INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

WILEY

INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. 00 (2017) 1–9
DOI: 10.1111/itor.12422

A branch-and-cut algorithm for the Team Orienteering Problem

Nicola Bianchessi[a], Renata Mansini[b] and M. Grazia Speranza[c]

[a] *Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Jakob-Welder-Weg 9, D-55128 Mainz, Germany*
[b] *Department of Information Engineering, University of Brescia, via Branze 38, 25123 Brescia, Italy*
[c] *Department of Economics and Management, University of Brescia, C.da S. Chiara 50, 25122 Brescia, Italy*
E-mail: nbianche@uni-mainz.de [Bianchessi]; renata.mansini@unibs.it [Mansini]; grazia.speranza@unibs.it [Speranza]

Received 24 August 2016; received in revised form 14 January 2016; accepted 6 April 2017

Abstract

The Team Orienteering Problem aims at maximizing the total amount of profit collected by a fleet of vehicles while not exceeding a predefined travel time limit on each vehicle. In the last years, several exact methods based on different mathematical formulations were proposed. In this paper, we present a new two-index formulation with a polynomial number of variables and constraints. This compact formulation, reinforced by connectivity constraints, was solved by means of a branch-and-cut algorithm. The total number of instances solved to optimality is 327 of 387 benchmark instances, 26 more than any previous method. Moreover, 24 not previously solved instances were closed to optimality.

Keywords: Team Orienteering Problem; two-index mathematical formulation; branch-and-cut algorithm

1. Introduction

The Team Orienteering Problem (TOP) can be defined over a complete directed graph $G = (V, A)$ with node set $V = \{0, \ldots, n+1\}$ and arc set $A = \{(i, j) : i \neq n+1;\ j \neq 0;\ i \neq j;\ i, j \in V\}$. The node set V contains nodes 0 and $n+1$, representing the initial and final depots, respectively, and the set $N = \{1, \ldots, n\}$, indicating the n customers. Each node $i \in V$ has a profit p_i, with $p_0 = p_{n+1} = 0$. A nonnegative travel time t_{ij}, with $t_{0,n+1} = 0$, is associated with each arc $(i, j) \in A$. Travel times satisfy the triangle inequality. A fleet F of m identical vehicles is available to serve the customers. When a customer is visited, the corresponding profit is collected by the vehicle serving the customer. Each customer cannot be visited more than once. Each route is a path from node 0 to node $n+1$ in graph G. Moreover, the duration of the route associated with each vehicle cannot exceed a predefined

threshold $T^{\max}$. The objective is to find a set of vehicle routes each of which not exceeding the maximum time duration and serving a subset of the customers in such a way that the total collected profit is maximized.

The problem, proposed by Chao et al. (1996), has been recently studied and new solution algorithms, both exact and heuristic, have been proposed. In this paper, we focus on mathematical formulations proposed for the problem and exact solution approaches. If we exclude some preliminary works by Gueguen (1999) and Butt and Ryan (1999) analyzing variants of the TOP (with time windows and a heterogeneous fleet of vehicles, respectively), the first exact approach is due to Boussier et al. (2007) where a branch-and-price algorithm was proposed. The method, based on a set packing formulation, has the advantage of being easily adaptable to other vehicle routing problems with profits. In particular, the authors apply it to the TOP and to a variant by considering capacity and time window constraints. The authors tested the method on the 387 benchmark instances proposed by Chao et al. (1996) by solving to optimality 270 instances. In Poggi et al. (2010), different formulations were proposed. However, they did not present a full implementation of the branching scheme and, thus, reported only partial results. In Dang et al. (2013), the authors solved a three-index formulation with a polynomial number of binary variables and generalized subtour elimination constraints. They introduced a branch-and-cut method based on valid inequalities and a set of dominance properties that includes symmetry breaking conditions on the objective function, lower and upper bounds on solution profit and on the number of customers, and clique cuts based on graph incompatibilities. They were able to solve 29 previously unsolved benchmark instances. More recently, Keshtkaran et al. (2016) proposed a branch-and-price approach where the pricing subproblem is solved by a bounded bidirectional dynamic programming algorithm with decremental state space relaxation featuring a two-phase dominance rule relaxation. The authors were able to solve 301 of 387 instances, by closing 17 previously unsolved instances. The most recent exact approach is due to El-Hajj et al. (2016) who proposed the effective use of a linear formulation with a polynomial number of variables. Cutting planes are the core component of the algorithm that closed to optimality 12 previously unsolved benchmark instances.

In this paper, we propose a new two-index formulation with a polynomial number of variables and constraints. This compact formulation was inspired by the formulation proposed by Maffioli and Sciomachen (1997) for the Sequential Ordering Problem (SOP) where jobs characterized by a release date and a due time have to be processed on a single machine. A setup time was also considered for changing from one job to the other. The SOP consists in finding an ordering of the jobs such that the completion time of the job processed last is minimized. The formulation we propose for the TOP was reinforced by the introduction of connectivity constraints (CCs) and solved by means of a branch-and-cut algorithm. We compared it with the methods of Boussier et al. (2007), Dang et al. (2013), Keshtkaran et al. (2016), and El-Hajj et al. (2016) on the 387 benchmark instances of Chao et al. (1996). The total number of instances solved to optimality is 311, when tested on a single thread, and 327 when multiple threads are allowed, 10 and 26 more than any previous method, respectively. Most interestingly, 24 previously unsolved instances were solved to optimality, reducing the number of open instances to 49.

The paper is organized as follows. In Section 2, we describe the mathematical formulation used, and in Section 3, the branch-and-cut approach. Computational results are presented in Section 4.

2. Mathematical formulation

We model the problem by means of a two-index commodity flow formulation that uses three sets of variables. Binary variables x_{ij}, $(i, j) \in A$, and y_i, $i \in N$, are equal to 1 if arc (i, j) is traversed and customer i is visited, respectively, whereas continuous variables z_{ij}, $(i, j) \in A \setminus \{(0, n+1)\}$, represent the arrival time at vertex j of a vehicle coming from vertex i. For each set $S \subseteq N$, let $\delta^+(S) = \{(i, j) \in A : i \in S, j \notin S\}$ and $\delta^-(S) = \{(i, j) \in A : i \notin S, j \in S\}$ be the set of arcs leaving and entering the set S, respectively, with $\delta^+(i) = \delta^+(\{i\})$ and $\delta^-(i) = \delta^-(\{i\})$. The model is as follows:

$$\max \sum_{i \in N} p_i y_i \tag{1}$$

$$\sum_{j \in N} x_{0j} = \sum_{i \in N} x_{i,n+1} = m \tag{2}$$

$$\sum_{(j,i) \in \delta^-(i)} x_{ji} = \sum_{(i,j) \in \delta^+(i)} x_{ij} = y_i \qquad i \in N \tag{3}$$

$$z_{0j} = t_{0j} x_{0j} \qquad j \in N \tag{4}$$

$$\sum_{(i,j) \in \delta^+(i)} z_{ij} - \sum_{(j,i) \in \delta^-(i)} z_{ji} = \sum_{(i,j) \in \delta^+(i)} t_{ij} x_{ij} \qquad i \in N \tag{5}$$

$$z_{ij} \leq T^{\max}_{j,n+1} x_{ij} \qquad (i, j) \in A \setminus \{(0, n+1)\} \tag{6}$$

$$z_{ij} \geq t^0_{ij} x_{ij} \qquad (i, j) \in A \setminus \{(0, n+1)\} \tag{7}$$

$$y_i \in \{0, 1\} \qquad i \in N \tag{8}$$

$$x_{ij} \in \{0, 1\} \qquad (i, j) \in A \setminus \{(0, n+1)\} \tag{9}$$

$$0 \leq x_{0,n+1} \leq m, \tag{10}$$

where $t^0_{ij} = t_{0i} + t_{ij}$, with $t_{00} = 0$, and $T^{\max}_{j,n+1} = T^{\max} - t_{j,n+1}$, with $t_{n+1,n+1} = 0$.

This formulation has a polynomial number of variables and constraints. The objective function (1) calls for the maximization of the collected profit. Constraints (2) and (3) are the degree constraints for the depots and the customer nodes, respectively. In particular, constraints (2) impose that exactly m vehicles have to leave and come back to the depot, whereas constraints (3) impose that a customer node i is entered and leaved exactly once if it is visited (i.e. if $y_i = 1$). Constraints (4) bound the flow originating from the initial depot. Flow conservation constraints (5) update the flow value coming out from each visited customer, while constraints (6) set the time limit on the duration of each route. Constraints (7) impose lower bounds on the values of the $\mathbf{z}$ variables in order to restrict the range of feasible values that these variables can assume in fractional solutions. Note that if $x_{ij} = 1$, then z_{ij} denotes the arrival time at node j, whereas $x_{ij} = 0$ implies $z_{ij} = 0$. Finally, (8)–(10) are variable definition constraints.

In order to strengthen the formulation, a further constraint can be imposed on the global duration of the routes:

$$\sum_{(i,j)\in A\setminus\{(0,n+1)\}} t_{ij}x_{ij} \leq mT^{\max}. \tag{11}$$

3. Branch-and-cut algorithm

In this section, we briefly describe the branch-and-cut algorithm used to solve the formulation described in Section 2 with the addition of valid inequality (11).

Although equalities (4) and (5) are sufficient to prevent the generation of subtours and to ensure the connectivity of each route, we decided to further strengthen the problem formulation by adding the following more general type of CCs (for the use of these cuts in three-index formulations for the Vehicle Routing Problem, see Toth and Vigo, 2002, p. 15):

$$\sum_{(i,j)\in\delta^+(S)} x_{ij} \geq y_h \quad S \subseteq N, |S| \geq 2, h \in S. \tag{12}$$

The number of the CCs is exponential in the size of N. This excludes the possibility to explicit them through a commercial Mixed-Integer Linear Programming (MILP) solver even for small-sized instances. The CCs are thus dynamically separated along the branch-and-bound tree.

Let $(\bar{\mathbf{y}}, \bar{\mathbf{x}}, \bar{\mathbf{z}})$ be an optimal fractional solution of (1)–(11). Violated CCs are identified by means of an exact separation algorithm as follows. We consider the graph $\bar{G} = (V, \bar{A})$ induced by the fractional solution, where arc $(i, j) \in \bar{A}$ if the corresponding $\bar{x}_{ij}$ value is greater than 0. With each arc $(i, j) \in \bar{A}$, a capacity equal to $\bar{x}_{ij}$ is associated. Then, for each pair $< i, n+1 >$, $i \in N$, the max-flow/min-cut problem is solved on graph $\bar{G}$, where vertex i is the source of the flow and vertex $n+1$ is the sink. We consider the partition S induced by the min-cut such that $i \in S$. If $n+1 \notin S$ and the value of the flow is less than $\bar{y}_v$ for some $v \in S$, then a violated CC has been identified. The violated constraint is inserted for $v^* = argmax_{v\in S}\{\bar{y}_v\}$. Max-flow/min-cut problems are solved by means of the algorithm described in Boykov and Kolmogorov (2004).

4. Experimental results

The branch-and-cut algorithm was implemented in C++ using CPLEX 12.6 Concert Technology. The code was compiled in release mode with MS Visual C++ 2012 Express Edition for Windows Desktop. The experiments were carried out on a 64-bit Windows computer, with the Intel Xeon processor W3680, 3.33 GHz, and 12 GB of RAM. CPLEX built-in cuts were used in all experiments. We set IloCplex::ParallelMode to 1 in order to force CPLEX to always use deterministic algorithms. Parameter IloCplex::Threads has been set to different values as illustrated in the following. For all the other CPLEX's parameters, we kept their default values. All experiments were run on the 387 benchmark instances proposed by Chao et al. (1996) with a time limit of two hours as in Boussier et al. (2007), Dang et al. (2013), Keshtkaran et al. (2016), and El-Hajj et al. (2016). Although a direct comparison of methods using different hardware is not a straightforward task, we recall here that

Table 1
Benchmark instances

Set	#	$\|N\|$	m	$T^{\max}$
1	54	32	2–4	3.8–22.5
2	33	21	2–4	1.2–42.5
3	60	33	2–4	3.8–55.0
4	60	100	2–4	3.8–40.0
5	78	66	2–4	1.2–65.0
6	42	64	2–4	5.0–200.0
7	60	102	2–4	12.5–120.0

Boussier et al. (2007) used a PC Pentium IV 3.2 GHz, Dang et al. (2013) tested their branch-and-cut method on an AMD Opteron 2.60 GHz using CPLEX 12.4 as an MIP solver, Keshtkaran et al. (2016) conducted their experiments on a single core of an Intel Core i7 3.6 GHz, and finally El-Hajj et al. (2016) run tests on an AMD Opteron 2.60 GHz using CPLEX 12.5 as an MIP solver.

The data set by Chao et al. (1996) consists of 387 instances divided into seven sets according to the number of control points (customers) ranging from 21 to 102. In each set, the instances differ only for the number of vehicles available to serve the customers (from two to four) and for the value of the maximum time duration of a route. For each set, Table 1 reports the number of instances in the set (#), the number of customers in each instance ($|N|$), the range values for the number of available vehicles (m), and for the maximum time duration of a route ($T^{\max}$).

As previously done in Dang et al. (2013), a node is removed if the travel time of the tour containing only such a node exceeds the time limit $T^{\max}$. Similarly, we eliminated all the inaccessible arcs.

We first show the results on the effectiveness of the CCs and then compare our method with the methods proposed by Boussier et al. (2007), Dang et al. (2013), Keshtkaran et al. (2016), and El-Hajj et al. (2016). A detailed comparison on the partial set of instances tested in Poggi et al. (2010) is reported in Keshtkaran et al. (2016).

Table 2 is devoted to testing the effectiveness of the CCs. This first set of experiments was carried out setting the number of threads available to CPLEX to 1 (IloCplex::Threads := 1). The first three columns report the set of instances (Set), the number of vehicles (m), and the number of instances in the row (#). Then, the following six columns show the results for the branch-and-cut algorithm on the compact formulation (1)–(11) with the separation of the CCs. The six columns report the number of instances solved to optimality, the average computational time needed in seconds to find the optimal solution, the number of unsolved instances, the final average optimality gap computed on the unsolved instances, the average number of branch-and-bound nodes inspected, and the average number of cuts generated. The last five columns report similar statistics for the case in which the CCs are not included. In general, including the CCs is beneficial, in terms of number of instances solved, computational time needed to find the optimal solutions, and optimality gap for the unsolved instances. There are only few cases (see, e.g., the results for Set 5) in which the solution of the pure compact formulation gives slightly better results. In such cases, it seems that the separation of the CCs guides the search toward area of the solution space not including good feasible solutions, leading CPLEX to explore a higher number of nodes, and to sometimes miss the optimal solution.

Table 2
Branch-and-cut algorithm—time limit: two hours—single thread

			With CCs						Without CCs				
			Solved		Unsolved				Solved		Unsolved		
Set	m	#	#	Time (seconds)	#	Gap (%)	Nodes	Cuts	#	Time (seconds)	#	Gap (%)	Nodes
1	2	18	18	2.233	0	–	290.7	38.3	18	4.728	0	–	605.4
	3	18	18	2.967	0	–	1042.3	24.0	18	6.489	0	–	1335.9
	4	18	18	0.611	0	–	259.4	14.1	18	1.628	0	–	537.2
2	2	11	11	0.973	0	–	272.1	22.4	11	0.855	0	–	282.1
	3	11	11	0.064	0	–	17.9	4.7	11	0.009	0	–	4.2
	4	11	11	0.000	0	–	0.0	0.2	11	0.000	0	–	0.0
3	2	20	20	28.930	0	–	5014.2	35.5	20	66.380	0	–	6895.3
	3	20	19	469.758	1	2.88	71,156.9	23.9	20	787.605	0	–	35,486.1
	4	20	19	276.147	1	1.79	61,471.2	29.4	19	198.621	1	1.79	48,443.3
4	2	20	17	1304.765	3	0.68	30,906.0	443.7	14	1927.436	6	1.74	45,082.1
	3	20	9	643.978	11	1.56	115,191.7	283.2	9	802.944	11	2.32	115,258.3
	4	20	9	282.322	11	3.91	125,023.1	106.5	9	327.222	11	4.49	112,327.5
5	2	26	24	648.763	2	1.57	59,728.4	642.8	25	409.544	1	0.96	87,440.7
	3	26	13	92.208	13	3.60	291,827.1	350.1	14	251.764	12	3.31	311,263.5
	4	26	17	487.365	9	4.58	307,584.4	98.0	17	471.153	9	4.13	272,844.4
6	2	14	11	405.091	3	1.96	75,377.9	1904.7	12	118.417	2	1.58	141,742.6
	3	14	11	656.191	3	2.67	127,752.2	143.7	12	364.100	2	1.82	93,577.4
	4	14	14	48.186	0	–	12,613.0	9.4	14	30.250	0	–	7160.4
7	2	20	19	1293.295	1	1.06	30,225.4	603.3	15	2178.367	5	1.22	72,520.9
	3	20	12	1018.058	8	2.89	137,497.6	293.5	11	846.218	9	2.86	148,221.8
	4	20	10	304.410	10	2.99	264,050.8	192.7	11	648.318	9	3.44	214,593.0
			Total	Average	Total	Average	Average	Average	Total	Average	Total	Average	Average
			311	394.279	76	2.97	95,608.9	256.0	309	438.032	78	3.01	95,295.4

Table 3
Comparison with other exact methods

	Solution algorithms													
	Boussier et al. (2007)		Dang et al. (2013)		Keshtkaran et al. (2016)		El-Hajj et al. (2016)		Branch-and-cut (with CCs)					
									Single thread			Multithread		
Set	Solved	Time (seconds)	Solved	Time (seconds)	Solved	Time (seconds)	Solved	Time (seconds)	Solved	Time (seconds)	Gap (%)	Solved	Time (seconds)	Gap (%)
1	51/54	38.2	54/54	–	54/54	12.9	54/54	–	54/54	1.9	0	54/54	1.1	0
2	33/33	0.1	33/33	–	33/33	0.1	33/33	–	33/33	0.3	0	33/33	0.2	0
3	50/60	103.8	60/60	–	60/60	258.3	60/60	–	58/60	254.3	2.34	60/60	184.9	0
4	25/60	459.9	22/60	–	20/60	120.4	30/60	–	35/60	871.9	2.49	39/60	870.4	2.29
5	48/78	200.6	44/78	–	60/78	252.2	54/78	–	54/78	463.9	3.80	60/78	517.9	3.49
6	36/42	286.1	42/42	–	36/42	203.2	42/42	–	36/42	343.1	2.32	36/42	22.1	1.92
7	27/60	203.2	23/60	–	38/60	768.6	27/60	–	41/60	971.5	2.85	45/60	992.8	2.53
Total	270/387		278/387		301/387		300/387		311/387			327/387		

Table 4
New optimal solutions

Instance	n	m	$T^{\max}$	Opt.	Time	
					Single thread	Multithread
p4.2.c	76	2	35	452	3.8	3.0
p4.2.d	90	2	40	531	115.4	32.5
p4.2.e	97	2	45	618	83.6	38.3
p4.2.g	98	2	55	757	513.3	178.7
p4.2.n	98	2	90	1174	885.5	289.4
p4.2.o	98	2	95	1218	1528.7	546.6
p4.3.d	45	3	26.7	335	5.2	2.1
p4.3.e	56	3	30	468	55.4	16.4
p4.3.f	71	3	33.3	579	35.1	9.0
p4.3.j	97	3	46.7	861	–	4196.8
p4.3.n	98	3	60	1121	–	6633.4
p4.4.g	49	4	27.5	461	33.5	9.0
p5.3.r	64	3	30	1125	–	6979.1
p7.2.j	100	2	100	646	1263.0	282.0
p7.2.k	100	2	110	705	1387.8	390.1
p7.2.l	100	2	120	767	690.0	126.4
p7.2.m	100	2	130	827	2156.1	1009.4
p7.2.n	100	2	140	888	1083.4	85.1
p7.2.o	100	2	150	945	1615.7	627.7
p7.2.p	100	2	160	1002	4234.1	3154.0
p7.2.r	100	2	180	1094	3063.1	4147.9
p7.2.s	100	2	190	1136	4417.5	5789.1
p7.3.o	100	3	100	874	–	5640.8
p7.3.p	100	3	106.7	929	–	6231.4

Table 3 summarizes the results of the comparison between our branch-and-cut algorithm with CCs and the other state-of-the-art exact algorithms. We included, in the comparison, two versions of our algorithm: single thread and multithread. In the first version, CPLEX is forced to use one thread only (IloCplex::Threads := 1), while in the second version CPLEX can use up to six threads (IloCplex::Threads := 6). For each known method, each row reports the number of instances solved to proven optimality (out of the total number of instances in the set), and the average computational time w.r.t. such instances optimally solved. Since Dang et al. (2013) and El-Hajj et al. (2016) do not report complete results on the computational times, only the number of instances solved to optimality is shown in the table for such methods. For each version of our algorithm we report, in addition to the number of solved instances and their average computational time, the average optimality gap (%) computed w.r.t. the instances not solved to proven optimality. In terms of total number of instances solved, the single-thread version solves 10 instances more than any other method, the multithread 26 more instances.

As each method solves different sets of instances, it is also interesting to look at the specific instances solved. Then, in Table 4, we list the instances solved to optimality for the first time by means of our branch-and-cut algorithm. Columns labeled with "Instance", "n", "m", and "$T^{\max}$" indicate the name of the instance solved, the number of customers, the number of vehicles, and the

time limit, respectively. Column "Opt." reports the objective function optimal value. The last two columns show the computational time in seconds required to solve the instance in the single-thread and multithread versions. A "–" indicates that the instance could not be solved to optimality within the time limit.

5. Conclusions

A new compact formulation for the TOP, with a polynomial number of variables and constraints, was presented. Such a formulation was reinforced with CCs and solved with a branch-and-cut algorithm. The approach was compared with all the previously proposed exact methods that were tested on the 387 benchmark instances. In terms of total number of instances, we solved to optimality 311 instances on a single thread and 327 on multiple threads. This implies that we solved 10 more instances and 26 more instances with a single thread and multiple threads, respectively, than any other previous method. Most interestingly, we solved 24 previously unsolved instances. This is due to the fact that the different methods solve different sets of instances and, somehow, complement each other. The extension of the proposed formulation to other routing problems is a promising research direction.

References

Boussier, S., Feillet, D., Gendreau, M., 2007. An exact algorithm for team orienteering problems. *4OR* 5, 211–230.

Boykov, Y., Kolmogorov, V., 2004. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26, 9, 1124–1137.

Butt, S.E., Ryan, D.M., 1999. An optimal solution procedure for the multiple tour maximum collection problem using column generation. *Computers & Operations Research*, 26, 4, 427–441.

Chao, I.-M., Golden, B.L., Wasil, E.A., 1996. The team orienteering problem. *European Journal of Operational Research* 88, 464–474.

Dang, D.-C., El-Hajj, R., Moukrim, A., 2013. A branch-and-cut algorithm for solving the team orienteering problem. In Gomes, C., Sellmann, M. (eds) *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Lecture Notes in Computer Science. Springer, Berlin-Heidelberg, pp. 332–339.

El-Hajj, R., Dang, D.-C., Moukrim, A., 2016. Solving the team orienteering problem with cutting planes. *Computers & Operations Research* 74, 21–30.

Gueguen, C., 1999. *Méthodes de résolution exacte pour les problémes de tournées de véhicules*. PhD thesis, Ecole centrale des arts et manufactures, Châtenay-Malabry, France.

Keshtkaran, M., Ziarati, K., Bettinelli, A., Vigo, D., 2016. Enhanced exact solution methods for the team orienteering problem. *International Journal of Production Research* 54, 591–601.

Maffioli, F., Sciomachen, A., 1997. A mixed-integer model for solving ordering problems with side constraints. *Annals of Operations Research* 69, 277–297.

Poggi, M., Viana, H., Uchoa, E., 2010. The Team Orienteering Problem: formulations and branch-cut and price. In Erlebach, T., Lübbecke, M. (eds) *10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'10)*. Schloss DagstuhlLeibniz-Zentrum für Informatik, Wadern, Germany, pp. 142–155.

Toth, P., Vigo, D. (eds), 2002. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, Philadelphia, PA.