Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in DISCRETE MATHEMATICS

Electronic Notes in Discrete Mathematics 64 (2018) 305–314 www.elsevier.com/locate/endm

A Hybrid Optimization Approach For the Steiner k-Connected Network Design Problem

Ibrahima Diarrassouba [1]

Le Havre University, LMAH, FR-CNRS-3335, Le Havre, France

Mohamed Khalil Labidi [2]

University of Tunis el Manar - Faculty of Science of Tunis, URAPOP, UR13ZS38, Tunis, Tunisia
&
Paris Dauphine University - PSL Research University, CNRS UMR 7243, LAMSADE, 75016 Paris, France

Ali Ridha Mahjoub [3]

Paris Dauphine University - PSL Research University, CNRS UMR 7243, LAMSADE, 75016 Paris, France

Abstract

In this paper, we consider the Steiner k-Edge-Connected Network Design Problem (SkESNDP). The problem finds its applications in the design of survivable telecommunications networks. We propose a parallel hybrid algorithm which aims to produce good solutions for large scale instances of the problem. Our approach is based on a Lagrangian relaxation of a flow-based integer programming formulation of the problem, a greedy and a genetic algorithms.

Keywords: Lagrangian relaxation, large scale, genetic algorithm, hybridization, parallel computing, SkESNDP.

https://doi.org/10.1016/j.endm.2018.02.005

1 Introduction

Let $G = (V, E)$ be an undirected graph, a subset of nodes $S \subseteq V$ called *terminals*, a weight function $\omega : E \to \mathbb{R}$ which associates the weight ω_e with each edge $e \in E$. The *Steiner k-Edge-Connected Network Design Problem* (SkESNDP for short) is the problem of finding a minimum weight subgraph of G spanning S such that between every two nodes $u, v \in S$, there are at least k edge-disjoint paths. The SkESNDP is a special case of a more general model, introduced by [7] and later called *generalized Steiner problem* by [9].

The SkESNDP is well known to be NP-hard and is a generalization of the Steiner tree problem in which it is required that the nodes of S are spanned by a Steiner tree of minimum weight.

Several works have been done on the Steiner network problem and its variants. Mahjoub and Kerivin [5] have presented a survey of the main variants of survivable network design problems and integer programming formulations as well as some polyhedral descriptions. Also, Magnanti and Raghavan [6] have presented differents types of formulations for a quite more general problem, which includes the SkESNDP. In particular, they have presented several formulations based on flow variables and compare them in terms of LP-bounds.

Survivable network design problems are still subject of extensive researches and several variants have been recently investigated. For instance, Binh et al. [3] and Bui et al. [4] have devised genetic algorithms for solving two variants of the survivable network design problem.

In this paper, we present a parallel hybrid algorithm for solving the SkESNDP. This approach takes advantage from the structure of the flow-based integer formulation presented in [6] and is based on a greedy, a Lagrangian relaxation and a genetic algorithms.

2 Integer programming formulation

First, we denote by $D = \{\{s, t\} \mid \text{for all } s, t \in S, \text{with } s \neq t\}$. We also let $d = |D| = |S|(|S| - 1)/2$. The SkESNDP can be formulated as the following integer program (see [6]). For each edge $uv \in E$, let x_{uv} be the $0 - 1$ variable

[1] Email: ibrahima.diarrassouba@univ-lehavre.fr
[2] Email: mohamed-khalil.labidi@dauphine.fr
[3] Email: mahjoub@lamsade.dauphine.fr

which takes value 1 if the edge uv is in the solution and 0 otherwise. Also let $\widetilde{G} = \{V, A\}$ the directed graph obtained from G by replacing each $uv \in E$ by two arcs (u, v) and (v, u), and for every pair $\{s, t\} \in D$, let f_{uv}^{st} be the flow variable associated with the arc (u, v) of $\widetilde{G}$. The SkESNDP is equivalent to

$$\min \sum_{uv \in E} \omega_{uv} x_{uv}$$

$$\sum_{v \in V \setminus \{u\}} f_{uv}^{st} - \sum_{l \in V \setminus \{u\}} f_{lu}^{st} = \left\{ \begin{array}{ll} k, \, if & u = s, \\ -k, \, if & u = t, \\ 0, \, if & u \in V \setminus \{s, t\}, \end{array} \right\} \text{ for all } u \in V \text{ and } \{s, t\} \in D, \quad (1)$$

$$\left. \begin{array}{l} f_{uv}^{st} \\ f_{vu}^{st} \end{array} \right\} \leq x_{uv}, \qquad \text{for all } uv \in E \text{ and } \{s, t\} \in D, \quad (2)$$

$$f_{uv}^{st}, f_{vu}^{st} \geq 0, \qquad \text{for all } uv \in E \text{ and } \{s, t\} \in D, \quad (3)$$

$$x_{uv} \leq 1, \qquad \text{for all } uv \in E, \quad (4)$$

$$x_{uv} \in \{0, 1\}, \qquad \text{for all } uv \in E, \quad (5)$$

$$f_{uv}^{st} \in \{0, 1\}, \qquad \text{for all } uv \in E, \{s, t\} \in D. \quad (6)$$

This formulation is called *Undirected Flow Formulation*. Inequalities (1) are the *flow conservation constraints*. Inequalities (2) are the *linking constraints*. They ensure that if an edge uv is not taken in the solution, then no st-flow can use this edge. Inequalities (3) and (4) are the trivial inequalities.

3 Parallel hybrid optimization algorithm

Our algorithm relies on the usage of parallel computing for solving the SkESNDP. Namely, we devise a heuristic for the problem based on three algorithms

- *a greedy heuristic (SH)*;
- *a Lagrangian relaxation algorithm (RLA)*;
- *a genetic algorithm (GA)*.

For our purpose, we run these three algorithms in a parallel computing framework. Also, as we will see, each iteration of each algorithm is used to improve the other algorithms, and hence, improve the whole algorithm. Moreover, we solve the Lagrangian relaxation algorithm using parallel computing.

We describe, in the remain of the section, each algorithm in the following and the communication between them.

3.1 The greedy successive heuristic

Our greedy algorithm (SH) for solving the SkESNDP consists in computing a series of minimum cost st-flows in the graph $\widetilde{G}$, for all $\{s,t\} \in D$.

First, we sort the pairs of D in an arbitrary order, say $\{s_1,t_1\},...,\{s_d,t_d\}$. We start with the pair $\{s_1,t_1\}$, and compute a minimum cost s_1t_1-flow in $\widetilde{G}$, where all the arcs have capacity 1 and both arcs (u,v) and (v,u) are cost ω_{uv}, for all $uv \in E$. Let A_1 be the set of arcs that have a s_1t_1-flow value of 1, and E_1 be the set of edges of G corresponding to the arcs of A_1. Then we choose the pair $\{s_2,t_2\}$, fix to ω_{uv} the cost of the arcs (u,v) and (v,u) for all $uv \in E \setminus E_1$ and fix to 0 the cost of the arcs (u,v) and (v,u), for all $uv \in E_1$, and compute a minimum cost s_2t_2-flow. We build the arc set A_2 and the edge set E_2, as before A_1 and E_1, and so on until all the pairs $\{s_i,t_i\}$ have been explored. Finally, we build a solution of the SkESNDP by considering the edges of $E_1 \cup E_2 \cup ... \cup E_d$.

3.2 The Lagrangian relaxation algorithm

In our Lagrangian relaxation algorithm, we consider the Undirected Flow Formulation and relax the linking constraints (2). Let λ_{uv}^{st}, for all $\{s,t\} \in D$ and all $(u,v) \in A$, be the Lagrangian multiplier associated with constraints (2). This yields the following problem

$$\min \sum_{uv \in E} \left[\omega_{uv} - \left(\sum_{\{s,t\} \in D} (\lambda_{uv}^{st} + \lambda_{vu}^{st}) \right) \right] x_{uv} + \sum_{\{s,t\} \in D} \sum_{uv \in E} (\lambda_{uv}^{st} f_{uv}^{st} + \lambda_{vu}^{st} f_{vu}^{st})$$

$$\sum_{v \in V \setminus \{u\}} f_{uv}^{st} - \sum_{l \in V \setminus \{u\}} f_{lu}^{st} = \left\{ \begin{array}{rl} k, \, if & u = s, \\ -k, \, if & u = t, \\ 0, \, if & u \in V \setminus \{s,t\}, \end{array} \right\} \text{for all } u \in V \text{ and } \{s,t\} \in D,$$

$$\begin{aligned} 0 \leq f_{uv}^{st}, f_{vu}^{st} \leq 1, &\qquad \text{for all } uv \in E, \{s,t\} \in D, \\ x_{uv} \leq 1, &\qquad \text{for all } uv \in E, \\ x_{uv} \in \{0,1\}, &\qquad \text{for all } uv \in E, \\ f_{uv}^{st} \in \{0,1\}, &\qquad \text{for all } uv \in E, \{s,t\} \in D. \end{aligned}$$

One can see that solving this relaxation consists in solving d independant minimum cost st-flow problems in the graph $\widetilde{G}$, for all $\{s,t\} \in D$. In fact, we can see that solving the minimum cost st-flow problems gives the optimal

value of the flow variables f^{st}_{uuv}, and for the variables x_{uv}, the optimal values are $x_{uv} = 1$ if $\omega_{uv} - \sum_{\{s,t\}\in D} (\lambda^{st}_{uv} + \lambda^{st}_{vu}) > 0$ and 0 otherwise.

Since the minimum cost st-flow problems are independant, they can be solved in parallel using d processors.

For the Lagrangian relaxation algorithm, the Lagrangian multipliers are updated using the subgradient method.

Also, notice that each iteration of the Lagrangian relaxation algorithm produces a feasible solution. The solution is obtained by considering all the edges of G corresponding to the arcs of $\widetilde{G}$ that get at least once a flow value of 1 during the resolution of the minimum cost st-flow problems. As we will see, the solution thus obtained are used as input of the genetic algorithm, which is described below.

3.3 The genetic algorithm

A genetic algorithm consists in considering a set of feasible solutions (population) of problem P, combining two solutions (parents solutions) chosen randomly, in order to produce one, or more, new feasible solutions. A genetic algorithm mainly relies on four aspects: the encoding (or representation) of each solution, the initial population, the evaluation and selection of each solution, and the crossover and reproduction of the solutions. For more details on genetic algorithms, the reader can refer to [8].

We describe in the following the main points we use for devising our genetic algorithm for solving the SkESNDP. Notice that the whole procedure is repeated until a given number of iteration is reached (counting starts after RLA and SH end).

3.3.1 Encoding and initial population

Each solution is represented by a $0-1$ vector $(\overline{f}^1, ..., \overline{f}^d)$, where $\overline{f}^i$ is a flow vector associated with a demand $\{s_i, t_i\}$, $i = 1, ..., d$. To see that such a vector allows to represent a solution of the SkESNDP, it suffices to consider the vector $\overline{x} \in \mathbb{R}^E$ such that $\overline{x}_{uv} = max\{f^i_{uv}, f^i_{vu},$ for all $i = 1, ..., d\}$, and notice that $(\overline{f}^1, ..., \overline{f}^d, \overline{x})$ satisfies constraints (1).

As initial population, we choose some feasible solutions produced by algorithms RLA and SH.

3.3.2 Evaluation and selection

Each solution $(\overline{f}^1, ..., \overline{f}^d, \overline{x})$ is evaluated by the weight of the subgraph of G it induces, namely by $Z = \sum_{uv \in E} \omega_{uv} \overline{x}_{uv}$.

The selection is made randomly according to a specific probability distribution scale. We start by dividing the population into five categories (A, B, C, D and E) based on the social ranking (fitness value), and then we select $\frac{1}{10}$ of the population to be reproduced knowing that the probability that a parent is chosen from the A social category for example is equal to 67%, and respectively to the B, C, D and E are equal to 19%, 10%, 3% and 1%.

Notice that the strategy of management of the pool size guarantees that the number of solutions slowly decreases until it remains in the end one solution.

3.3.3 Crossover and reproduction

The reproduction scheme is based on the two-point crossover. Two randomly chosen points a and b with $1 \leq a \leq b \leq d$ are used as cutting points. Two offspring are formed by permuting the two parents. The components between the two cutting points are inherited from the first parent and the remaining positions are filled from the second parent as long as the solution remains feasible.

Moreover, we assign to each pair of selected parents a measure that tries to predict on a $0-100$ scale the eventual quality of the produced offspring. It is obvious that if the two parents belong to the social class A, such parents are more able to produce good quality children than others. Thereby, according to this measure, the algorithm produces more offspring based on different randomly chosen two cutting points.

3.4 Hybridisation

The hybridisation scheme of the three algorithms (components) presented above is designed in order to take advantage from each algorithm. The results obtained at each iteration of each algorithm are used to reinforce (if possible) the other algorithms. The three components are self-contained, and are executed in a parallel multithreaded fashion. One the central tool of the algorithm is a pool of feasible solutions for the SkESNDP which is managed by the three algorithms. Figure 1 presents the communication scheme.

The communications between the components are summarized below. Note that UB_{SH}, UB_{RLA} and UB_{GA} denote the upper bounds obtained respectively by algorithm SH, RLA and GA, and UB denotes the best upper bound. Also,

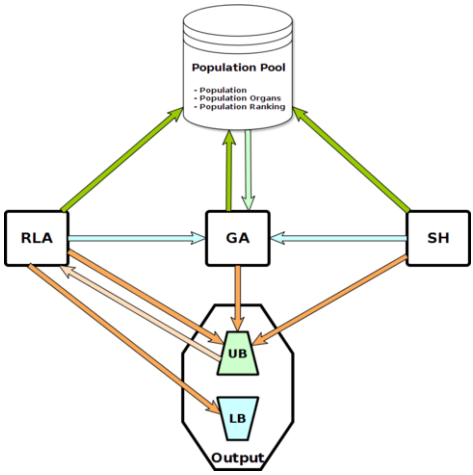

Fig. 1. PHA communication scheme

LB_{RLA} denotes the lower bound achieved by RLA and LB denotes the best lower bound.

(i) Execute SH, RLA et GA in a parallel multithreaded fashion.

(ii) SH:
  (a) For each iteration, SH adds to the pool the new solution (individual).
  (b) Updates UB with UB_{SH} if $UB_{SH} \leq UB$.
  (c) Informs GA when the algorithm ends.

(iii) RLA:
  (a) For each iteration, RLA uses UB to compute the Lagrangian multipliers,
  (b) solves the Lagrangian relaxation and adds to the pool the new solution,
  (c) updates UB with UB_{RLA} if $UB_{RLA} \leq UB$,
  (d) updates LB with LB_{RLA} if $LB_{RLA} \geq LB$,
  (e) informs GA when the algorithm ends.

(iv) GA:
  (a) For each iteration, GA reads the pool, processes the evaluation phase, and updates the pool by deleting the worst individuals (if SH and RLA are ended),
  (b) Selects the individuals to reproduce and computes the new generation,
  (c) Updates the pool by adding the new generation data,
  (d) Updates UB with UB_{GA} if $UB_{GA} \leq UB$.

4 Experimental Study

The algorithms described in the previous section have been implemented in C++, using LEMON graph structures [1]. They have been tested on an Intel i7 2.5 Ghz with 8 Gb of RAM, running under Linux. We have fixed the maximum CPU time to 2 hours. The test problems have been obtained from the TSPLIB library [2]. Each test set consists in complete graphs whose edge weights are the rounded Euclidean distance between the edge vertices. In addition, for each graph, a set S of terminals is defined including the first $|S|$ nodes of G.

We have used OpenMP library and the C++ inner parallel library `std::thread` in order to manage the multithreading parallel computations. Also, in order to avoid inconsistencies in the solutions pool, we use the C++ library `std::mutex` for the shared memory management.

We have also implemented the Undirected Flow Formulation using CPLEX 12.6 concert technology library and have solved the problem for the same instances. The results obtained by all the algorithms are summarized in Table 1 whose entries are:

$|V|$: number of nodes of the graph,

$|S|$: number of terminals,

UB : best upper bound,

LB : best lower bound,

Gap : relative error between the best upper and lower bounds,

CPU : total CPU time in hours:min:sec.

The results of PHA algorithm are interesting (see Table 1).

First we can easily notice that the approach is able to improve, for all the instances, the upper bounds given by SH and RLA. However we notice that, the upper bound given by RLA is very weak as in 17 instances over 20.

Second, comparing PHA to CPLEX we can see that our algorithm produces better upper bounds in 13 cases over 20 while CPLEX is able to solve to optimality three instances. We can also see that for 14 instances, CPLEX produces a better lower bound than PHA. Also, the gap produced by PHA is better than that produced by CPLEX for 13 instances.

Now, we compare the algorithms SH, RLA and GA, taken separately, with algorithm PHA. We can see that for all the instances, algorithm PHA produces

Table 1
Numerical results for the algorithms RLA, SH, PHA and CPLEX for $k = 3$

Instances			RLA				SH		PHA				CPLEX			
name	$\|V\|$	$\|S\|$	*UB*	*LB*	*Gap*	*CPU*	*UB*	*CPU*	*UB*	*LB*	*Gap*	*CPU*	*UB*	*LB*	*Gap*	*CPU*
	30	3	2539	2409.13	5.12	00:00:03	2800	00:00:00	2503	2285.98	8.67	00:00:00	2489	2489	0	00:00:00
	30	5	5507	2993.18	45.65	00:00:06	4022	00:00:00	3638	2949.69	18.92	00:00:45	3191	3191	0	00:02:08
berlin	52	3	2536	1501.06	40.81	00:00:04	2871	00:00:00	1853	1653.38	10.77	00:00:03	1776	1776	0	00:00:02
	52	5	11793	2201.2	81.33	00:00:20	3408	00:00:00	2971	2135.77	28.11	00:01:33	2462	2298.33	6.65	02:00:00
	52	7	20400	3127.03	84.67	00:00:37	4761	00:00:00	4187	3006.06	28.2	00:03:19	3802	3389.61	10.85	02:00:00
	70	5	1040	152.446	85.34	00:00:29	360	00:00:00	236	164.30	30.38	00:00:07	223	179	19.73	02:00:00
st	70	7	1548	172.465	88.86	00:00:43	408	00:00:00	364	204.73	43.76	00:03:02	252	225	10.71	02:00:00
	70	9	2920	201.558	93.1	00:01:13	552	00:00:00	507	222.73	56.07	00:07:10	688	297.111	56.82	02:00:00
	100	5	62059	5317.46	91.43	00:01:16	16366	00:00:00	13151	5085.76	61.33	00:05:10	15292	5812.28	61.99	02:00:00
	100	7	106829	5418.4	94.93	00:02:32	19902	00:00:00	19469	5871.49	69.84	00:12:48	24360	6270.08	74.26	02:00:00
	100	9	151739	5358.9	96.47	00:04:12	21614	00:00:00	19846	5270.00	73.45	00:24:35	49051	7821.06	84.06	02:00:00
	150	7	106958	4248.35	96.03	00:05:38	19870	00:00:00	17882	4280.29	76.06	00:26:35	31512	5556.12	82.37	02:00:00
kroA	150	9	155197	5034.3	96.76	00:09:34	21345	00:00:00	20038	4826.98	75.91	00:52:44	51384	6582	87.19	02:00:00
	150	11	240264	7681.11	96.8	00:14:15	24937	00:00:00	24016	7517.20	68.7	01:27:04	50323	8060.58	83.98	02:00:00
	200	9	194591	6294.12	96.77	00:17:02	22947	00:00:00	20403	6088.26	70.16	01:33:39	117741	5458.25	95.36	02:00:00
	200	11	296236	7472.18	97.48	00:22:48	27178	00:00:01	25195	7173.50	71.53	02:00:00	222560	6361.25	97.14	02:00:00
	200	13	375741	9256.4	97.54	00:35:33	29989	00:00:01	26397	8525.46	67.7	02:00:00	294825	7890.25	97.32	02:00:00
	318	11	34628	2848.95	91.77	00:54:49	6252	00:00:04	5671	1978.31	65.12	02:00:00	251507	0	100	02:00:00
lin	318	13	56637	3553.51	93.73	01:00:24	8051	00:00:05	7660	2656.36	65.32	02:00:00	93116900	0	100	02:00:00
	318	15	75722	3369.12	95.55	01:37:53	9331	00:00:07	9017	2749.56	69.51	02:00:00	–	–	–	–

a better upper bound than SH and RLA. We can also observe that, for several instances, the lower bound achieved by RLA taken separately is better than that obtained by PHA. These two observations show that the combination of the three algorithms does not always help in improving the lower bound, but clearly helps in improving the upper bound.

Finally, we can see that the CPU time of PHA is relatively small (less than 1h) for 13 instances over 20, while CPLEX reaches the maximum CPU time for almost all the instances (18 instances over 20). Moreover, PHA has been able to produce an upper bound for instance lin318-15 while CPLEX was not able to produce even a feasible solution due to lack of memory.

5 Concluding Remarks

In this paper, we have presented a parallel hybrid algorithm for the SkESNDP, based on a Lagrangian relaxation algorithm, a greedy heuristic and a genetic algorithm. The series of experiments we have conducted have shown that our algorithm outperforms CPLEX in the production of upper bounds, and this, even for large size instances. The algorithm is able to produce solutions with a guarantee since it produces both upper and lower bounds. This contrasts with

the behaviour of most of the heuristics that can be found in the litterature, in particular those based on meta-heuristics like genetic algorithms. Even if the gaps, and especially the lower bounds, are not very tight, such a behaviour is an interesting point that, we think, deserves to be improved.

Also, it would be interesting to improve the parallel implementation of the algorithm, still in the perspective of improving the solutions produced by the algorithm. This can be done by using for example heterogenous machines equipped by GPUs. It would also be interesting to improve the implementation of the algorithm by using distributed architectures.

Finally, we can investigate the usage of such techniques in a Branch-and-Cut algorithm for the SkESNDP. This could be used, for instance, for improving the cutting plane or the branching phases of a Branch-and-Cut algorithm.

References

[1] *COIN-OR library for efficient modeling and optimization in networks lemon*, http://lemon.cs.elte.hu/trac/lemon, accessed: 2016-10-11.

[2] *Gerhard Reinelt tsplib*, http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95, accessed: 2016-10-11.

[3] Binh, H. T. T., S. H. Ngo and D. N. Nguyen, "Genetic Algorithm for Solving Survivable Network Design with Simultaneous Unicast and Anycast Flows," Springer Berlin Heidelberg, Berlin, Heidelberg, 2013 pp. 1237–1247.

[4] Bui, L. T. and H. Thi Thanh Binh, *A survivable design of last mile communication networks using multi-objective genetic algorithms*, Memetic Computing **8** (2016), pp. 97–108.

[5] Kerivin, H. and A. R. Mahjoub, *Design of survivable networks: A survey*, Networks **46** (2005), pp. 1–21.

[6] Magnanti, T. L. and S. Raghavan, *Strong formulations for network design problems with connectivity requirements*, Networks **45** (2005), pp. 61–79.

[7] Steiglitz, K., P. Weiner and D. Kleitman, *The design of minimum-cost survivable networks*, IEEE Transactions on Circuit Theory **16** (1969), pp. 455–460.

[8] Talbi, E.-G., *A taxonomy of hybrid metaheuristics*, Journal of heuristics **8** (2002), pp. 541–564.

[9] Winter, P., *Steiner problem in networks: a survey*, Networks **17** (1987), pp. 129–167.