sensor is severely limited because the sensors use batteries. It is therefore important to design the topologies for communication in order to save energy consumption and make sensors operate as long as possible. For this problem, Heinzelman et al. [8] proposed an algorithm, called LEACH-C (low energy adaptive clustering hierarchy centralized), that uses arborescences with limited height for communication topologies. For more energy effcient communication networks, a multiround topology construction problem was formulated as an integer programming problem, and a heuristic solution method was proposed in [11]. In the formulation of [3], head consumptions are not considered, and the consumption at each node is the maximum tail consumption among the edges leaving the node. There are variations of the problem with respect to additional conditions on the spanning subgraph such as strong connectivity, symmetric connectivity, and directed out-tree rooted at a given node. Calinescu et al. [3] discussed the hardness of the problem and proposed several approximation algorithms.

For problem NCIPP, we proposed a two-phase algorithm [12]. In the first phase, it generates candidate in-trees to be packed. The node capacitated in-tree packing problem can be formulated as an IP (integer programming) problem, and the proposed algorithm employs the column generation method for the LP(linear programing)-relaxation of the problem to generate promising candidate in-trees. In the second phase, the algorithm computes the packing number of each in-tree. Our algorithm solves this second-phase problem by first modifying feasible solutions of the LP-relaxation problem and then improving them with a greedy algorithm.

In this paper, we propose a new first-phase algorithm. The new algorithm employs the Lagrangian relaxation instead of the LP-relaxation, and it uses the subgradient method to obtain a good Lagrangian multiplier vector. One of the merits of the classical subgradient method is that it is simple and easy to implement; however, it was rather slow and took long time to generate sufficient number of in-trees. To alleviate this, we incorporate various ideas to speed up the algorithm, e.g., rules to decrease the number of in-trees used for the subgradient method, and upper and lower bounding techniques to reduce the number of times some Lagrangian multipliers are updated.

We conducted computational experiments on randomly generated instances with up to 200 nodes. The results show that the new algorithm obtains solutions that deviate at most 1% from upper bounds, and comparisons with the previous algorithm show that our new method works more efficiently for large instances of this problem.

2 Formulation

The node capacitated in-tree packing problem can be formulated as the following IP problem:

$$\begin{aligned} \text{maximize} \quad & \sum_{j \in T_{\text{all}}} x_j, \\ \text{subject to} \quad & \sum_{j \in T_{\text{all}}} a_{ij} x_j \leq b_i, \quad \forall i \in V, \\ & x_j \geq 0, \qquad\qquad x_j \in \mathbb{Z}, \ \forall j \in T_{\text{all}}. \end{aligned} \tag{3}$$

The notations are summarized as follows:

V: the set of nodes,
T_{all}: the set of all spanning in-trees rooted at the given root $r \in V$,
a_{ij}: the consumption (defined by equation (1)) of an in-tree $j \in T_{\text{all}}$ at a node $i \in V$,
b_i: the capacity of a node $i \in V$,
x_j: the packing number of an in-tree $j \in T_{\text{all}}$,
$\mathbb{Z}$: the set of all integers.

We defined T_{all} as the set of all in-trees rooted at the given root $r \in V$. However, the number of in-trees in T_{all} can be exponentially large, and it is difficult in practice to handle all of them. We therefore

consider a subset $T \subseteq T_{\rm all}$ of in-trees and deal with the following problem:

$$
\begin{array}{lll}
P(T) \quad \text{maximize} & \displaystyle\sum_{j\in T} x_j, & \\
\text{subject to} & \displaystyle\sum_{j\in T} a_{ij}x_j \le b_i, & \forall i \in V, \\
& x_j \ge 0, & x_j \in \mathbb{Z},\ \forall j \in T.
\end{array}
$$

If $T = T_{\rm all}$, the problem $P(T_{\rm all})$ is equivalent to the original problem (3). We denote the optimal value of $P(T)$ by $\mathrm{OPT}_{P(T)}$.

To consider the Lagrangian relaxation problem of $P(T)$, the maximum packing number u_j of each in-tree $j \in T$ is defined as $u_j = \min_{i\in V:\, a_{ij}>0} \lfloor b_i/a_{ij} \rfloor$ (where $\lfloor y \rfloor$ stands for the floor function of y). The Lagrangian relaxation problem is formally described as follows:

$$
\begin{array}{lll}
LR(T,\lambda) \quad \text{maximize} & \displaystyle\sum_{j\in T} x_j + \sum_{i\in V} \lambda_i \left(b_i - \sum_{j\in T} a_{ij}x_j \right) & \\
& \displaystyle= \sum_{j\in T} c_j(\lambda)x_j + \sum_{i\in V} \lambda_i b_i, & \\
\text{subject to} & 0 \le x_j \le u_j, & \forall j \in T
\end{array}
$$

where $\lambda_i \ge 0$ is the Lagrangian multiplier for a node $i \in V$, $\lambda = (\lambda_1, \ldots, \lambda_{|V|})$ is the vector of Lagrangian multipliers, and $c_j(\lambda) = 1 - \sum_{i\in V} a_{ij}\lambda_i$ is the relative cost of an in-tree $j \in T$. We denote the optimal value of $LR(T,\lambda)$ by $\mathrm{OPT}_{LR(T,\lambda)}$ and an optimal solution of $LR(T,\lambda)$ by $x(\lambda)$. For any $\lambda \ge 0$, an optimal solution $x(\lambda)$ can be calculated easily as follows: For all $j \in T$, $x_j(\lambda) = u_j$ if $c_j(\lambda) > 0$, otherwise $x_j(\lambda) = 0$. In general, $\mathrm{OPT}_{LR(T,\lambda)}$ gives an upper bound of $\mathrm{OPT}_{P(T)}$ for any $\lambda \ge 0$.

3 New In-Trees Generating Algorithm

In this section, we explain the new algorithm to generate in-trees. Our algorithm prepares an initial set of in-trees by a simple algorithm in Section 3.1. It then generates in-trees by using the information from Lagrangian relaxation, whose details are explained in Sections 3.2–3.4. To obtain a good upper bound and a Lagrangian multiplier vector, it applies the subgradient method to a current in-tree set, and then it tries to add a new in-tree to the current in-tree set by solving a pricing problem. After adding a new in-tree, it applies the subgradient method to the new in-tree set, and the above steps are repeated until a stopping criterion is satisfied. We also explain a method that obtains good feasible solutions of $P(T_{\rm all})$ (i.e., this method corresponds to the second-phase algorithm in our previous paper [12]).

3.1 Initial set of in-trees

For the node capacitated in-tree packing problem, Imahori at el. [10] proved that it is NP-hard to find one packed in-tree that satisfies the node capacity restriction (2). Consequently, it is not always easy to create an initial set of in-trees. In this paper, we deal with problems for which this part is easy, e.g., those with $t(e) \ll b_i, \forall e \in \delta^+(i)$ and $h(e) \ll b_i, \forall e \in \delta^-(i)$ for all $i \in V$, where $\delta^+(i)$ (resp., $\delta^-(i)$) denotes the set of edges in the graph G leaving (resp., entering) a node $i \in V$. This assumption holds naturally in many applications.

The column generation method can be executed even with only one initial in-tree. However, we observed through preliminary experiments that the computation time was usually reduced if an initial set with more in-trees was given. We also observed that, for randomly generated in-trees, the computation time did not decrease much when we increased the number of in-trees in the initial set beyond $|V|$. We therefore employ $|V|$ randomly generated in-trees as the initial set of in-trees.

3.2 Subgradient method

We employ the subgradient method to obtain Lagrangian multiplier vectors λ that give good upper bounds of $P(T)$ (i.e., $\mathrm{OPT}_{LR(T,\lambda)}$) for the current set of in-trees T. The subgradient method is a well-known

heuristic approach to find a near optimal Lagrangian multiplier vector [1, 6, 9]. It uses the subgradient $s(\lambda) = (s_1(\lambda), \ldots, s_{|V|}(\lambda))$, associated with a given λ, defined by $s_i(\lambda) = b_i - \sum_{j \in T} a_{ij} x_j(\lambda)$ for all $i \in V$. This method repeatedly updates a Lagrangian multiplier vector, starting from a given initial vector, by the following formula:

$$\lambda_i := \max\left(0, \lambda_i - \pi \frac{\mathrm{UB}(\lambda) - \mathrm{LB}}{\sum_{i \in V} \{s_i(\lambda)\}^2} s_i(\lambda)\right), \quad \forall i \in V, \tag{4}$$

where $\mathrm{UB}(\lambda) = \mathrm{OPT}_{LR(T,\lambda)}$ is an upper bound of $P(T)$, LB is a lower bound of $P(T)$, and $\pi \geq 0$ is a parameter to adjust the step size. We denote $\theta(\lambda) := \pi(\mathrm{UB}(\lambda) - \mathrm{LB})/(\sum_{i \in V}\{s_i(\lambda)\}^2)$, which is called the step size in general. The parameter π is initially set to the value given to the subgradient method and is halved whenever the best upper bound is not updated in N consecutive iterations, where N is a parameter that we set $N = 30$ in our experiments. The iteration of the subgradient method is stopped when π becomes less than 0.005. In our algorithm, the above rule to update λ is slightly modified as follows: In the execution of (4), we use $s'_i(\lambda)$ instead of $s_i(\lambda)$, where $s'_i(\lambda) = 0$ if $\lambda_i = 0$ and $s_i(\lambda) < 0$ hold immediately before the execution of (4), and $s'_i(\lambda) = s_i(\lambda)$ otherwise.

Let SubOpt(LB, λ, π) be the subgradient method using a lower bound LB, starting from an initial vector λ and a parameter π. The procedure SubOpt returns ρ pairs $(\lambda^{(1)}, \pi^{(1)}), \ldots, (\lambda^{(\rho)}, \pi^{(\rho)})$ of Lagrangian multiplier vectors λ and parameters π such that for $k = 1, \ldots, \rho$, the multiplier vector $\lambda^{(k)}$ attains the kth best upper bound $\mathrm{UB}(\lambda)$ among those generated during the search, and the parameter $\pi^{(k)}$ is the value of π when $\lambda^{(k)}$ is found, where the parameter ρ specifies the number of pairs output by SubOpt. These pairs are used in the column generation method whose details are explained in the next section.

We set $\lambda_i = 2$ for all $i \in V$ as the initial Lagrangian multiplier vector and $\pi = 2$ as the initial parameter to adjust the step size if SubOpt is applied to the initial set of in-trees; otherwise (i.e., SubOpt is applied to an in-tree set after adding a new in-tree by the column generation method), the algorithm uses the information of the last execution of SubOpt as follows: The initial values of λ and π are set to $\lambda = \lambda^{(k)}$ and $\pi = \pi^{(k)}$ for the k such that the pair $(\lambda^{(k)}, \pi^{(k)})$ was used to generate the latest new in-tree by the column generation method. With this approach, SubOpt is able to decrease the number of iterations until a good Lagrangian multiplier vector is obtained.

We employ the greedy algorithm PackInTrees proposed in our previous work [12] as a method for producing a lower bound LB (feasible solution) of $P(T)$. This algorithm uses the maximum packing number, calculated based on the available capacity in each node, as the evaluation criterion of each in-tree. The proposed algorithm does not frequently update LB; PackInTrees is applied to an initial in-tree set, and then it is applied whenever a hundred new in-trees are added, because we confirmed through preliminary experiments that the performance of our algorithm was not affected much by the quality of lower bounds.

The above explanation of the algorithm describes only basic parts, but we also incorporated various ideas to speed up the algorithm, e.g., rules to decrease the number of in-trees used for the subgradient method, and upper and lower bounding techniques to reduce the number of times some Lagrangian multipliers are updated.

3.3 Column generation method

We employ the column generation method to generate candidate in-trees. It starts from an initial in-tree set $T \subseteq T_{\mathrm{all}}$ and repeatedly augments T until a stopping criterion is satisfied.

Let $T^+(\lambda)$ be the set of all in-trees having positive relative costs $c_j(\lambda) > 0$ for a Lagrangian multiplier vector λ (i.e., $T^+(\lambda) = \{j \in T_{\mathrm{all}} \mid c_j(\lambda) > 0\}$). It is clear from the method of solving $LR(T, \lambda)$ (see Section 2) that if a set of in-trees $T \subseteq T_{\mathrm{all}}$ satisfies $T^+(\lambda) \subseteq T$, then an optimal solution to $LR(T, \lambda)$ is also optimal to $LR(T_{\mathrm{all}}, \lambda)$. On the other hand, if there is an in-tree $\tau \in T_{\mathrm{all}}$ which is not included in T and has a positive relative cost $c_\tau(\lambda) > 0$, then an optimal solution $x(\lambda)$ to $LR(T, \lambda)$ cannot be optimal for $LR(T_{\mathrm{all}}, \lambda)$. It is therefore necessary to find a new in-tree $\tau \in T_{\mathrm{all}} \setminus T$ that satisfies

$$\sum_{i \in V} a_{i\tau} \lambda_i < 1. \tag{5}$$

The problem of finding such an in-tree (column) is generally called the *pricing problem*.

We showed in [12] that this pricing problem can be efficiently solved if λ is a feasible solution to the dual of the LP-relaxation problem of $P(T)$. To solve the pricing problem, the algorithm in our previous paper solves the problem of finding a new in-tree $\tau \in T_{\rm all} \setminus T$ that satisfies

$$\sum_{i \in V} a_{i\tau} \lambda_i = \min_{j \in T_{\rm all} \setminus T} \left(\sum_{i \in V} a_{ij} \lambda_i \right). \tag{6}$$

A nice feature of a dual feasible solution λ is that $c_j(\lambda) = 1 - \sum_{i \in V} a_{ij} \lambda_i \leq 0$ holds for all $j \in T$, and hence if an in-tree $\tau \in T_{\rm all}$ satisfying (5) is found, then we can conclude that τ is new, i.e., $\tau \notin T$. Then the problem of finding a new in-tree τ that satisfies (6) is equivalent to the problem of finding an in-tree τ that minimizes the left-hand side of (5) among all in-trees in $T_{\rm all}$. This problem is equivalent to the minimum weight rooted arborescence problem.

This problem takes as inputs a directed graph $G = (V, E)$, a root node $r \in V$ and an edge cost function $\phi : E \to \mathbb{R}$. The problem consists of finding a rooted arborescence with the minimum total edge cost. The problem can be solved in $O(|E||V|)$ time by Edmonds' algorithm [5]. Bock [2] and Chu and Liu [4] obtained similar results. Gabow et al. [7] presented the best results so far with an algorithm of time complexity $O(|E| + |V| \log |V|)$, which uses Fibonacci heap. We employed Edmonds' algorithm to solve this problem from the easiness of implementation.

When the pricing problem is solved for a Lagrangian multiplier vector, the nice feature of dual feasible solutions is not always satisfied, and the column generation method may not work; it may generate in-trees that are already in T. However, we observed through preliminary experiments that such duplicate generation is not frequent if good Lagrangian multiplier vectors are used. Based on this observation, we use Lagrangian multiplier vectors obtained by SubOpt.

To have higher probability of generating an in-tree not in T, our algorithm solves the pricing problem for more than one Lagrangian multiplier vector, and for this reason, we let the procedure SubOpt output ρ Lagrangian multiplier vectors that attain the best ρ upper bounds. Our column generation method solves the pricing problem for a Lagrangian multiplier vector $\lambda^{(k)}$ in the ascending order of k starting from $k = 1$ until a new in-tree $\tau \notin T$ is found or all $\lambda^{(1)}, \ldots, \lambda^{(\rho)}$ are checked. If a new in-tree is found, then it is added into the current set of in-trees T. On the other hand, if no new in-trees are found even after applying the column generation method to the ρ Lagrangian multiplier vectors, the entire procedure of generating in-trees stops.

3.4 Stopping criteria of the column generation method

In this subsection, we consider the stopping criteria of the column generation method. We introduce two stopping criteria and stop the algorithm when one of these criteria is satisfied.

The first one uses upper bounds of $\mathrm{OPT}_{P(T_{\rm all})}$. In our previous paper [12], we proposed a method that calculates an upper bound of $\mathrm{OPT}_{P(T_{\rm all})}$ from a given set of in-trees T and a nonnegative vector $\lambda \geq 0$. More precisely, this method creates a dual feasible solution of the LP-relaxation problem of $P(T_{\rm all})$. We observed through computational experiments that the method gives a tight upper bound if a good in-tree set T and an appropriate vector λ are given. We use this property as a stopping criterion of the algorithm. For the candidates of λ, we employed Lagrangian multiplier vectors obtained by SubOpt, and upper bounds of $P(T_{\rm all})$ are calculated in each iteration of the column generation method. Let UB^* be the best upper bound found by then during the iteration of our column generation algorithm. If T is not yet a good set of in-trees, UB^* is often updated in the following iterations. On the other hand, when T becomes a good set of in-trees (i.e, it includes most of valuable in-trees), UB^* is updated infrequently. Hence we stop the algorithm if UB^* is not updated in $|V|$ consecutive iterations.

The second stopping criterion is based on the overlapping of generated in-trees. When no new in-trees are found even after applying the column generation method to all ρ Lagrangian multiplier vectors obtained by SubOpt, we stop the algorithm (as stated in Section 3.3).

In the computational experiments in Section 4, we set the value of parameter ρ to 10. The value of parameter ρ has little influence on the performance of the algorithm as long as it is sufficiently large. Indeed, this value $\rho = 10$ was large enough in our experiments because with this value of ρ, the proposed algorithm never stopped with the second criterion.

3.5 Proposed algorithm to generate in-trees

The new algorithm to generate in-trees based on the column generation approach with the Lagrangian relaxation is formally described as Algorithm LRGenInTrees.

Algorithm 1 LRGenInTrees

Require: a graph $G = (V, E)$, a root node $r \in V$, tail and head consumption functions on edges $t : E \to \mathbb{R}_+$, $h : E \to \mathbb{R}_+$, node capacities $b_i \in \mathbb{R}_+$ for all $i \in V$, and a parameter ρ.
Ensure: a set of in-trees T.

1: Create the initial set T_0 of $|V|$ in-trees randomly. Set $T := T_0$, UB* $:= +\infty$, $\ell := 0$, $\lambda_i := 2$ for all $i \in V$ and $\pi := 2$.
2: Invoke PackInTrees and let LB be the obtained lower bound of $P(T)$.
3: Invoke SubOpt(LB, λ, π) to obtain $\lambda^{(1)}, \ldots, \lambda^{(\rho)}$ and $\pi^{(1)}, \ldots, \pi^{(\rho)}$, and set $\ell := \ell + 1$.
4: **for** $k = 1$ **to** ρ **do**
5: Calculate an upper bound UB of $\mathrm{OPT}_{P(T_{\mathrm{all}})}$ using the current in-tree set T and a vector $\lambda^{(k)}$ (by the method described in Section 3.4), and let UB* := UB and $\ell := 0$ if UB < UB*.
6: Solve the pricing problem for a vector $\lambda^{(k)}$ and let τ be the generated in-tree.
7: If $\tau \notin T$ holds, then set $T := T \cap \{\tau\}$, $\lambda := \lambda^{(k)}$ and $\pi := 4\pi^{(k)}$, and go to 10.
8: **end for**
9: Output the set of in-trees T and stop.
10: If $\ell = |V|$ holds, then go to 9.
11: If a hundred new in-trees are added into T after the last call to PackInTrees, then invoke PackInTrees and update LB.
12: Return to 3.

3.6 Method to obtain feasible solutions

We proposed an algorithm to generate a set of in-trees in the previous sections. To evaluate the performance of the proposed algorithm on the node capacitated in-tree packing problem, a method to obtain a feasible solution of $P(T_{\mathrm{all}})$ is necessary. Based on the second-phase algorithm proposed in [18], we devise a heuristic method called PackInTrees*.

Let T_0 be the initial set of in-trees and T_k be the set of in-trees T after the kth iteration of LRGenInTrees for $k = 1, \ldots, f$, where f is the number of in-trees generated by LRGenInTrees. The procedure PackInTrees* solves the LP-relaxation problems of $P(T_{f-\alpha}), \ldots, P(T_f)$ and obtains an optimal solution for each problem, where α is a parameter that we set $\alpha = 10$ in our computational experiments. For each optimal solution x^* of the LP-relaxation problems, a feasible solution of $P(T_{\mathrm{all}})$ is generated by rounding down every variable x^*_j of the solution, and then it is improved by applying PackInTrees, which is the greedy algorithm proposed in [12]. Among the α feasible solutions obtained by this procedure, PackInTrees* outputs the best one.

4 Computational Experiments

4.1 Experimental environment

We use instances consisting of randomly generated graphs in our experiment. We named them "rnd$n$-δ-b-(h, t or none)," where n is the number of nodes, δ is the edge density, b is the capacity of all $i \in V^-$ (where $V^- = V \setminus \{r\}$) and h, t or none shows which of head and tail consumptions is bigger (i.e., "h" implies that head consumptions are bigger than tail consumptions, "t" implies that tail consumptions are bigger than head consumptions, and no sign implies head and tail consumptions are chosen from the same range). We generated instances with $n = 100, 200$, $\delta = 5\%, 50\%$ and $b = 100000$ ($+\infty$ for the root node r). Instances of $\delta = 5\%$ (50%) are generated so that the out-degree of each node ranges from 4% (40%) to 6% (60%) of the number of nodes. Tail and head consumptions for "h" instances were randomly chosen from the integers in the intervals $[3, 5]$ and $[30, 50]$, respectively, for all edges not connected to the root. Similarly, those for "t" instances were randomly chosen from $[30, 50]$ and $[3, 5]$, and those for instances without "h" or "t" sign were randomly chosen from $[30, 50]$ and $[30, 50]$. The tail consumption of edges entering the root node r for all instances were randomly chosen from the integers in the interval $[300, 500]$ so that these edges cannot be used frequently.

Table 1: Computational results of the two algorithms

Instance name	$\|V^-\|$	$\|E\|$	$UB_{b.k.}$	Proposed Algorithm					Previous Algorithm [12]		
				$\|T\|$	UB*	Obj.	Gap (%)	Time (s)	Obj.	Gap (%)	Time (s)
rnd100-5-100000	100	473	1283	437	1285	1277	0.47	2.0	1095	14.65	3.0
rnd100-5-100000-h	100	473	2251	552	2254	2244	0.31	3.3	1836	18.44	4.4
rnd100-5-100000-t	100	473	2173	619	2281	2173	0.00	4.6	1903	12.43	5.0
rnd100-50-100000	100	4938	1498	510	1500	1491	0.47	3.9	1394	6.94	3.4
rnd100-50-100000-h	100	4938	2726	640	2730	2716	0.37	6.5	2640	3.15	5.0
rnd100-50-100000-t	100	4938	2701	439	2705	2687	0.52	3.9	1797	33.47	3.2
rnd200-5-100000	200	1970	1411	822	1413	1397	0.99	13.3	1147	18.71	45.8
rnd200-5-100000-h	200	1970	2602	930	2609	2584	0.69	17.3	1966	24.44	54.6
rnd200-5-100000-t	200	1970	2500	775	2615	2500	0.00	15.7	1879	24.84	34.9
rnd200-50-100000	200	20030	1569	871	1573	1554	0.96	25.9	1272	18.93	43.8
rnd200-50-100000-h	200	20030	2874	1412	2881	2851	0.80	62.3	2760	3.97	96.9
rnd200-50-100000-t	200	20030	2867	725	2878	2852	0.52	24.5	1872	34.71	30.1

The algorithms were coded in the C++ language and ran on a Dell PowerEdge T300 (Xeon X3363 2.83GHz, 6MB cache, 24GB memory), where the computation was executed on a single core. We used the primal simplex method in GLPK4.43[1] as LP solver.

4.2 Experimental results

Table 1 shows the results of the proposed algorithm for the problem instances explained in Section 4.1. It also shows the solutions obtained by the previous algorithm [12] for comparison purposes, where this algorithm was stopped when it generated the same number of in-trees as the new algorithm. The first three columns represent instance names, the number of nodes $|V^-|$ (without the root node), and the number of edges $|E|$. Column $UB_{b.k.}$ shows the best-known upper bounds of $OPT_{P(T_{all})}$ computed by the algorithm in [12], allowing long computation time of up to 170 minutes. The remaining columns include the experimental results of the proposed algorithm and the previous algorithm [12]. Column $|T|$ shows the number of in-trees generated by the algorithm LRGenInTrees, and column UB* shows the best upper bound of $OPT_{P(T_{all})}$ obtained by LRGenInTrees. The next three columns represent objective values, denoted "Obj.,", the gaps in % between $UB_{b.k.}$ and Obj., i.e., $((UB_{b.k.} - Obj.)/UB_{b.k.}) \times 100$, and computation times in seconds.

The results presented in Table 1 show that the proposed algorithm obtains better results than the previous algorithm. The proposed algorithm attained better objective values than the previous algorithm even though its computation time was shorter and the number of generated in-trees was the same. The gaps between upper bounds and objective values are quite small, and the proposed algorithm found exact optimal solutions for two instances.

5 Conclusions

In this paper, we proposed an algorithm to generate promising candidate in-trees for the node capacitated in-tree packing problem. This new algorithm generates a set of in-trees employing the subgradient method and the column generation method for the Lagrangian relaxation of the problem. We incorporated various ideas to speed up the algorithm, e.g., rules to decrease the number of in-trees used for the subgradient method, and upper and lower bounding techniques to reduce the number of times some Lagrangian multipliers are updated. The proposed algorithm obtained solutions whose gaps to the upper bounds are quite small, and was proved to be more efficient than the previous algorithm.

[1]GLPK - GNU Project - Free Software Foundation (FSF), `http://www.gnu.org/software/glpk/`, 28 February 2011.

References

[1] E. Balas and A. Ho. Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study. *Mathematical Programming Study*, 12:37–60, 1980.

[2] F. C. Bock. An algorithm to construct a minimum directed spanning tree in a directed network. In B. Avi-Itzak, editor, *Developments in Operations Research*, pages 29–44. Gordon and Breach, New York, 1971.

[3] G. Calinescu, S. Kapoor, A. Olshevsky, and A. Zelikovsky. Network lifetime and power assignment in ad-hoc wireless networks. In G. D. Battista and U. Zwick, editors, *Proceedings of the 11th European Symposium on Algorithms*, volume 2832 of *Lecture Notes in Computer Science*, pages 114–126. Springer, 2003.

[4] Y. Chu and T. Liu. On the shortest arborescence of a directed graph. *Science Sinica*, 14:1396–1400, 1965.

[5] J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards*, 71B:233–240, 1967.

[6] M. L. Fisher. The Lagrangian relaxation method for solving integer programming problems. *Management Science*, 27:1–18, 1981.

[7] H. N. Gabow, Z. Galil, T. Spencer, and R. E. Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica Archive*, 6:109–122, 1986.

[8] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan. An application-specific protocol architecture for wireless microsensor networks. *IEEE Transactions on Wireless Communications*, 1:660–670, 2002.

[9] M. Held and R. M. Karp. The traveling salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1:6–25, 1971.

[10] S. Imahori, Y. Miyamoto, H. Hashimoto, Y. Kobayashi, M. Sasaki, and M. Yagiura. The complexity of the node capacitated in-tree packing problem. *Networks*, To appear.

[11] M. Sasaki, T. Furuta, F. Ishizaki, and A. Suzuki. Multi-round topology construction in wireless sensor networks. In *Proceedings of the Asia-Pacific Symposium on Queueing Theory and Network Applications*, pages 377–384, 2007.

[12] Y. Tanaka, S. Imahori, and M. Yagiura. An lp-based heuristic algorithm for the node capacitated in-tree packing problem. *Compuer & Operations Research*, 39:637–646, 2012.