itcp

Information Theoretic Converse Prover

1.0

07/07/2016

Jayant Apte

Jayant Apte
Email: jayant91089@gmail.com
Homepage: https://sites.google.com/site/jayantapteshomepage/
Address: Department of Electrical and Computer Engineering
Drexel University
Philadelphia, PA 19104

Contents

1 Introduction **3**

2 Installation **4**

3 Usage **5**

3.1 Available functions 5

References **10**

Index **11**

Chapter 1

Introduction

ITCP stands for Information Theoretic Converse Prover. Several converse proofs, like those arising in multi-source network coding over directed acyclic hypergraphs, distributed storage, secret sharing, and graph guessing games etc., can be divided into two parts: 1) What to prove? 2) How to prove? ITCP is a software that allows one to use a computer to generate sensible statements of converse proofs i.e. part 1). The actual proof, i.e. part 2), can then be generated using Yeung's framework [Yeu97] and linear programming duality (see eg. [LWW15], [Tia14]). Currently, it supports the following types of converse proofs:

- Polyhedral Converse (Network Coding): Given a multi-source network coding instance, determine the tightest set of inequalities amoungst source rate and edge capacity variables that are implied by Shannon-type and (if specified) given non-Shannnon-type information inequalities
- Sum-rate Converse (Network Coding): Given a multi-source network coding instance, determine the least upper bound on the sum of source rates, implied by Shannon-type and (if specified) given non-Shannnon-type information inequalities, under specified edge capacities.
- Worst-case information ratio lower bounds (Secret Sharing): Given an access structure, determine the greatest lower bound on the workst case information ratio implied by Shannon-type and (if specified) given non-Shannnon-type information inequalities
- Guessing Number upper bounds (Guessing Games played on directed graphs): Given a directed graph, determine the least upper bound on its guessing number implied by Shannon-type and (if specified) given non-Shannnon-type information inequalities

The last three types of converse proofs can be performed by using linear programming techniques. This involves solving a linear program, whose optimum value gives us an upper (lower) bound. Then, one uses dual optimal vertex to provide a human readable proof that tells us how to combine various information inequalities to arrive at the aforementioned upper (lower) bound. For the first type of converse proof, however, one requires more sophisticated tools. For that purpose, ITCP uses polyhedral projection algorithm symchm, implemented in another GAP package that goes by The same name. In all of the problems mentioned above, one has to either project or solve linear programs over polytopes that have exponentially many dimensions compared to the problem size. A key feature of ITCP is automatic detection of problem symmetries and symmetry exploitation to reduce the ComplexityOfAlgebra of solving the above problems, so that one can approach ever larger problems from a computational perspective.

Chapter 2

Installation

ITCP requires GAP package symchm [Apt16] for symmetry exploiting polyhedral projection, along with package `qsopt_ex-interface` for solving linear programs exactly using rational arithmetic. You can find information about how to install them in their respective user manuals. To get the newest version of ITCP, download the .zip archive from `https://github.com/jayant91089/itcp` and unpack it using

```
unzip itcp-x.zip
```

Do this preferably inside the *pkg* subdirectory of your GAP 4 installation. It creates a subdirectory called *itcp*. This completes the installation of the package. If you do not know the whereabouts of the *pkg* subdirectory, invoke the following in GAP:

Code
```
GAPInfo.("RootPaths");
```

Look for pkg directory inside any of the paths returned. One can then start using ITCP by invoking

gap >

Code
```
LoadPackage( "itcp");
```

from within GAP. This would automatically load the dependencies *symchm* and `qsopt_ex-interface`, so you don't have to load them seperately.

Chapter 3

Usage

3.1 Available functions

In this section we shall look at the functions provided by ITCP.

3.1.1 NetSymGroup

▷ `NetSymGroup(`*`ncinstance`*`)` (function)

Returns: A group

This function finds the Network Symmetry Group of the input network instance bounding the rate region of an instance of mult-source network coding on directed acyclic hypergraphs. It accepts following arguments:

- *ncinstance* is a list $[cons, nsrc, nvars]$ containing 3 elements:
  - *cons* is a list of network coding constraints.
  - *nsrc* is the number of sources.
  - *nvars* is the number of random variables associated with the network.

Returns a subgroup of symmetric group of *nvars* labels

Example
```
gap> # Define a size 8 IDSC instance
> idsc:=[ [ [ [ 1, 2, 3 ], [ 1, 2, 3, 4, 5, 6, 7, 8 ] ],\
>       [ [ 4, 5 ], [ 1, 2, 4, 5 ] ], [ [ 5, 6 ], [ 1, 2, 5, 6 ] ],\
>       [ [ 6, 7 ], [ 1, 2, 6, 7 ] ], [ [ 7, 8 ], [ 1, 2, 7, 8 ] ],\
>       [ [ 4, 8 ], [ 1, 2, 4, 8 ] ], [ [ 4, 6 ], [ 3, 4, 6 ] ],\
>       [ [ 5, 8 ], [ 3, 5, 8 ] ], [ [ 4, 7 ], [ 3, 4, 7 ] ],\
>       [ [ 5, 7 ], [ 3, 5, 7 ] ], [ [ 6, 8 ], [ 3, 6, 8 ] ] ], 3, 8 ];
gap> G:=NetSymGroup(idsc);
Group([ (5,8)(6,7), (4,5)(6,8), (4,6)(7,8), (1,2) ])
gap> Size(G);
20
```

NOTE 1: Certain naming convensions are followed while defining network coding instances. The source messages are labeled with $[1...nsrc]$ while rest of the messages are labeled $[nsrc...nvars]$. Furthermore, the list *cons* includes all network constraints except source independence. This constraint is implied with the labeling i.e. ITAP enforces it by default for the messages labeled $[1...nsrc]$.

3.1.2 NCRateRegionOB

▷ NCRateRegionOB(*ncinstance, usesym, optargs*) (function)

Returns: A list

This function finds the minimal (non-redundant) collection of inequalities bounding the rate region of an instance of mult-source network coding on directed acyclic hypergraphs. It accepts following arguments:

- *ncinstance* is a list $[cons, nsrc, nvars]$ containing 3 elements:
  - *cons* is a list of network coding constraints.
  - *nsrc* is the number of sources.
  - *nvars* is the number of random variables associated with the network.
- *usesym* is a boolean indicating whether symmetry should be used in computation
- *optargs* is a list of optional arguments $[nsrec, ..]$ where *nsrec*, when specified, is a GAP record that specifies any non-Shannon type information inequalities to be included in the computation.

Returns a list $[rr, rrstring]$ where *rr* is a list of normal vectors of the inequalities bounding the rate region, that are inequivalent under the Network Symmetry Group of *ncinstance* if *usesym* is true, otherwise *rr* contains all such normal vectors, under assumption of no symmetries. *rrstring* is a string that can be used to display inequalities associated with *rr* in an easier to read format.

Example
```
gap> # define a network instance (in this case, Fano network)
> F:=[ [ [ [ 1, 2 ], [ 1, 2, 4 ] ], [ [ 2, 3 ], [ 2, 3, 5 ] ],\
>       [ [ 4, 5 ], [ 4, 5, 6 ] ], [ [ 3, 4 ], [ 3, 4, 7 ] ],\
>       [ [ 1, 6 ], [ 3, 1, 6 ] ], [ [ 6, 7 ], [ 2, 6, 7 ] ],\
>       [ [ 5, 7 ], [ 1, 5, 7 ] ] ], 3, 7 ];;
gap> rlist:=NCRateRegionOB(F,true,[]);;
gap> Display(rlist[2]);
0 >= -w2
0 >= -w1
0 >= -w3
+R6 >= +w3
+R5 >= +w3
+R7 >= +w1
+R4 >= +w1
+R6 +R7 >= +w2 +w3
+R4 +R6 >= +w2 +w3
+R4 +R5 >= +w2 +w3
+R6 +R7 >= +w1 +w2
+R4 +R6 >= +w1 +w2
+R4 +R5 >= +w1 +w2
gap> # Define a size 8 IDSC instance with NSG isomorphic to S2 X D5
> idsc:=[ [ [ [ 1, 2, 3 ], [ 1, 2, 3, 4, 5, 6, 7, 8 ] ],\
>       [ [ 4, 5 ], [ 1, 2, 4, 5 ] ], [ [ 5, 6 ], [ 1, 2, 5, 6 ] ],\
>       [ [ 6, 7 ], [ 1, 2, 6, 7 ] ], [ [ 7, 8 ], [ 1, 2, 7, 8 ] ],\
>       [ [ 4, 8 ], [ 1, 2, 4, 8 ] ], [ [ 4, 6 ], [ 3, 4, 6 ] ],\
>       [ [ 5, 8 ], [ 3, 5, 8 ] ], [ [ 4, 7 ], [ 3, 4, 7 ] ],\
>       [ [ 5, 7 ], [ 3, 5, 7 ] ], [ [ 6, 8 ], [ 3, 6, 8 ] ] ], 3, 8 ];
gap> rlist1:=NCRateRegionOB2(idsc,true,[]);;
gap> Display(rlist1[2]);
0 >= -w2
0 >= -w3
+R4 >= 0
+R4 +R6 >= +w3
+R4 +R5 >= +w1 +w2
+R4 +1/2R5 +1/2R8 >= +w1 +w2 +1/2w3
+1/2R4 +1/2R5 +1/2R6 +1/2R7 >= +w1 +w2 +1/2w3
+2/3R4 +2/3R5 +1/3R6 +1/3R8 >= +w1 +w2 +2/3w3
+2/3R4 +1/3R5 +1/3R6 +1/3R7 +1/3R8 >= +w1 +w2 +2/3w3
+1/2R4 +1/2R5 +1/2R6 +1/4R7 +1/4R8 >= +w1 +w2 +3/4w3
+R4 +1/2R5 +1/2R6 +1/2R7 >= +w1 +w2 +w3
+R4 +1/2R5 +1/2R6 +1/2R8 >= +w1 +w2 +w3
+R4 +1/3R5 +1/3R6 +1/3R7 +1/3R8 >= +w1 +w2 +w3
+2/3R4 +2/3R5 +1/3R6 +2/3R7 +1/3R8 >= +w1 +w2 +4/3w3
+R4 +1/2R5 +1/2R6 +R7 >= +w1 +w2 +3/2w3
+R4 +1/2R5 +1/2R6 +1/2R7 +1/2R8 >= +w1 +w2 +3/2w3
+2R4 +R6 +R7 >= +w1 +w2 +2w3
+R4 +R5 +R6 +R7 >= +w1 +w2 +2w3
```

NOTE 2: Currently ITCP supports specifiation of non-Shannon ineqality of Zhang and Yeung [ZY97] and 214 new non-Shannon inequalities found by Dougherty, Freiling and Zeger [DFZ11]. These inequalities can be applied to any collection of 4 subsets of a given set of random variables. The keys of the record *nsrec* can be integers $1,\ldots,215$ where index 1 corresponds to the ZY inequality while $2,\ldots,215$ correspond to the inequalities found by Dougherty, Freiling and Zeger. The value associated wth each key is a list of 4-subsets of subsets of *nvars* random variables associated with the network. Internally, several permuted forms of these inequalities are created, so as to not break the symmetries of the network coding instance, that are exploited during the computation.

3.1.3 NCSumRateUB

▷ `NCSumRateUB(ncinstance, caps, optargs)` (function)

Returns: A rational number

This function finds least upper bound on the sum of all source rates that is implied by the Shannon-type and (if specified) the non-Shannon-type inequalities. It accepts following arguments:

- *ncinstance* is a list $[cons, nsrc, nvars]$ containing 3 elements:
  - *cons* is a list of network coding constraints.
  - *nsrc* is the number of sources.
  - *nvars* is the number of random variables associated with the network.
- *caps* is a list of $nvars - nsrc$ non-negative integers, specifying the capacity of the edges
- *optargs* is a list of optional arguments $[nsrec, ..]$ where *nsrec*, when specified, is a GAP record that specifies any non-Shannon type information inequalities to be included in the computation (see NOTE 2 for explanation of the format).

Returns a rational number specifying the upper bound

```
Example
gap> # define a network instance
> N:= [[ [ [ 1 ], [ 1, 3 ] ], [ [ 1 ], [ 1, 4 ] ],[ [ 1, 2, 5 ],\
>       [ 1, 2 ] ],[ [ 1, 2, 3 ], [ 2, 3 ] ],[ [ 2, 4 ], [ 1, 2, 4 ] ],\
>       [ [ 2, 3, 4, 5 ], [ 3, 4, 5 ] ]] , 2, 5 ];;
gap> ub:=NCSumRateUB(N,[1,1,1],[]);;
Original LP dimension...28
LP dimension after considering symmetries...22
gap> ub;
2
```

3.1.4 SSWorstInfoRatioLB

▷ `SSWorstInfoRatioLB(Asets, nvars, optargs)` (function)

Returns: A rational number specifying the lower bound

This function finds the greatest lower bound on the worst case information ratio of a secret sharing instance.

- *Asets* - A list of authorized sets each specified as a subset of $[nvars-1]$
- *nvars* - Number of participants (including one dealer)
- *optargs* is a list of optional arguments $[nsrec,..]$ where *nsrec*, when specified, is a GAP record that specifies

any non-Shannon type information inequalities to be included in the computation (see NOTE 2 for explanation of the format). Returns a rational number

```
Example
gap> # define an access structure
> Asets:=[[2,3],[3,4],[4,5]];;
gap> lb:=SSWorstInfoRatioLB(Asets,5,[]);;
Original LP dimension...20
LP dimension after considering symmetries...12
gap> lb;
3/2
```

NOTE 3: No input checking is performed to verify if input *Asets* follows labeling convensions.

3.1.5 GGnumberUB

▷ `GGnumberUB(Asets, nvars, optargs)` (function)

Returns: A rational number specifying the upper bound

This function finds the least upper bound on the guessing number of a directed graph.

- *G* - A list with 2 elements: 1) a list of vertices 2) a GAP record each vertex of the graph to a a list neighbors with edges incoming to it
- *optargs* is a list of optional arguments $[nsrec,..]$ where *nsrec*, when specified, is a GAP record that specifies

any non-Shannon type information inequalities to be included in the computation (see NOTE 2 for explanation of the format). Returns A rational number

Example

```
gap> # define a directed graph (in this case, the cycle graph C5)
> C5:=[ [ 1, 2, 3, 4, 5 ],\
>   rec( 1 := [ 2, 5 ], 2 := [ 1, 3 ], 3 := [ 2, 4 ],\
>       4 := [ 3, 5 ], 5 := [ 4, 1 ] ) ];;
gap> ub:=GGnumberUB(C5,[]);;
Original LP dimension...25
LP dimension after considering symmetries...5
gap> ub;
5/2
```

NOTE 4: Certain naming convensions are followed while graph G. The vertices are labeled by the set $\{1,...,n\}$ where n is the number of vertices of the graph. Hence, the keys of the record G are integers $\{1,...,n\}$. The value for a key t is a list $[i, j, ...]$ where $i, j, ...$ are verties s.t. $(i,t),(j,t),...$ are the directed edges of the graph.

References

[Apt16] Jayant Apte. symchm - A GAP4 package for symmetry-exploiting polyhedral projection, 2016. http://www.ece.drexel.edu/walsh/aspitrg/software.html. 4

[DFZ11] R. Dougherty, C. Freiling, and K. Zeger. Non-shannon information inequalities in four random variables. *CoRR*, abs/1104.3602, 2011. 7

[LWW15] Congduan Li, Steven Weber, and John MacLaren Walsh. On multi-source networks: Enumeration, rate region computation, and hierarchy. *CoRR*, abs/1507.05728, 2015. 3

[Tia14] Chao Tian. Characterizing the rate region of the (4, 3, 3) exact-repair regenerating codes. *IEEE Journal on Selected Areas in Communications*, 32(5):967–975, 2014. 3

[Yeu97] R. W. Yeung. A framework for linear information inequalities. *IEEE Transactions on Information Theory*, 43(6):1924–1934, Nov 1997. 3

[ZY97] Zhen Zhang and R.W. Yeung. A non-shannon-type conditional of information quantities. *Information Theory, IEEE Transactions on*, 43(6):1982–1986, Nov 1997. 7

Index

itcp, 3

GGnumberUB, 8

NCRateRegionOB, 6
NCSumRateUB, 7
NetSymGroup, 5

SSWorstInfoRatioLB, 8