Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 22, no. 4, pp. 555–575 (2018)
DOI: 10.7155/jgaa.00478

Approximation Algorithms for Not Necessarily Disjoint Clustered TSP

Nili Guttmann-Beck[1] *Eyal Knaan*[1] *Michal Stern*[1,2]

[1]Academic College of Tel-Aviv Yaffo, Yaffo, Israel
[2]Caesarea Rothchild Institute, university of Haifa, Haifa, Israel

Abstract

Let $G = (V, E)$ be a complete undirected graph with vertex set V, edge set E and let $H =< G, \mathcal{S} >$ be a hypergraph, where $\mathcal{S}$ is a set of not necessarily disjoint clusters $S_1, \ldots, S_m$, $S_i \subseteq V$ $\forall i \in \{1, \ldots, m\}$. The clustered traveling salesman problem *CTSP* is to compute a shortest Hamiltonian path that visits each one of the vertices once, such that the vertices of each cluster are visited consecutively. In this paper, we present a 4-approximation algorithm for the general case. When the intersection graph is a path, we present a 5/3-approximation algorithm. When the clusters' sizes are all bounded by a constant and the intersection graph is connected, we present an optimal polynomial time algorithm.

Submitted:	Reviewed:	Revised:	Accepted:	Final:
June 2017	June 2018	September 2018	September 2018	September 2018
		Published:		
		November 2018		

Article type:	Communicated by:
Regular paper	S. Albers

E-mail addresses: becknili@mta.ac.il (Nili Guttmann-Beck) eyal@berale.co.il (Eyal Knaan) stern@mta.ac.il (Michal Stern)

1 Introduction

Let $G = (V, E)$ be a complete undirected graph with vertex set V, edge set E and edge lengths $l(e)$. Let $H =< G, \mathcal{S} >$ be a hypergraph, where $\mathcal{S}$ is a set of not necessarily disjoint clusters $S_1, \ldots, S_m$, $S_i \subseteq V$ $\forall i \in \{1, \ldots, m\}$, such that $\cup_{i=1}^{m} S_i = V$. The clustered traveling salesman problem *CTSP* is to compute a shortest Hamiltonian path that visits each one of the vertices once, such that the vertices of each cluster are visited consecutively.

One of the main results of this paper is a 4-approximation algorithm for the general case of the *CTSP*. When the intersection graph is connected and the clusters satisfy the property that for every $i \notin \{j, k\}$: $S_i \nsubseteq (S_j \cup S_k)$, the problem has a feasible solution only if the intersection graph is a path. For this case, we present a $\frac{5}{3}$-approximation algorithm. For both approximation algorithms, we assume that edge lengths satisfy the triangle inequality. Another main result of this paper is for the case where the intersection graph is connected and clusters' sizes are all bounded by a constant, without any additional constraints on the intersections sizes. For this case, we present a polynomial time algorithm which finds an optimal solution. When a feasible solution does not exist, all of the above algorithms give an appropriate statement.

The *CTSP* may be considered as a generalization of the classic traveling salesman problem (*TSP*) where $S_i = \{v_i\}$ and $m = |V|$. The *CTSP* may also be considered as a generalization of the problem where the clusters are disjoint. While for every instance of the problem with disjoint clusters there exists a feasible solution, the intersection between clusters may impose additional constraints creating instances with no feasible solution. The *TSP* and the disjoint-clustered *TSP* are known to be NP-hard [6]. Information about different versions of *TSP* can be found in [14] and [9]. In [6] Christofides gives a well known 1.5-approximation algorithm for the *TSP*. In [7] Frederickson, Hecht and Kim present approximation algorithms for some routing problems, including the Stacker-Crane problem, searching for a *TSP* path which must include a pre-defined set of arcs. In [11] Hoogeveen presents approximation algorithms for minimum length *TSP* paths, where part or all of the endpoints are known.

A lot of research has also been investigated on the clustered *TSP* problem, where the clusters are disjoint. A heuristic for this problem is presented in [4], a branch and bound algorithm for solving this problem is presented in [15] and bounded-approximation algorithms are in [2] and [10]. In [1] the ordered disjoint clustered *TSP* is considered and a $\frac{5}{3}$-approximation algorithm is offered. In [16] a genetic algorithm for solving this problem is presented. All these algorithms cannot be applied to the *CTSP*, as the clusters intersections impose additional restrictions on the required *TSP* path. However, in one case of the problem (Subsection 3.2) we manage to convert the problem into ordered disjoint clusters, and we use the algorithm in [1] for the last step of the solution.

Related work introduces a polynomial time algorithm where the offered solution is a tree (instead of a path) and each cluster is spanned by a sub-tree [12]. In [13] the case where each cluster is spanned by a complete star is solved in polynomial time.

The unweighted version of the *CTSP* may be solved in linear time by the *PQ*-tree data structure presented by Booth and Lueker in [3]. The unweighed solution offers a feasible solution for the weighted *CTSP*. Thus, the *PQ*-tree data structure offers an initial solution in some of the algorithms presented in this work. Related work for the unweighted version is the recognition of interval graphs: Given a graph G, find whether a representation of a path and a collection of sub-paths (clusters) of the path exist, such that the intersection graph of the collection of clusters is the given graph G. The recognition of interval graph may be solved in linear time [8].

A possible application for the problem described in [4] and [15] arises from the area of robotics. In a warehouse, an order for goods contains several sub-orders, not necessarily disjoint, each of which will call for several goods. A motorized robot is dispatched through the warehouse to pick up the goods for each sub-order, the robot may deal with parallel sub-orders, but once it starts handling a sub-order it must complete it consecutively. The aim is to find minimal length route for the robot, such that the order of picking the goods for each sub-order is consecutive.

A similar application described in [15] is the Numerically Controlled machine. In this application, there is a set of customers that require one or more operations with different tools. A machine containing the different tools travel among the customers. Since operating each tool is costly, the path for operating each tool must be consecutive.

Another application is the Physical Mapping of Chromosomes in bio- informatics, described in [5]. Each chromosome is mapped by n probes $P_1, \ldots, P_n$ and m clones $C_1, \ldots, C_m$. The problem is to reconstruct the probes in a manner satisfying that all the probes which hybridizes to one clone appear consecutively in the solution. Using the Hamming distance, the problem is in fact the *CTSP*.

In Section 2 we survey known algorithms which we use throughout the paper. In section 3 we present two approximation algorithms. The first is Algorithm *NDC*, which is a 4-approximation algorithm for the general case of *CTSP*. The second is Algorithm *IGP*, which is a $\frac{5}{3}$-approximation algorithm for the case where the intersection graph is a path. We also prove that this requirement is not too strict as in many cases either the intersection graph is a path or there is no feasible solution. In section 4 Algorithm *BC* is presented. This is a polynomial time algorithm which finds an optimal solution when the intersection graph is connected and the clusters' sizes are bounded by a constant C.

2 Preliminaries

We present here some known notations and results that will be used throughout the paper.

2.1 Christofides' Algorithm for the TSP Path

In [6] Christofides presents the following well known algorithm to approximate the minimal length *TSP* path:

1. Find T a minimum spanning tree of the graph.
2. Find M a minimum weight perfect matching for the odd degree vertices of T.
3. The union of edges in T and M form an Eulerian path of the graph.
4. Use the triangle inequality to create a *TSP* path whose length is no bigger than the length of the Eulerian path.

Theorem 1 *([6]) Christofides' Algorithm returns a TSP path whose length is at most $\frac{3}{2}$ the length of the optimal solution in $O(n^3)$ time complexity assuming the edges' lengths satisfy the triangle inequality.*

2.2 Shortest TSP Path with Known Endpoints

Hoogeveen in [11] studies approximation algorithms for finding a shortest TSP path for three variants of the problem, varying according to the number of known endpoints: zero, one, or two. The algorithm is a slight variation of Christofides' Algorithm. When the number of known endpoints is zero or one, the approximation bound is $\frac{3}{2}$. When two endpoints are given, Hoogeveen's Algorithm is a $\frac{5}{3}$-approximation bound.

The main steps of the algorithm are:

1. Find T a minimum spanning tree of the graph.
2. Find a set $W \subseteq V$ containing all the fixed endpoints of even degree and all other vertices of odd degree. Find M a minimum weight perfect matching on W, leaving $2 - k$ vertices exposed, where k is the number of fixed endpoints.
3. The graph which is the union of edges in T and M is connected and has either two or zero odd degree vertices. In the latter case, there is one fixed endpoint which is exposed in M. Delete an arbitrary edge touching this vertex. Find an Eulerian path using the two odd-degree vertices at its end-points.
4. Use the triangle inequality to create a *TSP* path whose length is no bigger than the length of the Eulerian path.

Theorem 2 *([11]) Hoogeveen's Algorithm returns a TSP path whose length is at most $\frac{5}{3}$ the length of the optimal solution when the two endpoints are known or at most $\frac{3}{2}$ the length of the optimal solution when one endpoint is known assuming the edges' lengths satisfy the triangle inequality. The algorithm works in $O(n^3)$ time complexity.*

2.3 Stacker Crane

The Stacker Crane problem is presented in [7]. This is a version of the *TSP* path, where a set of arcs must be traversed. Given $G = (V, E, A)$ a graph with E a set of undirected edges and A a set of directed arcs, find a minimum length tour, visiting all the vertices in V and traversing all the arcs in A. Assuming the edges lengths satisfy the triangle inequality, the authors offer a 1.8-approximation algorithm. This algorithm is based on running two Procedures *LARGEARC* and *LARGEEDGE* and returning the better solution. Procedure *LARGEARC* is more suitable when the arcs in A are long (compared with E) and Procedure *LARGEEDGE* is more suitable when the edges in E are long. We use Procedure *LARGEARC* as part of one of the algorithms presented in this paper. The main steps of Procedure *LARGEARC* are:

1. Find M a minimum length matching on the endpoints of A using edges from E.
2. The union of M and A creates disjoint cycles, since the degree of each vertex is even.
3. Represent each cycle as a vertex and find a minimum spanning tree using edges from E.
4. Double the edges of the spanning tree.
5. Create a *TSP* tour which traverses all the arcs in A. Use the triangle inequality to find a tour whose length is bounded by $3l(E) + l(A)$.

Theorem 3 *([7]) Procedure LARGEARC returns a TSP path whose length is at most* $3l(E) + l(A)$ *assuming the edges' lengths satisfy the triangle inequality.*

2.4 Ordered Disjoint Clusters

For the special case of disjoint clusters, when the order of the clusters in the *TSP* path is given, Anily, Bramel and Hertz in [1] present an approximation algorithm, yielding a $\frac{5}{3}$-approximation with time complexity of $O(n^3)$.

The main steps of the algorithm are:

1. Find a set of edges that represent the shortest connections between consecutive clusters, denote $a_i, b_i \in S_i$ as the endpoints of these edges.
2. Find a minimum spanning tree within each cluster and let $\mathcal{F}$ be the union of these trees.
3. Augment the graph by duplicating each vertex a_i and b_i with even degree in $\mathcal{F}$. Add a zero length edge between each vertex and its duplicate.
4. Define a symmetric weight function on the set of vertices (including the new duplicates) and find a minimum weight perfect matching using this weight function.

5. Combine all the above edges to construct a feasible solution.

Theorem 4 *([1]) Anily, Bramel and Hertz' Algorithm returns a TSP path whose length is at most $\frac{5}{3}$ the length of the optimal solution in $O(n^3)$ time complexity assuming the edges' lengths satisfy the triangle inequality.*

2.5 PQ-tree

The PQ-tree is a data structure introduced by Booth and Lueker [3] for checking the consecutive ones property (COP) in linear time. A binary matrix satisfies the COP if there exists a permutation of its rows such that in each column the ones appear consecutively. In our application, each row represents a vertex from V and each column represents a cluster in $\mathcal{S}$. A matrix cell contains 1 if the row's vertex exists in the column's cluster. The PQ-tree data structure represents all the permutations of vertices in V that satisfy the clusters' constraints.

A PQ-tree is a rooted tree with internal nodes of two types P and Q. The children of a P-node occur in no particular order, while the children of a Q-node occur in a preserved order, up to reversal. The frontier of a PQ-tree is the permutation of the tree's leaves by reading the label of the leaves from left to right. Two PQ-trees are equivalent if one is obtained from the other either by permuting arbitrarily the order of all the children of a P-node or reversing the order of the children of a Q-node.

The Booth-Lueker Algorithm, henceforth denoted as BL-Algorithm, uses a pattern matching routing based on 11 templates. Each template consists of a pattern matching a possible sub-tree of the current PQ-tree and a replacement of this pattern. The BL-Algorithm works on the tree from bottom to top, replacing the appropriate patterns. After applying the algorithm, either the frontier of the tree satisfies the COP or a 'no feasible solution' message is returned.

In our application, the leaves of the tree represent all the vertices in V. After applying BL-Algorithm, each permutation of the nodes created by the tree represents a possible order of the vertices in a *TSP* path. When the COP is satisfied, the *TSP* path visits the vertices of each cluster consecutively.

Note that for the restricted case of disjoint clusters, an appropriate PQ-tree contains 3 levels. The leaves level represents the vertices in V. The middle level contains one P-node for each cluster, whose sons are the vertices contained in this cluster, with no particular order. The top level contains one P-node as a root, whose sons are all the P-nodes of the middle level.

The structure of a PQ-tree and the use of P-nodes in the tree, allows every order of the vertices inside each cluster and every order of the clusters in the *TSP* path, thus creating all the feasible solutions of the problem. In a PQ-tree which represents an ordered disjoint clustered *TSP* we simply replace the root P-node by a Q-node which defines the order of the clusters.

Definition 5 *In a hypergraph $H =< G, \mathcal{S} >$ with a PQ-tree representation, a node in the PQ-tree* **spans** *$v \in V$ if it is an ancestor of the leaf in the tree which represents v.*

Remark 6 *In this paper we make the following adjustments on the PQ-tree, before applying the BL-Algorithm :*

- *Every original vertex is represented by a leaf, and we denote each leaf as a V-***node** *(vertex node). We add a father node which is a P-node that spans only this leaf. In this manner we assure that every node has at least one ancestor node which is a P-node. This will be used later in Lemma 25.*
- *When a node has exactly two children there is no difference between a P-node and a Q-node. We change every P-node which spans only two children into a Q-node. Therefore, in our representation, a P-node spans at least 3 vertices.*

2.6 G_{int}

Let $H =< G, \mathcal{S} >$ be a hypergraph with $G = (V, E)$ and $\mathcal{S} = \{S_1, \dots, S_m\}$, $S_i \subseteq V$ $\forall i \in \{1, \dots, m\}$. Denote by G_{int} the intersection graph of H, where G_{int} is a graph with node set $s_1, \dots, s_m$, such that s_i represents cluster S_i and an edge exists between s_i and s_j if and only if $S_i \cap S_j \neq \phi$.

Theorem 7 *Checking whether G_{int} is a path and creating G_{int} when it is a path takes $O(mn)$ time complexity, where $m = |\mathcal{S}|$ and $n = |V|$.*

Proof: When G_{int} is a path the following two conditions are satisfied:

1. Each vertex $v \in V$ is contained in at most two clusters from $\mathcal{S}$.
2. There are two nodes in G_{int} with degree 1, all the other nodes in G_{int} have degree 2.

Assume the hypergraph is presented in a table where each row represents a vertex from V and each column represents a cluster in $\mathcal{S}$. A table cell contains 1 if the row's vertex exists in the column's cluster. Perform the following steps:

1. Traverse the input table and create for each vertex $v \in V$ the list of clusters containing this vertex. If a vertex is contained in more than two clusters, indicate that G_{int} is not a path and stop.
2. Create an $m * m$ neighbourhood table for G_{int} using the lists of clusters containing each vertex.
3. Verify that G_{int} is a path using the degree of each cluster in the neighbourhood table.

The above steps require $O(mn)$ time complexity and either indicate that G_{int} is not a path, or create the neighbourhood table which represents G_{int} when it is a path. □

3 Approximation Algorithms

In this section we offer two approximation algorithms. The first algorithm is a bounded 4-approximation algorithm which works on any instance of the *CTSP* problem, even when the intersection graph is not connected. The second approximation algorithm is appropriate for the special case when the intersection graph is connected and forms a path. In this case we achieve a bounded $\frac{5}{3}$-approximation algorithm.

3.1 The General Case

In this section we address the general case, where the clusters' sizes are not bounded and the intersection graph is not necessarily connected. We call the algorithm for approximating this general case *NDC* - "Non-Disjoint Clusters" (See Figure 1). First, the algorithm creates one *PQ*-tree for each connected component of the intersection graph. Next, it adds a root *P*-node whose sons are the roots of previously created *PQ*-trees, thus combining all the trees into one *PQ*-tree, which represents the whole hypergraph. Then the algorithm spans the tree from bottom to top, creating a path to represent each scanned tree node. The path is created either by the order defined by *Q*-nodes in the *PQ*-tree, or using Procedure LARGEARC from Stacker Crane (see [7]) to approximate the path representing the vertices that correspond to all the descendants of a *P*-node.

Remark 8 *By the PQ-tree properties, all the V-nodes that are descendants of the same PQ-tree node are on a consecutive sub-path in any feasible solution.*

Remark 9 *During the algorithm, at the end of every step which handles a P-node, the node is replaced by a Q-node. So when the algorithm reaches a node in a higher level, all its children are either V-nodes or Q-nodes.*

Definition 10 *Denote the followings:*

- *opt - the value of an optimal solution.*
- T^{PQ} *- A PQ-tree on* $\mathcal{S} = \{S_1, \dots, S_m\}$.
- $P^{OPT}(u)$ *- The sub-path spanning all descendants of u in an optimal solution.*
- $P^{NDC}(u)$ *- The sub-path spanning all descendants of u in the solution returned by Algorithm NDC (see Figure 1).*
- P^{NDC} *- The path returned by Algorithm NDC.*
- l_{NDC} *- Distances defined during Algorithm NDC.*

Lemma 11 *In Algorithm NDC, for every node u in* T^{PQ}, $l_{NDC}(P^{NDC}(u)) \leq 3l_{NDC}(P^{OPT}(u))$.

```
NDC (Non-Disjoint Clusters)
    input
    A hypergraph $H =< G, \mathcal{S} >$, where $G = (V, E)$
    with edge lengths $l(e)$, $\forall e \in E$ .
    $\mathcal{S} = \{S_1, \ldots, S_m\}$, $S_i \subseteq V$ $\forall i \in \{1, \ldots, m\}$.
    assumption
    The edge lengths satisfy the Triangle Inequality.
    returns
    A clustered $TSP$ path $P$, or a statement "No feasible solution".
    begin
    Find $G_{int}$ (defined in 2.6).
    Calculate a PQ-tree for each connected component of $G_{int}$,
    using BL-Algorithm ([3]).
    if BL-Algorithm returns "No feasible solution"
      on any of the connected components
      then Return "No feasible solution".
     end if
    Add a P-node as a root, and connect by an edge each PQ-tree as a subtree,
    creating one PQ-tree denoted by $T^{PQ}$.
    Initialize $l_{NDC}(e) = l(e)$ for every $e \in E$.
    All the following steps of the algorithm use the $l_{NDC}$ distances.
    Scan the tree from bottom to top.
    for every $u$ a node which is not a leaf in $T^{PQ}$:
        if $u$ is a Q-node:
          then Use the order defined by the Q-node to create a path $P_u$ in $G$.
          else (u is a P-node)
               if (all the children of $u$ are V-nodes)
                 then Approximate $P_u$ a TSP path spanning all the children
                      of $u$, using Christofides' Algorithm [6].
                 else (u is a P-node with at least one Q-node child defined in
                      this algorithm in lower level of $T^{PQ}$ )
                      Let $\{v_1^u, \ldots, v_k^u\} \subset V$ be the children of $u$ which
                      are V-nodes, and let $\{q_1^u, \ldots, q_j^u\}$ be the children of $u$
                      which are Q-nodes (defined in a lower level of $T^{PQ}$).
                      Each Q-node $q_j^u$ represents an edge $E_j^u$ in $G$.
                      Create $P_u$ a Stacker Crane path on $\{v_1^u, \ldots, v_k^u\}$ and
                      $\{E_1^u, \ldots, E_j^u\}$, using Procedure LARGEARC ([7]).
               end if
        end if
        Represent the path $P_u$:
        · By an edge $E_u$ in $G$ whose length is the length of the path.
        · By a Q-node $q_u$ in $T^{PQ}$,
        where all the vertices of the path are children of $q_u$.
        $q_u$ replaces $u$ in $T^{PQ}$.
```

Figure 1: Algorithm NDC (continues)

```
        Update the distances in G:
        for every  $v \notin E_u$:
            $l_{NDC}(v, E_u) = \min_{w \in P_u} l((v, w))$
        end for
        for every  $E_v \neq E_u$:
            $l_{NDC}(E_v, E_u) = \min_{w_1 \in P_v, w_2 \in P_u} l((w_1, w_2))$
        end for
    end for
    Return $P_u$ where $u$ is the root node of $T^{PQ}$.
    end NDC
```

Figure 1: Algorithm *NDC* (continued)

Proof: The proof of the lemma is by induction on the level of node u in T^{PQ}. The induction is carried on according to the different cases of u in the algorithm.

We note that the lengths used in the algorithm, l_{NDC} (which are minimal distances), are not longer than the original lengths used by the optimal solution.

1. u is a Q-node:

   If all the children of u are V-nodes, then $P^{NDC}(u)$ is $P^{OPT}(u)$, hence $l_{NDC}(P^{NDC}(u)) = l_{NDC}(P^{OPT}(u))$.

   Otherwise, for every w, a child of u that is not a V-node, w is a Q-node created during the algorithm and is represented in G by an edge created in a lower level of T^{PQ} (at an earlier stage of the algorithm). By the induction hypothesis, $l_{NDC}(P^{NDC}(w)) \leq 3l_{NDC}(P^{OPT}(w))$. Since the order of the children of u is uniquely defined, the same order also exists in the optimal solution, giving that $l_{NDC}(P^{NDC}(u)) \leq 3l_{NDC}(P^{OPT}(u))$.

2. u is a P-node and all the children of u are V-nodes:

   By Theorem 1, Christofides Algorithm gives

   $$l_{NDC}(P^{NDC}(u)) \leq 1.5l_{NDC}(P^{OPT}(u)).$$

3. u is a P-node with at least one child which is a Q-node defined in lower level of T^{PQ}:

   Let $w_1, \ldots, w_k$ be the children of u which are Q-nodes defined in lower level of T^{PQ}. Let $E_1, \ldots, E_k$ be the corresponding edges defined in G by the algorithm. By the construction of the algorithm, each edge E_j is created to represent the path $P^{NDC}(w_j)$ with $l(E_j) = l(P^{NDC}(w_j))$. Let A be the union of all these edges $A = \cup_{j=1}^{k} E_j$, giving that $l_{NDC}(A) = \sum_{j=1}^{k} l_{NDC}(E_j) = \sum_{j=1}^{k} l_{NDC}(P^{NDC}(w_j))$. Let A' be the union of the corresponding optimal sub-paths: $A' = \cup_{j=1}^{k} P^{OPT}(w_j)$. Hence $l_{NDC}(A') = \sum_{j=1}^{k} l_{NDC}(P^{OPT}(w_j))$.

Using Algorithm *LARGEARC* and by Theorem 3, the length of the returned $P^{NDC}(u)$ is bounded by $3{*}(l_{NDC}(P^{OPT}(u)) - l_{NDC}(A')) + l_{NDC}(A)$. By the induction hypothesis, for every w_j a child of u, $l_{NDC}(P^{NDC}(w_j)) \leq 3l_{NDC}(P^{OPT}(w_j))$. Hence $l_{NDC}(A) \leq 3\sum_{j=1}^{k} l_{NDC}(P^{OPT}(w_j))$. Therefore,

$$\begin{aligned} l_{NDC}(P^{NDC}(u)) &\leq 3*(l_{NDC}(P^{OPT}(u)) - l_{NDC}(A')) + l_{NDC}(A) \\ &\leq 3*(l_{NDC}(P^{OPT}(u)) - 3\sum_{j=1}^{k} l_{NDC}(P^{OPT}(w_j) \\ &\quad +3\sum_{j=1}^{k} l_{NDC}(P^{OPT}(w_j)) \\ &= 3*(l_{NDC}(P^{OPT}(u))) \end{aligned}$$

□

Corollary 12 $l_{NDC}(P^{NDC}) \leq 3opt$.

Theorem 13 $l(P^{NDC}) \leq 4opt$.

Proof: The length of P^{NDC}, calculated by the lengths defined in the algorithm, uses minimal distances which might use inner vertices of the sub-paths. We note that for every sub-cluster, this may happen exactly once for entering the sub-cluster and once for leaving the sub-cluster. The true lengths may be larger, but, using the triangle inequality, the length added to the final *TSP* solution is bounded by the lengths of the optimal *TSP* sub-path inside each sub-cluster. Since the optimal solution contains a *TSP* path inside each sub-cluster, the total added length is bounded by *opt*. □

Corollary 14 *Algorithm NDC, for the general not necessarily disjoint clusters CTSP, returns a 4-approximated solution, in polynomial time, when a feasible solution exists, or reports that there is no feasible solution.*

3.2 Intersection Graph Path

In this section we present algorithm *IGP* (Intersection Graph Path) which is an approximation algorithm for the case when G_{int} is a path. The path representing G_{int} uniquely defines the order of the clusters in the solution *TSP* path. The algorithm (see Figure 2) first verifies that G_{int} is a path. In this case, the algorithm uses the order of the clusters in this path to partition the graph vertices into $2m-1$ disjoint sub-clusters $B_1, \ldots, B_{2m-1}$. In the final step we use the algorithm presented in [1] to find the required *TSP* path. The approximation ratio of the algorithm in this case is $\frac{5}{3}$.

We also prove that when $S_i \not\subseteq (S_j \cup S_k)$ for every $i \notin \{j, k\}$, then a feasible solution exists only when G_{int} is a path. Hence, requiring that G_{int} is a path is relevant for most interesting instances of the *CTSP* problem.

```
IGP (Intersection Graph Path)
    input
    A hypergraph $H =< G, \mathcal{S} >$, where $G = (V, E)$
    with edge lengths $l(e)$, $\forall e \in E$ .
    $\mathcal{S} = \{S_1, \ldots, S_m\}$ $S_i \subseteq V$ $\forall i \in \{1, \ldots, m\}$.
    assumptions
    The edge lengths satisfy the Triangle Inequality.
    returns
    A TSP path P, or a statement "$G_{int}$ is not a path".
    begin
    If there is only one cluster in G, return an approximated TSP path
    using Christofides' Algorithm [6].
    Check whether $G_{int}$ (defined in 2.6) is a path.
    if ($G_{int}$ is not a path)
      then return "$G_{int}$ is not a path".
      else Create $G_{int}$ as a path.
           Use the order of the nodes in the path representing $G_{int}$
           to define an order on the clusters: $S_1, S_2, \ldots, S_m$.
           Identify the following partition of V:
           $B_1 = S_1 \backslash S_2$.
           for every $i \in \{1, \ldots, m-2\}$
               $B_{2i} = S_i \cap S_{i+1}$.
               $B_{2i+1} = S_{i+1} \backslash (S_i \cup S_{i+2})$.
           end for
           $B_{2(m-1)} = B_{2m-2} = S_{m-1} \cap S_m$.
           $B_{2m-1} = S_m \backslash S_{m-1}$.
           Calculate and return a TSP path using Anily et al. Algorithm ([1]).
    end if
    end IGP
```

Figure 2: Algorithm *IGP*

Theorem 15 *Algorithm IGP (see Figure 2), for CTSP with an intersection graph which is a path, returns a $\frac{5}{3}$-approximated solution in $O(n^3)$ time complexity.*

Proof: If there is only one cluster, we use Christofides' Algorithm. By Theorem 1 we find a $\frac{3}{2}$- approximated *TSP* path in $O(n^3)$ time.
Otherwise, suppose that there are at least two clusters. If G_{int} is not a path, the algorithm reports an appropriate message. If G_{int} is a path, it uses the order of the clusters to define the sub-clusters $B_1, \ldots, B_{2m-1}$.

The last step of Algorithm *IGP* uses Anily et al. Algorithm, hence by Theorem 4 the approximation ratio of Algorithm *IGP* is $\frac{5}{3}$.

By Theorems 1, 4 and 7 the time complexity for the whole algorithm is $O(n^3)$. □

The next theorem justifies our interest in the special case where G_{int} is a path. It proves that when no two clusters contain a third one, then a feasible solution exists only when G_{int} is a path.

Theorem 16 *In a hypergraph $H =< G, \mathcal{S} >$, suppose that for every $i \notin \{j,k\}$: $S_i \not\subseteq (S_j \cup S_k)$ and that there exists a feasible CTSP path for H, then the corresponding intersection graph is a path.*

Proof: Consider a feasible solution. This solution is a path on the vertices in V. This path defines an order of the clusters in $\mathcal{S}$ and therefore implies a path in G_{int}. Hence, G_{int} includes a path: $s_{p_1}, \ldots, s_{p_m}$. Suppose G_{int} includes an edge outside the path: (s_{p_i}, s_{p_j}) with $j > i+1$. Therefore, $S_{p_i} \cap S_{p_j} \neq \phi$. Since $j > i+1$ there is another index k satisfying $i < k < j$. The feasible solution contains, in the following order, all the vertices of S_{p_i}, the vertices of $S_{p_k} \backslash (S_{p_i} \cup S_{p_j})$ and only after that all the vertices of S_{p_j}. Since $S_{p_i} \cap S_{p_j} \neq \phi$ we get that $S_{p_k} \backslash (S_{p_i} \cup S_{p_j}) = \phi$, giving that $S_{p_k} \subseteq (S_{p_i} \cup S_{p_j})$, contradicting the assumption of the lemma. □

Remark 17 *When the intersection graph of H is a path and the intersection's size of every two clusters is bounded by a constant, it is possible to obtain a bounded $\frac{5}{3}$-approximated algorithm which works in $O(\sum_{j=1}^{m} |S_j|^3 + mn)$ time complexity, using dynamic programming.*

4 A Polynomial Algorithm for Bounded Size Clusters

In this section we assume that the intersection graph is connected and that $|S_i| < C$ for every $i \in \{1, \ldots, m\}$, for a constant C, but we pose no additional constraints on the size of the intersections. For this case we present a polynomial time algorithm, denoted by *BC* (Bounded size Clusters), which finds an optimal solution, even when the edge lengths do not satisfy the triangle inequality. Note

that in this case we profit from the additional constraints posed by the clusters, as they enable us to obtain a polynomial algorithm.
In this case, G_{int} is not necessarily a path, even when a feasible solution exists. Therefore, instead of G_{int}, we use the structure of the appropriate *PQ*-tree to define ordered disjoint sub-clusters $B_1, \dots, B_q$. Note that these sub-clusters are different from the ones defined and used in algorithm *IGP*. On these sub-clusters we activate a special defined dynamic procedure, denoted by *DPBC* (Dynamic Programming for Bounded size Clusters), to find a *TSP* path which satisfies all constraints imposed by the clusters.

In Procedure *DPBC*:

1. for every $u \in B_i$, $2 \le i \le q$, we calculate $f(u)$ - the length of the optimal shortest *TSP* path, which ends at u and includes all the vertices in $B_1 \cup \dots \cup B_{i-1} \cup \{u\}$ and spans consecutively every contained-cluster of $B_1 \cup \dots \cup B_{i-1} \cup \{u\}$.
2. for every $v \in B_q$, we calculate $d(v)$ - the length of the optimal shortest *TSP* path, which starts at v, spans all the vertices in B_q and spans consecutively every contained-cluster of B_q.

In the end, the algorithm uses the above function values and a concatenation of the corresponding *TSP* paths to create one *TSP* path which spans all the vertices of the graph in an appropriate order.
We start with some definitions.

Definition 18 *In a PQ-tree:*

- *An* **ancestor-P-node** *is a P-node which has only Q-nodes as ancestors.*
- *A* **high-Q-node** *is a Q-node with only Q-nodes as its ancestors. (A high-Q-node does not have an ancestor which is a P-node.)*

Definition 19 *In a hypergraph $H =< G, \mathcal{S} >$ with a PQ-tree representation, a cluster $S_i \in \mathcal{S}$ is:*

- **P-nested-cluster** *if there is an ancestor-P-node which spans all the vertices of S_i and at least one vertex which is not in S_i.*
- **non-contained-cluster** *if there is no cluster S_j such that $S_i \subsetneq S_j$.*

Note that there are clusters which may be neither P-nested-clusters nor non-contained-clusters.

Lemma 20 *In a hypergraph $H =< G, \mathcal{S} >$, the order of non-contained-clusters in every feasible solution is unique.*

Proof: First, we identify a partition of V (the vertices of G) into disjoint sub-clusters defined by the intersections of the non-contained-clusters, such that every non-empty intersection of at least two non-contained clusters defines a sub-cluster. Every cluster contains at least two sub-clusters, since it must intersect with at least one other non-contained cluster by the connectivity of the intersection graph. We claim that the order of these sub-clusters is unique in every feasible solution and that the unique order of the sub-clusters implies a unique order of the non-contained clusters.

Suppose, by contradiction, that there is a non-contained cluster S_i with two different feasible orderings of its sub-clusters, $F1$ and $F2$, such that $F2$ is not the reversal of $F1$. Since its sub-clusters must appear consecutively in every order, for S_i to have two different orderings of its sub-clusters, it must contain at least three disjoint sub-clusters: $S_{i_1}, S_{i_2}, S_{i_3}$. Without loss of generality, suppose that in $F1$ they are ordered $(S_{i_1}, S_{i_2}, S_{i_3})$ and in $F2$ they are ordered $(S_{i_1}, S_{i_3}, S_{i_2})$. Since the ordering $(S_{i_1}, S_{i_2}, S_{i_3})$ is feasible, there is a non-contained cluster $S_j \neq S_i$, such that S_{i_3} is contained in $S_i \cap S_j$ and S_{i_2} is not contained in S_j. Since S_j is a non-contained cluster $S_j \backslash S_i \neq \phi$. In $F1$ the vertices of the sub-clusters contained in $S_j \backslash S_i$ must appear adjacent to S_{i_3}, to ensure that all the vertices of S_j appear consecutively in any feasible solution. However, in $F2$ the vertices of S_j do not appear consecutively, since $S_{i_3} \subset S_j$, and $S_{i_2} \not\subset S_j$, a contradiction.

Previous reasoning proves that the order of the sub-clusters contained in S_i is unique in any feasible solution. Since each sub-cluster is defined by an intersection with other non-contained clusters, the order of S_i with respect to other non-contained clusters is uniquely defined by the order of its sub-clusters. Thus, implying a unique order among all non-contained clusters. □

Lemma 21 *In a hypergraph $H =< G, \mathcal{S} >$ with a PQ-tree representation and $|V| > C$, a P-node spans at most $C - 1$ vertices.*

Proof: Since $|V| > C$ there are at least two non-contained-clusters. Clearly, if a P-node spans more than C vertices, then it spans vertices of at least two non-contained-clusters. Under the assumption of connected intersection graph, the same also holds when a P-node spans exactly C vertices. This contradicts Lemma 20 which states that the order between any two non-contained-clusters is unique and therefore cannot be defined by a P-node. □

Corollary 22 *In a hypergraph $H =< G, \mathcal{S} >$ with a PQ-tree representation, if S_i is a P-nested-cluster, it cannot be a non-contained-cluster, therefore there is a cluster S_j such that $S_i \subsetneq S_j$.*

Proof: Suppose S_i is a P-nested-cluster and a non-contained-cluster. Since it is a P-nested-cluster, there is a P-node p which spans all the vertices of S_i and at least one vertex which is not in S_i. Since all the vertices of S_i must appear consecutively, there is a node t in the PQ-tree which spans all the vertices of S_i and is a child of p. According to Remark 6, a P-node has at least 3 children, therefore p has another two children t_1, t_2, each of them spans vertices which are not in S_i. Since p is a P-node, both orderings are allowed t_1, t, t_2 and t, t_1, t_2, which gives two allowed orderings between S_i and another non-contained cluster, contradicting Lemma 20, which states that the order of the non-contained-clusters is unique. □

```
BC (Bounded size Clusters)
    input
    A hypergraph $H =< G, \mathcal{S} >$, where $G = (V, E)$
    with edge lengths $l(e)$, $\forall e \in E$ .
    $\mathcal{S} = \{S_1, \ldots, S_m\}$, $S_i \subseteq V$, $|S_i| \leq C$, $\forall i \in \{1, \ldots, m\}$.
    assumption
    The intersection graph of G is connected.
    returns
    A TSP Path P, or a statement "No feasible solution".

    begin
    If there is only one cluster in G, return the optimal TSP path.
    Apply BL-Algorithm on H (see [3]) to find a PQ-tree .
    if BL-Algorithm returns "No feasible solution"
      then Return "No feasible solution".
      else Let $T^{PQ}$ be the PQ-Tree returned by BL-Algorithm.
    end if
    Define a set of disjoint sub-clusters $B_1, \ldots, B_q$:
    - Each sub-cluster is the set of vertices spanned by an ancestor-P-node.
    - The order of the sub-clusters is defined by the high-Q-nodes,
    which are the ancestors of the ancestor-P-nodes.
    (For an ancestor-P-node ignore the structure of the tree under this node.)

    Return an optimal TSP path P,
    using the dynamic programming in Procedure DPBC (Figure 4).
    end BC
```

Figure 3: Algorithm *BC*

Corollary 23 *In a PQ-tree which represents a given hypergraph, a non-contained-cluster cannot be a P-nested-cluster, hence only a high-Q-node may span all its vertices. Therefore, the order of the non-contained-clusters is uniquely defined by the high-Q-nodes.*

Corollary 24 *In a PQ-tree which represents a given hypergraph, a cluster that has an ancestor which is an ancestor-P-node and spans all its vertices, has at most C vertices for which an optimal TSP path order can be found in constant time, when C is constant.*

Lemma 25 *In a PQ-tree which represents a given hypergraph, every V-node is spanned by exactly one ancestor-P-node.*

Proof: Every V-node is a leaf of the tree and, according to changes we defined in Remark 6, is connected by an edge to a P-node. Hence, it also has an ancestor-P-node. By the structure of a tree it can only have one ancestor-P-node. □

```
DPBC (Dynamic Programming for Bounded size Clusters)
    input
    A graph $G = (V, E)$ with edge lengths $l(e)$, $\forall e \in E$.
    A partition of $V$ into disjoint sub-clusters $B_1, \ldots, B_q$.
    returns
    A TSP path $P$.

    begin
    for every $u \in B_2$:
        Calculate $f(u) =$ the length of the optimal TSP path which:
        - Spans the vertices of $B_1 \cup \{u\}$,
        - Spans consecutively every contained-cluster of $B_1$,
        - Ends at $u$.
        Save the corresponding path as $Pu$.
    end for

    for every $3 \le i \le q$:
        Call Procedure BP (Figure 5)
        to calculate $f(u)$ and $P_u$ for every $u \in B_i$.
    end for

    for every $v \in B_q$:
        Calculate $d(v) =$ the length of the optimal TSP path which:
        - Starts at $v$,
        - Spans the vertices of $B_q$,
        - Spans consecutively every contained-cluster of $B_q$.
        Save the corresponding path as $PE_v$.
    end for
    Find $f_{DP} = \min_{v \in B_q}\{f(v) + d(v)\}$ .
    Calculate and return the TSP path whose length is $f_{DP}$.
    This path is the concatenation of $P_{v^*}$ and $PE_{v^*}$,
    where $v^* = argmin_{v \in B_q}\{f(v) + d(v)\}$.
    end DPBC
```

Figure 4: Procedure *DPBC*

```
BP (Between Path)
    input
    A graph $G = (V, E)$ with edge lengths $l(e)$, $\forall e \in E$.
    Two sub-clusters $B_{i-1}, B_i \subset V$
    Function values $f(v)$ and corresponding paths $P_v$ for every $v \in B_{i-1}$.
    returns
    Function values $f(u)$ and corresponding paths $P_u$ for every $u \in B_i$ .
    begin
    for every $u \in B_i$:
        for every $v \in B_{i-1}$:
            Calculate $bd(v, u)$ = the length of the optimal TSP path
            which:
            - Starts at $v$,
            - Spans the vertices of $B_{i-1} \cup \{u\}$,
            - Spans consecutively every contained-cluster of $B_{i-1}$,
            - Ends at $u$.
            Save the corresponding path as $PB_{(v,u)}$.
        end for
        Calculate $f(u) = \min_{v \in B_{i-1}}\{f(v) + bd(v, u)\}$.
        Save the corresponding path as $P_u$,
        where $P_u$ is the concatenation of $P_{v^*}$ and $PB_{(v^*,u)}$,
        where $v^* = argmin_{v \in B_{i-1}}\{f(v) + bd(v, u)\}$.
    end for
    end BP
```

Figure 5: Procedure *BP*

Corollary 26 *In Algorithm BC (see Figure 3) each one of the vertices (of V) belongs to exactly one sub-cluster from $B_1, \ldots, B_q$.*

Theorem 27 *Algorithm BC, for CTSP with clusters' sizes which satisfy $|S_i| \leq C$, for a constant C (see Figures 3, 4 and 5), returns an optimal solution in polynomial time, when a feasible solution exists, or reports that there is no feasible solution.*

Proof: If there is only one cluster, an optimal *TSP* path can be found in constant time, under the assumption that C is constant.
Otherwise, suppose that there are at least two clusters. Each sub-cluster in $B_1, \ldots, B_q$, $q \geq 3$, is defined by an ancestor-P-node. By Lemma 21 every sub-cluster contains at most $C - 1$ vertices. By definition, all the ancestors of ancestor-P-nodes are high-Q-nodes. Therefore, the order of $B_1, \ldots, B_q$ in an optimal solution is uniquely defined by the order of high-Q-nodes of the PQ-tree. This is the same order imposed on the non-contained-clusters, which is uniquely defined in Corollary 23.

The correctness of the dynamic programming (which can be trivially proved by induction) guarantees the return of an optimal solution.

For the complexity, the algorithm contains the following steps:

1. Find a *PQ*-tree using *BL*-Algorithm (defined in [3]).
2. For every $v \in B_{i-1}$ and $u \in B_i$, calculate the optimal *TSP* path which starts at v, ends at u and spans all the vertices of B_{i-1}. The optimal solution can be found in polynomial time, since the clusters' sizes are bounded by C. Note that we calculate the paths inside B_{i-1}, whose size is bounded by C-1. Therefore, we can also demand that the paths satisfy all the constraints imposed by the contained clusters, in polynomial time complexity.
3. Calculate $f(v)$ for every $v \in B_i$, $i \in \{3, \ldots, q\}$ (using the optimal paths found in step 2).
4. Calculate $f(v)$ for every $v \in B_2$ and $d(v)$ for every $v \in B_q$.

All the above steps are polynomial, assuming that C is constant. □

5 Summary and Future Research

Our significant result is the bounded approximation algorithm for finding a *TSP* path when not necessarily disjoint clusters are defined on the vertex set. We also present a better approximation when certain restrictions are imposed on the structure of the intersection graph. When the clusters' sizes are bounded and the intersection graph is connected we present a polynomial time algorithm for finding an optimal solution.

It will be interesting to research the general case further, trying to improve the approximation bound for this case. Furthermore, additional special cases of the problem may be defined and solved.

Acknowledgements

We thank Ephraim Korach for introducing and motivating us to work on this problem. We would also like to thank Refael Hassin for his careful reading and helpful suggestions.

References

[1] S. Anily, J. Bramel, and A. Hertz. A $\frac{5}{3}$-approximation algorithm for the clustered traveling salesman tour and path problems. *Operations Research Letters*, 24(1-2):29–35, 1999. doi:10.1016/S0167-6377(98)00046-7.

[2] E. M. Arkin, R. Hassin, and L. Klein. Restricted delivery problems on a network. *Networks*, 29(4):205–216, 1997. doi:10.1002/(SICI)1097-0037(199707)29:4<205::AID-NET3>3.3.CO;2-X.

[3] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using *PQ*-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976. doi:10.1016/S0022-0000(76)80045-1.

[4] J. A. Chisman. The clustered traveling salesman problem. *Computers & Operations Research*, 2(2):115–119, 1975. doi:10.1016/0305-0548(75)90015-5.

[5] T. Christof, M. Jünger, J. Kececioglu, P. Mutzel, and G. Reinelt. A branch-and-cut approach to physical mapping of chromosomes by unique end-probes. 4(4):433–447, 02 1997. doi:10.1089/cmb.1997.4.433.

[6] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[7] G. N. Frederickson, M. S. Hecht, and C. E. Kim. Approximation algorithms for some routing problems. In *17th Annual Symposium on Foundations of Computer Science*, pages 216–227. IEEE Computer Society, 1976. doi:10.1109/SFCS.1976.6.

[8] D. R. Fulkerson and O. A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15:835–855, 1965. doi:10.2140/pjm.1965.15.835.

[9] G. Gutin and A. Punnen. Editorial: the traveling salesman problem. *Discrete Optimization*, 3(1):1, 2006. doi:10.1016/j.disopt.2005.12.001.

[10] N. Guttmann-Beck, R. Hassin, S. Khuller, and B. Raghavachari. Approximation algorithms with bounded performance guarantees for the clustered traveling salesman problem. *Algorithmica*, 28(4):422–437, 2000. doi:10.1007/s004530010045.

[11] J. A. Hoogeveen. Analysis of Christofides' heuristic: some paths are more difficult than cycles. *Operations Research Letters*, 10(5):291–295, 1991. doi:10.1016/0167-6377(91)90016-I.

[12] E. Korach and M. Stern. The clustering matroid and the optimal clustering tree. *Mathematical Programming*, 98(1-3, Ser. B):385–414, 2003. doi:10.1007/s10107-003-0410-x.

[13] E. Korach and M. Stern. The complete optimal stars-clustering-tree problem. *Discrete Applied Mathematics*, 156(4):444–450, 2008. doi:10.1016/j.dam.2006.12.004.

[14] E. L. Lawler, J. K. Lenstra, A. H. G. R. Kan, D. B. Shmoys, et al. *The traveling salesman problem: a guided tour of combinatorial optimization.* Wiley, New York, 1985.

[15] F. C. J. Lokin. Procedures for travelling salesman problems with additional constraints. *European Journal of Operational Research*, 3(2):135–141, 1979. doi:10.1016/0377-2217(79)90099-7.

[16] J. Y. Potvin and F. Guertin. A genetic algorithm for the clustered traveling salesman problem with a prespecified order on the clusters. In *Advances in computational and stochastic optimization, logic programming, and heuristic search*, volume 9 of *Operations Research/Computer Science Interfaces Series*, pages 287–299. Springer, 1998. doi:10.1007/978-1-4757-2807-1_11.