Faster Algorithms for Half-Integral T-Path Packing[*]

Maxim Babenko[1] **and Stepan Artamonov**[2]

1 **National Research University Higher School of Economics, Moscow, Russia**
`maxim.babenko@gmail.com`

2 **Moscow State University, Moscow, Russia**
`stiartamonov@gmail.com`

Abstract

Let $G = (V, E)$ be an undirected graph, $T \subseteq V$ be a set of *terminals*. Then a natural combinatorial problem consists in finding the maximum number of vertex-disjoint paths connecting distinct terminals. For this problem, a clever construction suggested by Gallai reduces it to computing a maximum non-bipartite matching and thus gives an $O\left(m\sqrt{n}\frac{\log n^2/m}{\log n}\right)$-time algorithm (hereinafter $n := |V|$, $m := |E|$).

Now let us consider the fractional relaxation, i.e. allow T-path packings with arbitrary nonnegative real weights. It is known that there always exists a half-integral solution, that is, one only needs to assign weights $0, \frac{1}{2}, 1$ to maximize the total weight of T-paths. It is also known that an optimum half-integral packing can be found in strongly-polynomial time but the actual time bounds are far from being satisfactory.

In this paper we present a novel algorithm that solves the half-integral problem within $O\left(m\sqrt{n}\frac{\log n^2/m}{\log n}\right)$ time, thus matching the complexities of integral and half-integral versions.

1998 ACM Subject Classification G.1.6 Optimization, G.2.2 Graph algorithms

Keywords and phrases graph algorithms, multiflows, path packings, matchings

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2017.8

1 Introduction

1.1 Basic Notation and Definitions

We shall use some standard graph-theoretic notation throughout the paper. For an undirected graph G, we denote its sets of vertices and edges by $V(G)$ and $E(G)$, resp. A similar notation is used for paths, trees, and etc. A subgraph of G induced by a subset $U \subseteq V(G)$ is denoted by $G[U]$. Let $P = (x_1, \ldots, x_p)$ be a path viewed as a sequence of vertices. By $P[x_i, x_j]$ we denote the part of the path from vertex x_i to vertex x_j. For paths P and Q with a common endpoint, we denote by $P \circ Q$ the concatenation of P and Q.

Let $G = (V, E)$ be an undirected graph, $T \subseteq V$ be a set of *terminals*. By a *T-path* we mean a simple s-t path P connecting two distinct terminals $s, t \in T$ such that all intermediate vertices of P are in $V - T$. A *T-path packing* is a weighted collection $\mathcal{P} = \{\alpha_1 \cdot P_1, \ldots, \alpha_k \cdot P_k\}$ where each P_i is a T-path and each α_i is a non-negative real *weight*. The *value* of a T-path packing $\mathcal{P}$ is $\operatorname{val}(\mathcal{P}) := \alpha_1 + \ldots + \alpha_k$. For $v \in V$, let $\mathcal{P}(v) := \sum (\alpha_i : v \in V(P_i))$ be the total weight of paths in $\mathcal{P}$ containing v. Clearly, $\sum_{t \in T} \mathcal{P}(t) = 2 \operatorname{val}(\mathcal{P})$.

[*] Supported by RFBR grant 16-01-00362.

Let $c\colon V \to \mathbb{Z}_+$ be vertex *capacities.* A T-path packing $\mathcal{P}$ is called *c-feasible* if $\mathcal{P}(v) \le c(v)$ for all $v \in V$. Then a natural combinatorial problem consists in finding a c-feasible T-path packing $\mathcal{P}$ of maximum value $\operatorname{val}(\mathcal{P})$.

1.2 Prior Art and Our Contribution

The above T-path packing problem has two flavors: one might either require all path weights to be integral or agree to deal with arbitrary nonnegative real weights.

One of the simplest cases is $c \equiv 1$. Then for the integral version of the problem a clever trick due to Gallai [10, Th. 74.1] reduces the path packing problem to finding a maximum matching in an auxiliary (possibly non-bipartite) graph. This implies an $O\left(m\sqrt{n}\frac{\log n^2/m}{\log n}\right)$-time algorithm [5] (hereinafter $n := |V|$, $m := |E|$). This connection to matchings, in fact, is not surprising since for $T = V$ the problem is equivalent to maximum matchings (either integral or fractional).

For general c, the integral version has a rich and sophisticated combinatorial structure, which was first studied by Mader [7]. Polynomial-time algorithms for this problem appear in, e.g., [8], [9]; however the actual complexity bounds are not satisfactory. In particular, strongly-polynomial approaches seem to be non-combinatorial and rely on general LP solvers.

For the fractional version of the problem, Pap [8] proved the primal *half-integrality*; i.e. there always exists a T-path packing with half-integral weights and maximum value. Due to this, we can double the capacities and search for an integral solution; hereinafter we mostly focus on integral packings.

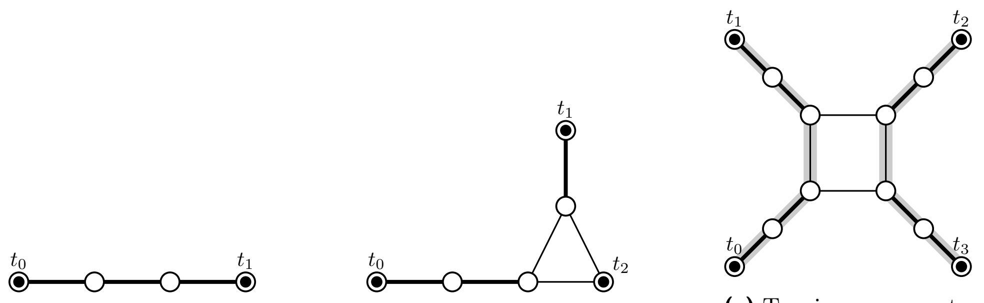
Thus the fractional relaxation of $c \equiv 1$ is exactly $c \equiv 2$. This case seems pretty natural and the existence of the elegant algorithm of Gallai suggests that some efficient direct combinatorial approach might exists here. Indeed, Babenko [1] proposed a combinatorial algorithm that computes a maximum path packing in $O(mn^2)$ time. The latter bound, however, is significantly worse than that of Gallai, which seems weird since typically fractional relaxations of integral problems are easier, both structurally and algorithmically.

Also for arbitrary even capacities, Babenko and Karzanov [3] proposed a weakly polynomial scaling algorithm with complexity $O(\Lambda(n,m,C)n^2 \log^2 n \log C)$, where C is the maximum vertex capacity and $\Lambda(n,m,C)$ denotes the complexity of finding a maximum integral flow in a digraph with n vertices, m edges, and integer edge capacities not exceeding C.

In this paper we present two new faster algorithms for the case $c \equiv 2$ (or, equivalently, the fractional version of the Gallai's problem). The first algorithm employs a simple labeling technique and runs in $O(mn)$ time. The second algorithm is more involved and runs in $O\left(m\sqrt{n}\frac{\log n^2/m}{\log n}\right)$ time. It relies on the Gallai's approach to construct the initial approximation (a packing with all path weights equal to 2) and then turns it into an optimal solution in linear time; this is where the previous labeling technique appears useful. Note that a conceptually similar approach was employed for computing maximum triangle-free 2-matchings [2].

The rest of the paper is organized as follows. In Section 2 we introduce some basic facts about T-path packings and the Edmonds–Gallai decomposition. Section 3 and Section 4 present the descriptions of $O(mn)$-time and $O(m\sqrt{n}\log\frac{n^2}{m})$-time algorithms, resp. Finally in Section 5 we conclude.

Note that due to the lack of space some proofs are moved into Appendix.



(a) A double path. **(b)** An odd star. **(c)** Turning an even star into double paths.

Figure 1 Elements of a canonical path packing. Terminal vertices are marked with dots.

2 Preliminaries

2.1 Canonical Decomposition

A 2-feasible T-path packing $\mathcal{P}$ is called *canonical* if it can be represented as a vertex-disjoint union of *elements* of two types: *double paths* and *odd stars*. Both of our algorithms will produce canonical packings.

A *double path* (Fig. 1(a)) is a collection $\mathcal{P}' = \{2 \cdot P\}$ consisting of a T-path P that is taken twice. A *star* (Fig. 1(b)) is a collection $\mathcal{P}'' = \{P_0, \dots, P_{k-1}\}$ of T-paths that form a star-like shape. The formal definition is rather technical and proceeds as follows. There must be a sequence of distinct terminals $t_0, \dots, t_{k-1}$ such that P_i connects t_i and t_{i+1} ($i = 0, \dots, k-1$ and indices are taken modulo k). For each i, P_i and P_{i+1} share a common path incident to t_{i+1}. Such common paths are called *legs*. If a vertex v belongs to both $P_i \in \mathcal{P}''$ and $P_j \in \mathcal{P}''$ then $|i - j| = 1$ and v should belong to the corresponding leg. Vertices covered by a single path (resp. two paths) in $\mathcal{P}''$ are called *inner* (resp. *leg vertices*). Each vertex belonging to a leg is covered by exactly two paths in $\mathcal{P}''$. Edges that are traversed by exactly one P_i in $\mathcal{P}''$ form a simple cycle called the *inner cycle* of the star. A star is called *odd* (resp. *even*) if k is odd (resp. even). Note that each even star can be easily turned into a number of double paths as shown in Fig. 1(c); this preserves the packing value.

In both algorithms we only consider *canonical* path packings. We also stress that canonical elements $\mathcal{P}_1, \dots, \mathcal{P}_l$ that form $\mathcal{P}$ are vertex-disjoint, i.e. each $v \in V$ is covered by at most one of $\mathcal{P}_1, \dots, \mathcal{P}_l$. Clearly this is only important for odd stars since overlapping a double path with any other component violates vertex capacities.

2.2 Min-Max Relation

▶ **Lemma 1.** *Let $\mathcal{P}$ be a 2-feasible T-path packing, U be an arbitrary subset of V. Then*

$$\mathrm{val}(\mathcal{P}) \le |T| + |U \cap T| + 2|U \setminus T| - \mathrm{ot}(G - U), \tag{1}$$

where $G - U$ stands for the graph obtained from G by removing vertices in U, and for a subgraph H of G, $\mathrm{ot}(H)$ denotes the number of connected components of H containing exactly one terminal.

The $O(mn)$ algorithm will provide both a T-path packing $\mathcal{P}$ and a set U turning (1) into equality, thus proving the following min-max relation:

▶ **Theorem 2.** *For 2-feasible T-packings $\mathcal{P}$,*

$$\max_{\mathcal{P}} \operatorname{val}(\mathcal{P}) = \min_{U \subseteq V} |T| + |U \cap T| + 2|U \setminus T| - \operatorname{ot}(G - U). \tag{2}$$

2.3 Edmonds–Gallai Decomposition

For the faster algorithm we will need several standard facts concerning maximum matchings (see [6, Ch. 3] for a survey). For an undirected graph $G = (V, E)$, let $\nu(G)$ denote the size of a maximum matching in G and $\operatorname{odd}(G)$ be the number of connected components of G with an odd number of vertices. A graph G is *factor-critical* if for any $v \in V$, $G - v$ admits a perfect matching. For a matching M, a vertex v is called *exposed* if $v \notin V(M)$.

▶ **Theorem 3** (Tutte–Berge, [6, Ch. 3, Th. 3.1.14])**.** $\nu(G) = \min_{U \subseteq V} \frac{1}{2} (|V| + |U| - \operatorname{odd}(G - U))$.

▶ **Theorem 4** (Edmonds–Gallai, [6, Ch. 3, Th. 3.2.1])**.** *For an undirected graph G, let*

$$\begin{aligned} D &:= \{v \in V \mid \textit{there exists a maximum matching exposing } v\}, \\ A &:= \{v \in V - D \mid v \textit{ is a neighbor of } D\}, \\ C &:= V - (A \cup D). \end{aligned}$$

Then $U := A$ achieves the minimum in the Tutte–Berge formula, and D is the union of the odd connected components of $G[V - A]$. Every connected component of $G[D]$ is factor-critical. Any maximum matching in G induces a perfect matching in $G[C]$ and a matching in $G[V - C]$ that matches all vertices of A to distinct connected components of $G[D]$.

Note that once a maximum matching M in G is found, an Edmonds–Gallai decomposition of G can be constructed in linear time by running a search for an M-augmenting path. Most algorithms that find M yield this decomposition as a by-product.

3 $O(mn)$-time algorithm

The algorithm will be iterative. Each iteration gets the current T-path packing $\mathcal{P}$, which is represented as a set of its canonical elements, and aims to increase $\operatorname{val}(\mathcal{P})$.

Each iteration will assign *labels* from the set $\{*\} \cup T$ to vertices of G, i.e. maintain a partial map $l \colon V \to \{*\} \cup T$. During one iteration, a once labeled vertex never changes its label. Here $*$ is a special label, the remaining labels correspond to terminals.

A vertex v is called *free* if $\mathcal{P}(v) = 0$, and *covered* otherwise. Similarly an edge e is called *free* if there does not exist $P \in \mathcal{P}$ such that $e \in E(P)$. Note that since $\mathcal{P}$ is canonical, if a terminal t is not free then $\mathcal{P}(t) = 2$. For a vertex $v \in V - T$ covered by $\mathcal{P}$, $\mathcal{P}(v) = 1$ if it is an inner vertex of some odd star and $\mathcal{P}(v) = 2$ otherwise.

The outline of the algorithm is the following. We initially set $l(t) := t$ for each free $t \in T$; other vertices are unlabeled. Then, if for the current $\mathcal{P}$ there is no free terminal then $\mathcal{P}$ is obviously maximum. Otherwise, we enumerate all free terminals and run a certain search procedure from each such terminal.

The search assigns labels to the vertices it visits; see Subsection 3.1 for the details. It builds a certain tree; let F_t be this tree for the current terminal t. During this search we may say that we obtained a *breakthrough*, which means that we can modify $\mathcal{P}$ to increase its value in $O(m + n)$ time. The detailed explanation of a breakthrough is given in Subsection 3.2. When the search at the current free terminal finishes and yields no breakthrough, we will ignore this t with its tree F_t and proceed with searches from other free terminals. If no free terminals are left then it will be shown in Subsection 3.3 that the current $\mathcal{P}$ is maximum.

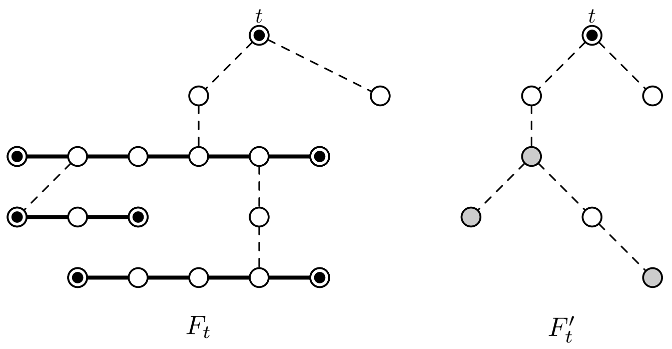


Figure 2 Example of F_t and F'_t. Grayed vertices correspond to contracted double paths.

3.1 Labeling Procedure

Let t be the current free terminal (see above). To start the search from t, we mark t as *active*. We will maintain the property that all active vertices are labeled. Then we run the following labeling procedure.

Let v be an active vertex, $l(v) = \alpha$. The algorithm scans all free edges incident to v. Let $e = \{v, w\}$ be some free edge. We perform a case-splitting as follows:

(A1) $l(w)$ is defined.

  (A1-1) $l(w) = \alpha$ or $l(w) = *$. Then edge e is ignored.

  (A1-2) $l(w) \neq \alpha$ and $l(w) \neq *$. Then we get a breakthrough.

(A2) $l(w)$ is not defined and w is a free vertex. Then we set $l(w) := \alpha$, make w active and add w together with e to F_t.

(A3) $l(w)$ is not defined and w is covered by a double path $P = (x_0 = \beta, \ldots, x_k = \gamma)$ (viewed as a sequence of vertices). Here $\beta, \gamma \in T$. Let $w = x_j$. Then we set $l(x_i) := \beta$ for all $i < j$, $l(x_i) := \gamma$ for all $i > j$, and $l(x_j) := *$, mark all these labeled vertices except for x_j as active, and add the whole path P together with the edge e to the tree F_t.

(A4) $l(w)$ is not defined and w belongs to an odd star S. Then we get a breakthrough.

When all free edges incident to v are examined, v becomes inactive.

Note that once the algorithm reaches a double path P in the current $\mathcal{P}$, it immediately labels all of its vertices. In particular, $*$-vertices are only present on reached paths P, and for each such path there is a unique such vertex; note that this vertex might be a terminal.

Consider the tree F_t rooted at t and contract each double path P that belongs to F_t into a new vertex $v(P)$. Then we obtain a tree F'_t where each non-root vertex corresponds to either a double path or a free non-terminal vertex; see Fig. 2.

Note that by choosing the order in which we examine active vertices and scan edges incident to them, we may assume that this tree F'_t is a DFS-tree, that is, if there is an edge $\{x, y\}$ outside the tree between two distinct vertices x, y of the tree then x is an ancestor of y or vice versa. To achieve this property, we run the usual DFS from the terminal t; the only difference with the standard DFS is that when examining an edge e triggers case (A3), all vertices that become active are added to the DFS stack.

We note that all vertices that were labeled during the current search from t belong to F_t, and if for some edge e case (A1-2) applies then it either connects F_t with a free terminal distinct from t or the image of this edge in F'_t connects two vertices which either coincide or one is an ancestor of the other; note that loops are also possible here as there might be an edge outside F_t connecting two vertices that belong to one double path.

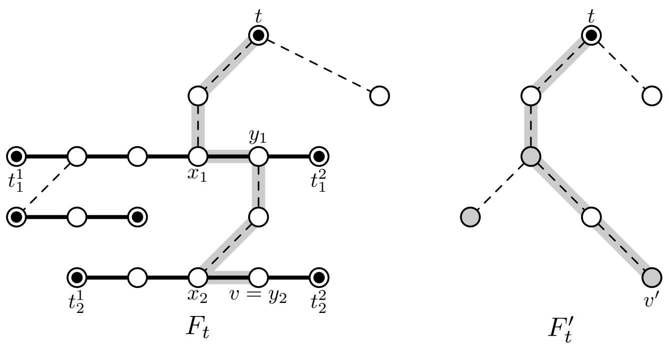


Figure 3 An example of trees F_t and F'_t. Grayed edges indicate paths Q in F_t and Q' in F'_t.

3.2 Breakthrough

Suppose that at some point during an iteration we obtained a breakthrough. We now prove that in this case we can augment the current T-path packing $\mathcal{P}$. Let t be the free terminal that is currently being evaluated, F_t be the tree that grows from it, F'_t be the tree obtained from F_t after contracting each double path. For trees F'_t containing at least one contracted vertex, we define the following operation that alters $\mathcal{P}$ preserving its size but making the terminal t covered and exposing some of the earlier covered terminals.

Let v be some vertex in F_t, v' be its image in F'_t, Q be the (unique) t–v path in F_t, and Q' be the image of Q in F'_t. Let $P_1, P_2, \ldots, P_k$ be the sequence of double paths which appear as contracted vertices $p_1, p_2, \ldots, p_k$ in Q' when travelling from t to v', and denote by t_i^1 and t_i^2 the endpoints of P_i. Path Q intersects each P_i by a segment; let x_i and y_i be the first and the last (resp.) vertices of this segment (assuming that Q is directed from t to v). Note that vertices x_i have label $*$. Also note that $x_i \neq y_i$ as we do not scan edges from x_i. See Fig. 3 for an example.

Note that for each P_i we may exchange t_i^1 with t_i^2; we shall be using this idea extensively to simplify the case-splitting.

The cornerstone is the following switching routine. Fix $i \in \{1, \ldots, k\}$ and an arbitrary endpoint of P_i, say t_i^2. Then we can alter $\mathcal{P}$ (namely, change $P_1, \ldots, P_i$) so that t becomes covered, all vertices between t_i^2 and x_i (excluding x_i) in P_i are made free, and the rest of the terminals do not change their covering status. We call this $\textsc{expose}(t_i^2)$.

To prove this, we apply induction on i. If $i = 1$ then we replace P_1 with $P_1[t_1^1, x_1] \circ Q[x_1, t]$. Now let $i > 1$. Exchanging t_{i-1}^1, t_{i-1}^2 if needed, we may assume that $P_{i-1}[t_{i-1}^1, y_{i-1}]$ contains x_{i-1}. We first invoke $\textsc{expose}(t_{i-1}^2)$ to update $P_1, \ldots, P_{i-1}$ and make t_{i-1}^2 free. Then we replace P_i with $P_i[t_i^1, x_i] \circ Q[x_i, y_{i-1}] \circ P_{i-1}[y_{i-1}, t_{i-1}^2]$. It is straightforward to see that the resulting double paths do not intersect and cover the desired set of vertices.

Now, recall that we can get the breakthrough in cases (A1-2) and (A4). Let v be the current active vertex and $e = \{v, w\}$ be the edge triggering the breakthrough. We keep the above notation for Q, P_i, t_i^1, t_i^2, x_i, y_i. Exchanging t_i^1, t_i^2 if needed, we may assume that $P_i[t_i^1, y_i]$ contains x_i for all $i \in \{1, \ldots, k\}$.

3.2.1 Case A1-2

Note that w can either be a free terminal or a vertex in F_t since the only other labeled vertices are the ones that belong to the trees $F_{t'}$ for free terminals t' that were already scanned, but in that case e should have already been visited while examining w and would have either led to a breakthrough or to vertex v being added to $F_{t'}$, which did not happen.

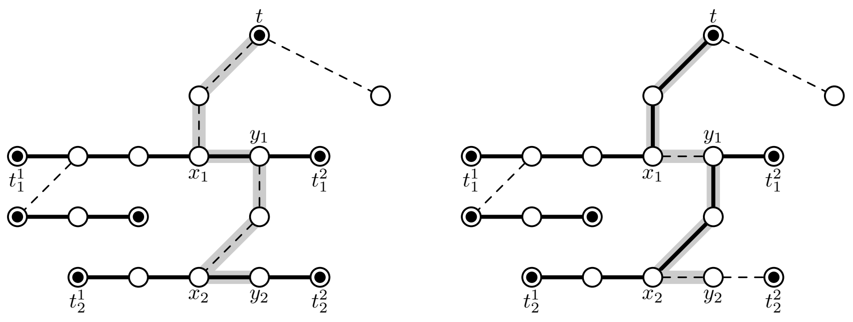


Figure 4 An example of EXPOSE(t_2^2) call. Grayed edges indicate paths Q in F_t and Q' in F'_t.

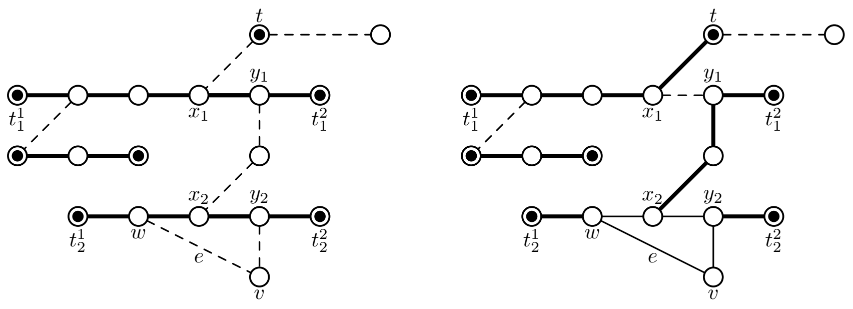


Figure 5 Subcase 2a ($k > 1$).

We thus consider these two cases.

1. w is a free terminal (other than t). If $k = 0$ then we just add $Q \circ e$ as a new double path to $\mathcal{P}$. Otherwise, we first apply EXPOSE(t_k^2) and then add $P_k[t_k^2, y_k] \circ Q[y_k, v] \circ e$ as a new double path.
2. w belongs to F_t. As mentioned above, F'_t is a DFS-tree, so the image of e in F'_t is either a loop or connects a vertex with its ancestor; it might be a loop if both v and w belong to the same double path P_k (in this case $P_k[v, w]$ contains x_k as $l(v) \neq l(w)$). We shall apply EXPOSE to alter some double paths in $\mathcal{P}$ and turn the rest into an odd star. Namely, we have the following possible subcases for w:
   a. w belongs to P_k. We first note that y_k and v have the same label. Then, note that w belongs to $P_k[t_k^1, x_k]$ since otherwise w and y_k and thus v would have had the same label. If $k = 1$ then we form an odd star with legs $Q[t, x_1]$, $P_1[t_1^1, w]$, $P_1[t_1^2, y_1]$ and the inner cycle $P_1[w, y_1] \circ Q[y_1, v] \circ e$. Otherwise we apply EXPOSE(t_{k-1}^2) and create a similar odd star with legs $P_{k-1}[t_{k-1}^2, y_{k-1}] \circ Q[y_{k-1}, x_k]$, $P_k[t_k^1, w]$, $P_k[t_k^2, y_k]$ and the inner cycle $P_k[w, y_k] \circ Q[y_k, v] \circ e$, see Fig. 5.
   b. w belongs to P_i for $i < k$ and x_i does not belong to $P_i[w, y_i]$. We apply EXPOSE(t_i^2) and create an odd star S with the inner cycle $P_i[w, y_i] \circ Q[y_i, v] \circ e$. The legs of S are formed as follows. For endpoint t_j^q of P_j for $j \in \{i+1, \ldots, k\}$, $q \in \{1, 2\}$, we take the part of P_j from t_j^q to the closest of x_j and y_j. For endpoint t_i^2 of P_i, we take the part of P_i from t_i^2 to the closest of y_i and w, see Fig. 6.
   c. w belongs to P_i for $i < k$ and x_i belongs to $P_i[w, y_i]$. If $i = 1$ then we replace all $P_1, \ldots, P_k$ with an odd star. The inner cycle, the legs for terminals t_j^q for $j \in \{1, \ldots, k\}$, $q \in \{1, 2\}$, and the leg for terminal t are constructed similarly to the previous case; see Fig. 7. Otherwise we apply EXPOSE(t_{i-1}^2) and then again form a similar odd star using $P_{i-1}[t_{i-1}^2, y_{i-1}] \circ Q[y_{i-1}, x_i]$ as its leg from t_{i-1}^2.

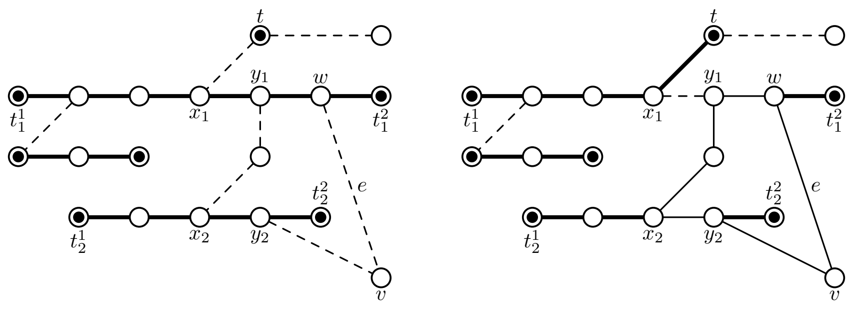


Figure 6 Subcase 2b.

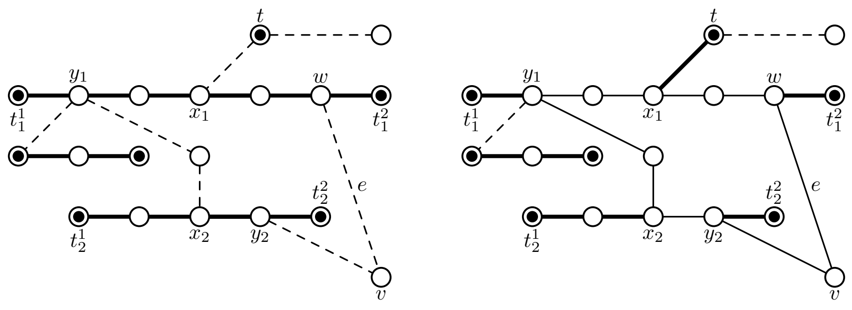


Figure 7 Subcase 2c ($i = 1$).

d. w is a free vertex belonging to F_t. We follow from w to t in F_t until either reaching P_i for some $i \in \{1, \ldots, k\}$ or the root t. In the first subcase the vertex where we stop is exactly y_i. Moreover, $i \neq k$ as $l(v) \neq l(w)$. We apply EXPOSE(t_i^2) and construct an odd star. In the second subcase no EXPOSE call is needed and we just construct an odd star out of $P_1, \ldots, P_k$. In both subcases the inner cycle of this star is $Q[w, v] \circ e$. We omit the details since they are straightforward and very similar to the above.

3.2.2 Case A4

Here we have two cases: either w is an inner vertex of a star S or w belongs to a leg of S.

1. w is an inner vertex of S. Then we apply EXPOSE(t_k^2), attach $P_k[t_k^2, y_k] \circ Q[y_k, v] \circ e$ as a new leg to S thus turning S into an even star, and finally dissolve S into a collection of double paths.
2. w belongs to a leg L of S ending in a terminal t_L. In this case we similarly apply EXPOSE(t_k^2), then remove leg L from S making S an even star, dissolve S into a collection of double paths, and finally add double path $P_k[t_k^2, y_k] \circ Q[y_k, v] \circ e \circ L[w, t_L]$.

Note that in each of the mentioned cases after the transformation all terminals that were covered by $\mathcal{P}$ remain covered by the new path packing, plus t becomes covered, which means that the value of path packing increases by 2, as desired.

3.3 Proof of Maximality

Suppose that we visited all free terminals and did not find a breakthrough. This means that there is no free edge between two labeled vertices with different non-$*$ labels or a non-$*$ labeled and an unlabeled vertex. This in turn means that for each free terminal t and its tree F_t (in which t is the unique free terminal), the only edges that connect F_t with vertices outside F_t have label $*$ on at least one of its ends. Define $U := \{v \mid l(v) = *\}$ and look at the connected components of $G - U$. Let $U' := U \cap T$, $U'' := U \setminus T$. Consider a terminal $s \in T \setminus U$ and suppose that s belongs to some tree F_t. Then in the connected component of $G - U$ containing s there is just one terminal (namely s itself). Indeed, we prove that this component is $G[S]$ for $S := \{v \mid l(v) = s\}$. Suppose first that s is covered by a double path P, and $x \in V(P)$ is the (unique) vertex with $l(x) = *$. In this case $S = V(P[s, x] - x)$, and s is obviously connected to all vertices in S. If there would be an edge connecting S with the rest of $G - U$, it would have to go either to a different tree, or to a vertex with a different label, or to an unlabeled vertex; all these cases are impossible. Thus S is the connected component of $G - U$ containing s and it indeed has just one terminal. Similar reasoning applies if s is a free terminal in F_t, meaning that $s = t$; here S consists of all vertices in F_t for which the path to t in F_t consists solely of free vertices.

Now suppose that we have k free terminals $t_1, \ldots, t_k$. Fix $i \in \{1, \ldots, k\}$ and consider F_{t_i}. Let u'_i (resp. u''_i) be the number of terminal (resp. non-terminal) vertices in this tree that belong to U. Then in F_{t_i} we have one free terminal t_i and $u'_i + 2u''_i$ covered terminals that are the only terminals in their connected components of $G - U$ (see above). Summing over all i, we get $\mathrm{ot}(G - U) \geq k + |U'| + 2|U''|$. We then obtain

$$\mathrm{val}(P) = |T| - k \geq |T| + |U'| + 2|U''| - \mathrm{ot}(G - U) = |T| + |U \cap T| + 2|U \setminus T| - \mathrm{ot}(G - U).$$

This together with (1) proves the min-max relation Theorem 2 and the fact that the current $\mathcal{P}$ is optimal.

3.4 Complexity Analysis

▶ **Lemma 5.** *Each iteration can be implemented to run in $O(m + n)$ time.*

Since each iteration either results in a breakthrough which increases the value of $\mathcal{P}$ or reports that the current $\mathcal{P}$ is maximum, the number of iterations is $O(n)$, thus the total time complexity is $O(mn)$.

4 $O\left(m\sqrt{n}\frac{\log n^2/m}{\log n}\right)$-time algorithm

Following Gallai [10, Th. 73.1], we build an auxiliary graph $\widehat{G} = (\widehat{V}, \widehat{E})$ from G as follows. For each $v \in V - T$, we make a second copy v' of v. Vertices v and v' are called *mates*. For each edge $\{v, u\}$, we add edge $\{v', u\}$ and also $\{v', u'\}$ if $u \in V - T$. We also add edges $\{v, v'\}$ for each $v \in V - T$.

For each subset $U \subseteq V$, denote by $\widehat{U} \subseteq \widehat{V}$ its *image* in $\widehat{G}$ that consists of all $t \in T \cap U$ and includes both v and v' for all $v \in U \setminus T$. We call a subset $W \subseteq V(\widehat{G})$ *symmetric* if v belongs to W iff v' belongs to W, for each $v \in V - T$. Clearly, symmetric subsets of $\widehat{V}$ are exactly the images of subsets of V.

The outline of the algorithm is the following. First, we construct a maximum matching in $\widehat{G}$ with some special properties. Gallai [10, Th. 73.1] showed that a maximum matching in $\widehat{G}$ provides a maximum packing of T-paths subject to unit capacities; in particular, the number of vertices not covered by this matching in $\widehat{G}$ is equal to the number of terminals that are not covered by a maximum T-path packing.

Recall that we are interested in the case $c \equiv 2$, so after doubling path weights Gallai's packing is not necessarily a maximum one. However this matching gives us an approximation for the optimal solution.

The next step is to augment this approximation using the $O(mn)$ algorithm. We will need to run this algorithm in some subgraphs of G, however in each subgraph we will only need to run one iteration of the algorithm, thus leading to linear time complexity for this step.

4.1 Algorithm Description

First, we compute an arbitrary maximum matching in $\widehat{G}$ and an Edmonds–Gallai decomposition $(\widehat{D}, \widehat{A}, \widehat{C})$ of $\widehat{G}$. Let $\widehat{D_1}, \ldots, \widehat{D_k} \subseteq \widehat{D}$ be the factor-critical connected components of $\widehat{G} - \widehat{A}$. The following lemma justifies the notation and proves that these sets are images of certain sets $D, D_i, A, C \subseteq V$.

▶ **Lemma 6.** *$\widehat{D}, \widehat{D_i}, \widehat{A}, \widehat{C}$ are symmetric subsets of $\widehat{V}$.*

Each component $\widehat{D_i}$ consists of an odd number of vertices and thus contains at least one terminal. A component $\widehat{D_i}$ is called *bad* if D_i contains exactly one terminal, and *good* otherwise. Typically $\widehat{G}$ admits more than just one maximum matching. Our goal is to choose this matching in a proper way, i.e. to cover as many bad components as possible; cf. [2]. To this aim, we first build an auxiliary bipartite graph H with vertices in the *upper* part corresponding to contracted components $\widehat{D_i}$, and $\widehat{A}$ as the the *lower* part. Edges of H correspond to edges of $\widehat{G}$ between $\widehat{A}$ and $\widehat{D}$. Let H' be formed from H by dropping all vertices in the upper part corresponding to good components. We construct a maximum (bipartite) matching M' in H' and then augment it to a maximum matching M in H.

It is widely known that augmenting a matching only increases the set of covered vertices. Also the resulting M is a maximum matching in H and by Theorem 4 this M covers all vertices in $\widehat{A}$.

We convert M into a matching $\widehat{M}$ in $\widehat{G}$ by taking the preimages w.r.t. contractions. Now, for each component $\widehat{D_i}$ covered by M in H, we extend $\widehat{M}$ to cover all vertices of $\widehat{D_i}$ as it is factor-critical. For each good component $\widehat{D_j}$ not covered by M in H, we extend $\widehat{M}$ to cover all vertices of $\widehat{D_j}$ expect for one arbitrarily chosen terminal t_j. We also extend $\widehat{M}$ with some perfect matching in $\widehat{C}$.

Let $\widehat{D'}$ be the union of bad components that are not covered by M in H, T' be the set of terminals belonging to components in $\widehat{D'}$ (one per component). Let T'' be the set of terminals belonging to good components that are not covered by $\widehat{M}$ (again, one per component).

We now turn this matching $\widehat{M}$ into a path packing (subject to $c \equiv 2$) that covers terminals $T - (T' \cup T'')$ (twice); the reader may refer to [10, Th. 73.1]. Let us temporarily remove $\widehat{D'} \cup T''$ from $\widehat{G}$ and the corresponding $D \cup T''$ from G. Then $\widehat{M}$ is a perfect matching. Let N be the set of edges $\{v, v'\} \in \widehat{E}$ for $v \in V - T$. It is straightforward to see that the union $\widehat{M} \cup N$ consists of the following vertex-disjoint components: edges $\{v, v'\}$, simple cycles avoiding terminals, and paths between terminals. After shrinking edges $\{v, v'\}$ in these paths we obtain a collection of disjoint T-paths in G covering all terminals covered by $\widehat{M}$ in $\widehat{G}$. We assign weight 2 to these paths, i.e. regard them as double paths. Let $\mathcal{P}$ be the resulting path packing in G.

The last step is to deal with the terminals T''. Let $\widehat{D_i}$ be a good component not covered by M in H. Then $\widehat{M} \cap E(\widehat{D_i})$ is a nearly-perfect matching in $\widehat{D_i}$, i.e. it exposes exactly one (terminal) vertex $t_i \in D_i \cap T$. Note that the current $\mathcal{P}$ covers all terminals in D_i expect for t_i. Our goal is to adjust $\mathcal{P}$ to cover t_i as well. To accomplish this, we run a search procedure from Section 3 for $G[D_i]$ (using t_i as the starting terminal). This iteration may either find a breakthrough or report that the current packing is maximum. We prove that the latter is not possible, completing the description of the algorithm.

▶ **Lemma 7.** *$\mathcal{P}$ can always be augmented in each good $\widehat{D_i}$.*

Proof. Suppose that the augmentation procedure did not find a breakthrough. Let F_t be the tree obtained by the algorithm, $P_1, \dots, P_k$ be the set of double paths belonging to F_t. Note that $k > 0$, since otherwise it would mean that t is not connected to the other terminals in D_i, but it must be connected as $G[D_i]$ is factor-critical. Let $U := \{v \mid l(v) = *\}$, $u' = |U \cap T|$, $u'' = |U \setminus T|$. Note that due to the above observation U is not empty. Removing U from D_i gives us at least $u' + 2u'' + 1$ connected components containing exactly one terminal (see Subsection 3.3). Let us now look at the images of these sets in $\widehat{G}$. Note that $|\widehat{U}| = u' + 2u''$. Graph $\widehat{G}[\widehat{D_i} - \widehat{U}]$ has at least $u' + 2u'' + 1$ connected components of odd size. Indeed, consider the subgraph of $\widehat{G}$ induced by $\widehat{V(P_i)}$ for some double path P_i. Let x_i be the unique vertex of P_i with $l(x_i) = *$. If $x_i \in T$ then x_i does not have a mate in $\widehat{D_i}$, and removing it gives one connected component which is symmetric and contains a single terminal (the other endpoint of P_i). If $x_i \notin T$ then removing $\{x_i, x_i'\}$ gives rise to two connected components, each of which is symmetric and contains a single terminal (the respective endpoint of P_i).

Now, recall again that $\widehat{G}[\widehat{D_i}]$ is factor-critical. We pick and remove an arbitrary vertex in $\widehat{U}$; the resulting graph must contain a perfect matching. However, if we proceed and remove the remaining $u' + 2u'' - 1$ vertices of $\widehat{U}$ then the resulting graph would contain $u' + 2u'' + 1$ connected components of odd size, which is a contradiction to Theorem 3. ◀

4.2 Proof of Maximality

▶ **Theorem 8.** *The above algorithm indeed produces a maximum packing $\mathcal{P}$.*

Proof. Consider the maximum matching M' in H' constructed by the algorithm and let L be the corresponding minimum vertex cover in H'. It is straightforward to see that the set $L \cap \widehat{A}$ is symmetric, i.e. is equal to $\widehat{L_A}$ for some $L_A \subseteq A$. Indeed, suppose that some vertex $v \in V - T$ belongs to $L \cap \widehat{A}$ but its mate v' does not. This means that for every edge $e = \{v', w\} \in E(H')$, w should belong to L. But then v can be removed from this vertex cover, contradicting its minimality. The case $v' \in L \cap \widehat{A}$, $v \notin L \cap \widehat{A}$ is symmetric.

Define $L_D := L \setminus \widehat{L_A}$. Let β be the number of bad components $\widehat{D_i}$. Then $\widehat{G} - \widehat{L_A}$ contains at least $\beta - |L_D|$ connected components with just one terminal (each bad component $\widehat{D_i}$ not covered by L can only be connected to $\widehat{L_A}$ and hence separates when $\widehat{L_A}$ gets removed). Therefore $\operatorname{ot}(G - L_A) \geq \beta - |L_D|$.

Note that in the resulting path packing $\mathcal{P}$ exactly $\beta - |M'|$ terminals are not covered. Also $|M'| = |L_D| + |\widehat{L_A}|$ by the max-matching min-cover duality. Then $\operatorname{val}(\mathcal{P}) = |T| - (\beta - |M'|) = |T| - (\beta - |\widehat{L_A}| - |L_D|) \geq |T| + |\widehat{L_A}| - \operatorname{ot}(G - L_A) = |T| + |L_A \cap T| + 2|L_A \setminus T| - \operatorname{ot}(G - L_A)$, which by Theorem 2 implies that $\mathcal{P}$ is maximum. ◀

4.3 Complexity Analysis

Using the algorithm from [5], a maximum matching in $\widehat{G}$ and an Edmonds-Gallai decomposition for $\widehat{G}$ can be constructed in $O\left(m\sqrt{n}\frac{\log n^2/m}{\log n}\right)$ time. The algorithm from [4], which has the same time bound, can be applied to construct $\widehat{M}$ in the bipartite H, and finally again the matching algorithm for general graphs from [5] can be used to augment $\widehat{M}$ in each $\widehat{D_i}$ which are not covered by M in H.

Turning the union $\widehat{M} \cup N$ into the desired collection of T-paths can obviously be done in $O(m + n)$ time. As for the last step that deals with good components not covered by M

in H, in each such component we need to run exactly one iteration of the first algorithm, thus for a component D_i it takes $O(|V(G[D_i])| + |E(G[D_i])|)$ time. Summing over all such components we get $O(m+n)$ time; hence the total time complexity is $O\left(m\sqrt{n}\frac{\log n^2/m}{\log n}\right)$.

5 Conclusions

We have presented an efficient algorithm for constructing maximum integral packings of T-path subject to capacities $c \equiv 2$ (or, equivalently, fractional packings subject to $c \equiv 1$). A natural question is if some similar approach could handle the case of arbitrary even capacities and give rise to, e.g., a pseudo-polynomial or a even strongly-polynomial algorithm.

Note that the standard vertex splitting (which replaces each vertex v with capacity $c(v)$ by $c(v)/2$ copies $v_1, \ldots, v_{c(v)/2}$ with capacity 2 each and adjusts the edges appropriately) does not help, since one needs to prevent paths in $\mathcal{P}$ from having (distinct) endpoints to be split-mates that correspond to a single $t \in T$ — a new restriction, which is non-local and is seemingly inexpressible in terms of the Gallai's auxiliary graph.

We suspect that there is a way to extend the labeling approach presented in Section 3 in the needed way and hope to deal with this topic in a subsequent paper.

References

1 M. A. Babenko. A fast algorithm for the path 2-packing problem. *Theory of Computing Systems*, 46(1):59, 2008. URL: `http://dx.doi.org/10.1007/s00224-008-9141-y`, `doi:10.1007/s00224-008-9141-y`.

2 M. A. Babenko, A. Gusakov, and I. P. Razenshteyn. Triangle-free 2-matchings revisited. *Discrete Math., Alg. and Appl.*, 2:643–654, 2010.

3 M. A. Babenko and A. V. Karzanov. Min-cost multiflows in node-capacitated undirected networks. *Journal of Combinatorial Optimization*, 24(3):202–228, 2012. URL: `http://dx.doi.org/10.1007/s10878-011-9377-3`, `doi:10.1007/s10878-011-9377-3`.

4 T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. *J. Comput. Syst. Sci.*, 51:261–272, October 1995.

5 A. V. Goldberg and A. V. Karzanov. Maximum skew-symmetric flows and matchings. *Mathematical Programming*, 100(3):537–568, 2004.

6 L. Lovász and M. D. Plummer. *Matching Theory.* Akadémiai Kiadó - North Holland, Budapest, 1986.

7 W. Mader. Über die maximalzahl kantendisjunkter H-wege. *Archiv der Mathematik (Basel)*, 31:382–402, 1978.

8 G. Pap. Some new results on node-capacitated packing of A-paths. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 599–604, New York, NY, USA, 2007. ACM. URL: `http://doi.acm.org/10.1145/1250790.1250878`, `doi:10.1145/1250790.1250878`.

9 G. Pap. Strongly polynomial time solvability of integral and half-integral node-capacitated multiflow problems. *EGRES Technical Report*, TR-2008-12, 2008.

10 A. Schrijver. *Combinatorial Optimization.* Springer, Berlin, 2003.