NAME

dw – utility function for interacting with D-Wave System

SYNOPSIS

dw {dw-subcommand} ...

DESCRIPTION

Provides a general-purpose interface for interacting with the D-Wave System. Manages connections, solvers and workspaces. Displays geometry data. Builds and executes quantum machine instructions (QMIs). Embeds QUBOs to create parameterized quantum machine instructions (PQMIs). Binds and validates PQMIs. Provides simple constraint to QUBO compilation. Includes a front-end interface for qbsolv.

Many **dw** subcommands set and display environment variables used to access the D-Wave System. These variables are specific to the current shell and its children. The **dw** command is implemented partially via Bash shell functions and thus requires the use of the Bash shell.

Interactive Bash shells must source the definition of the **dw** command from $DWAVE_HOME/bin/dw_setup. Place the following command in the Bash shell start-up file (for example ~/.bashrc):

```
source $DWAVE_HOME/bin/dw_setup
```

Non-interactive shells (such as ones that are created to execute shell scripts) must also read the definition of the **dw** command. To do this, place this command in the start-up file:

```
export BASH_ENV=$DWAVE_HOME/bin/dw_setup
```

This is suitable if no other logic must be included during shell start-up.

If there is additional logic that must be executed at start-up time of a non-interactive shell, then all logic must be placed in a single environment file and the location of that file should be exported as the value for BASH_ENV. In that case, the command to source $DWAVE_HOME/bin/dw_setup should be placed into the environment file.

SUBCOMMANDS

dw convert

File formats and naming conventions changed in moving from qOp version 2.2 to 2.3. If you have installed a version of qOp prior to 2.3, then run **dw convert** to update your existing workspaces.

dw format {*file.qmi*|*file.sol*|*file.epqmi*}

This command will report the format of a file in the current workspace. For qOp 2.3 (resp. 2.2) the format is "3" (resp. "2"). The **dw format** command only works with the three file types as specified by the suffixes listed above.

dw get ...

Displays the value for one of the following configuration variables:

connection – **dw get connection** displays the currently active connection if one has been set. If there is no current connection, the available connections as listed in the .dwrc file are listed and the user can choose to select one of them.

solver – **dw get solver** displays the currently active solver if one has been set. If there is no current solver and the connection has been set, the available solvers are listed and the user can choose to select one of them.

workspace – **dw get workspace** displays the currently active workspace. The top directory of the workspace is located within $DWAVE_WORKSPACE and its name is given by concatenating "workspace." with the workspace name. For example, the 128-qubit simulator has a workspace at $DWAVE_WORKSPACE/workspace.c4 since the name of this workspace is "c4".

env – **dw get env** displays the values for environment variables whose values are manipulated by **dw**. They are:

DW_INTERNAL__WORKSPACE

DW_INTERNAL__CONNECTION

DW_INTERNAL__WSPATH

DW_INTERNAL__HTTPLINK

DW_INTERNAL__HTTPPROXY

DW_INTERNAL__SOLVER

DW_INTERNAL__TOKEN

The DW_INTERNAL__ prefix is stripped from each of these environment variables before display.

path – **dw get path** displays the Unix path to the current workspace directory. The workspace directory is independent of the current working directory for the shell. The workspace directory is intended to hold binary file formats for QMI, PQMI and SOL files as well as other file formats used by **dw** commands. These files generally work within a single geometry only and so workspaces are a simple mechanism for helping to segregate file formats by geometry.

geometry – **dw get geometry** displays the current geometry. The Chimera geometry of the D-Wave System is characterized by three parameters:

L – half the number of qubits in the unit cell

M – number of rows of unit cells in the Chimera grid

N – number of columns of unit cells in the Chimera grid

For the 128-qubit simulator, these parameters all have the value 4. The total number of qubits and couplers and active qubits and couplers is also displayed.

qubits – **dw get qubits** displays a bit mask for the set of all qubits in the current geometry. The bit mask contains a 1 for active qubits and 0 for inactive qubits.

couplers – **dw get couplers** displays a bit mask for the set of all couplers in the current geometry. The bit mask contains a 1 for active couplers and 0 for inactive couplers.

Qnnnn – **dw get Qnnnn** displays connection information for a particular qubit. The qubit number, which must be a four-digit zero-padded value, is a linear index starting at 0. The display shows whether the qubit is active or not and displays the Chimera index for the qubit. The qubit neighbors are displayed along with the names and statues of the couplers connecting the qubit to its neighbors.

Cnnnn – **dw get Cnnnn** displays the connection information and active/inactive status for a particular coupler. The two qubits

connected by the coupler are displayed along with their status and Chimera index.

param – **dw get param** displays the declared parameters and indicates which parameter is current.

variable – **dw get variable** lists the declared variables.

assert – **dw get assert** lists the declared assertions. Each assertion is given a label of the form Adddd. The first assertions following **dw init** is numbered A0000 and the numeric part of the label is incremented by one for each successive assertion. The form **dw get assert Adddd** lists detailed information about the specified assertion. This includes the active parameter, the active variables and the truth table for all possible states of the assertion. A value of 1 (resp. 0) means that a state is valid (resp. invalid).

macro – **dw get macro** lists the built-in and user-defined macros available within the **dw set assert** command. See **dw-macro(1)** for more information.

qubo – **dw get qubo** displays a .qubo file from the workspace. The .qubo file is created as a side-effect of **dw embed**. The .qubo file lists the parameters, variables, terms and asserts which are parsed from a .q file. With no arguments, **dw get qubo** displays the default .qubo file (if one exists) in the workspace directory. With a single argument, **dw get qubo *name.qubo*** displays the specified .qubo file.

embedding – **dw get embedding** displays the embedding created as a side-effect of **dw embed**. Each logical variable in the .q file maps to a connected set of qubits which is listed following the variable name. With no arguments, **dw get embedding** displays the default.epqmi file (if one exists) in the workspace directory. With a single argument, **dw get embedding *name.epqmi*** displays the specified file. The .epqmi suffix indicates that the file contains both an embedding (e) and a parameterized QMI (pqmi).

pqmi – **dw get pqmi** displays the PQMI created by **dw embed**. The PQMI display contains five sections:

PQMI Header Section – L, M and N parameters in geometry

PQMI Active Qubit Mask – Active qubits in geometry

PQMI Parameter Section – Parameters in the PQMI

PQMI Coefficient Section – Qubit weights, Coupler strengths

PQMI Validation Section – Validation expressions

These sections are all contained within the binary file format of the PQMI and are produced by **dw embed.**

*.qmi – **dw get [fmt=qubist] *name*.qmi** displays a QMI in a simple text format. If the optional fmt=qubist argument is provided, an alternate format is generated. This format can be pasted directly into the Data sheet of Qubist.

*.sol – **dw get [-s NNN] *name*.sol** displays a solution in a SOL file. If the –s argument is omitted, all solutions are displayed. Otherwise, the specified solution is displayed. Solutions are numbered starting from 1.

version – **dw get version** displays the version numbers for the components of the qOp package.

dw set ...

Allows the user to set the connection and solver, to create a QMI or to build a set of parameters, variables and assertions for the constraint compiler.

connection – **dw set connection *[c-name]*** sets the connection to a specified option from the ~/.dwrc file. With no arguments, the connection is unset.

solver – **dw set solver *[s-name]*** sets the solver to a specified option supported by the current connection. With no arguments, the solver is unset.

*.qmi – **dw set [fmt=qubist] *name.qmi*** reads standard input, which is expected to contain a text representation of a QMI. Each line of the input should match one of the four following patterns:

SCALE_FACTOR <== fff.fff

CONST <== fff.fff

Qnnnn <== fff.fff

Cnnnn <== fff.fff

The lines may be in any order. The lines are parsed to create a QMI which is written to the current workspace with the specified name. If the optional fmt=qubist argument is provided, an

alternate input format is used. The format is the same as that used on the Data sheet of Qubist.

param – **dw set param *param-name*** declares a parameter with name ***param-name*** and makes that parameter current. Since the **dw init** command clears the current parameter, there is no current parameter until the next **dw set param** command has been issued. It is not an error to issue a **dw set param** command more than once for the same parameter. This is useful when managing multiple assertions that are tied to the same parameter.

variable – **dw set variable *variable-name*** declares a variable with name ***variable-name***. It is an error to declare the same variable more than once. Each variable is assumed to take the value 0 or 1.

assert – **dw set assert *assertion*** declares an assertion ***assertion***, which is a shell expression over declared variables. The assertion is evaluated for each possible assignment of 0/1 values to its variables. If the expression evaluates to 0 (resp. 1) the assertion is valid (resp. invalid). The assertion is tied to the current parameter. This provides a mechanism for independently strengthening or weakening QUBO terms associated to each assertion. In addition to shell expressions, ***assertion*** can invoke a macro. Built-in and user-defined macros are available. See **dw-macro(1)** for more information.

macro – ***dw set macro macro-name macro-expr*** defines a new user-defined macro for use in subsequent **dw set assert** statements. See **dw-macro(1)** for more information.

dw cd

dw cmp

dw cp

dw ls

dw mkdir

dw mv

dw pwd

dw rm

dw rmdir

dw touch

dw cat

dw find

These are workspace management commands. All of them function in exactly the same way as their Unix counterparts, except that they operate on the current workspace directory rather than the shell's current working directory.

After setting the connection and solver for the 128-qubit simulator, **dw** is configured to use the workspace located at $DWAVE_WORKSPACE/workspace.c4. This is true regardless of the current working directory in the shell when the **dw set connection** and **dw set solver** commands were issued. To create a subdirectory under the $DWAVE_WORKSPACE/workspace.c4 use **dw mkdir *dir-name***. To change the current workspace to that directory, use **dw cd *dir-name***. List the contents of the current workspace directory with **dw ls**. Options such as -l for the Unix **ls** command can also be used with **dw ls**. Files in the workspace directory can be renamed, copied and deleted with **dw mv**, **dw cp** and **dw rm**. Remove workspace directories with **dw rmdir**. An empty file can be created in the workspace directory via **dw touch**. The path relative to the top of the workspace is displayed via **dw pwd**. **Dw cat** writes the contents of the named file(s) in the workspace to standard output, which makes it easy to copy a file out of the workspace. **Dw find** runs in the workspace directory, making it easy to locate files in the workspace without having to navigate to that directory.

dw init

The **dw init** command prepares the current workspace for a compilation session by deleting all temporary files which are created by **dw set param**, **dw set variable**, **dw set assert** and **dw set macro** commands. All four of these commands write temporary files into the current workspace. If the current workspace is changed during a compilation session, then consistency across the files may be lost.

dw compile

The **dw compile** command activates the compilation stage of the constraint compiler. All parameters, variables and assertions are converted into a QUBO which is then written to standard output. Typically one will initiate a compilation session with

the **dw init** command followed by a sequence of **dw set param**, **dw set variable** and **dw set assert** commands. After all parameters, variables and assertions have been declared, issue the **dw compile** command to create a QUBO which represents the entire problem. The output from **dw compile** will usually be captured in a .q file.

dw embed [-r seed] ***input.q [embedding.e] [-c chain_parameter] [-o output.epqmi]***

The **dw embed** command requires as input a .q file, which specifies a constrained QUBO. If no embedding is provided, **dw embed** attempts to map each QUBO variable to a connected set of qubits in the current Chimera architecture. An embedding can be provided, causing **dw embed** to skip the step of generating its own embedding. In either case, a PQMI is formed from the QUBO and the embedding. The PQMI contains expressions to compute the coefficients for all qubits and couplers used by the embedding. The embedding and PQMI are written together in a single binary file with .epqmi suffix in the current workspace. If no output is specified, the file is named default.epqmi. Otherwise, the name provided via the optional **-o *output.epqmi*** argument is used. Embedding uses a random number generator, so each call to **dw embed** will generally result in a distinct embedding and PQMI. To ensure repeatability, use the optional **-r *seed*** argument which specifies a seed for the random number generator used during embedding. If the embedding requires multi-qubit chains for any variable, a chain strength parameter named param_chain is included in the PQMI. To change the name of the chain strength parameter, use the optional **-c *chain_parameter*** argument. The file format of the .q file is described elsewhere.

dw qbsolv ***input.q [-o output.pqb]***

The **dw qbsolv** command requires an input a .q file, which defines a constrained QUBO. The QUBO is prepared for execution with qbsolv, which is alternative execution method useful for problems that are too large or dense to embed into a QMI. The output file, if specified, should have suffix .pqb which stands for Parametrized QuBo. If no output is specified, the name of the output will be taken from the input .q file. The user can override this default by providing the **-o output.pqb** argument. In either case the output is written into the current workspace. The output file contains the same set of parameters defined in the .q file and so it must be bound via **dw bind** before execution.

dw bind [param1=val1 [param2=val2 [...]]] [file1.b [file2.b [...]]]

The **dw bind** command gives values to the parameters of a EPQMI or PQB file. If only one of the two file types (PQMI or PQB) exists, then that file is taken as the input for the **dw bind** command. If both exist, then the newer of the two file types is used. If **dw bind** uses a PQMI file, the output of **dw bind** is a QMI which is ready for execution. If **dw bind** uses a PQB file, the output is a qbsolv job. The output of binding an EPQMI file is a .qmi file. The output of binding a .pqb file is a .qbi file.

The parameters appearing in a PQMI arise in two ways. First, the underlying QUBO lists parameters whose values should be specified before execution. Second, most QUBOs require multi-qubit chains when embedded into Chimera. The validity of these chains is affected by the chain strength parameter, whose default name is param_chain and which may be overridden by the optional -c argument to **dw embed**. Only PQMI jobs use the chain strength parameter. Qbsolv jobs only use parameters specified in the underlying QUBO.

Values can be provided on the command line by providing an argument of the form param=val with no intervening spaces. Values can also be provided in a bind file (*.b), each line of which has the same param=val format. It is permissible to mix command argument and bind file specification of parameter values. Command arguments and bind files are processed in the order listed. If a parameter appears more than one time in command arguments or bind files, its last appearance takes precedence. The **dw bind** command always operates on the EPQMI or PQB file with name default.epqmi or default.pqb in the current workspace.

dw exec [-t] [num_reads=NNN] [-p {optimize|sample}] [-c] [beta=BBB] [annealing_time=TTT] {file1.qmi *[file2.qmi [...]]|file.qbi}*

The **dw exec** command executes one or more QMIs or one qbsolv job on any connection and solver. The QMI files are generally created via either **dw set *name.qmi*** or by binding an EPQMI file which was created via **dw embed**. A qbsolv job is usually created by binding a PQB file which was created by **dw qbsolv**. The output from **dw exec** is one .sol output file for each .qmi input file or one .qbo file for the .qbi input file.

The following options apply only to QMI jobs: Specify –t to display timing output. If the **num_reads** option does not appear on the command line, the default number of samples read for each QMI is 10. Use –p followed by either optimize or sample to invoke post-processing, if supported by the current solver. Specify –c to enable chain fix-up during post-processing. Set the beta value by providing beta=BBB as an argument for those solvers that support sampling. Set the annealing time by providing annealing_time=TTT with TTT specified in microseconds.

dw val [-v] [-s nnn] {file.sol|file.qbo}

The **dw val** command applies validation criteria to one or more solutions from a .sol or .qbo file and reports on the results. A QMI that has been generated directly from **dw set *name.qmi*** has no validation criteria and so **dw val** should not be used. An EPQMI generated via **dw embed** contains validations and so **dw val** uses validations from the default.epqmi file in the current workspace. A PQB generated via **dw qbsolv** contains validations and so can also be used with **dw val**.

With no argument, four totals are displayed for .sol files:

Total samples

Distinct samples

Total valid

Distinct valid

With the –v argument, all validation criteria are evaluated for each solution. Solutions meeting all validation criteria are marked valid, and solutions failing at least one validation criteria are marked invalid. The number of occurrences of each solution and energy are displayed for each solution. With the –s argument, the information displayed is limited to a single solution. Solutions are numbered starting with 1.

For .qbo files, displayed information is limited to the solution (interpreted according to the logical variables in the underlying QUBO), overall validation status, objective value, and the result of each validation criteria.

dw add sum.qmi file1.qmi *[file2.qmi [...]]*

The **dw add** command adds one or more QMIs from the current workspace and writes the sum QMI back to the current workspace. Note that adding two or more QMIs simply means adding the corresponding weights and strengths. For example, if file1.qmi contains a weight of 4.7 for Q0123 and file2.qmi contains a weight of -2.3 for Q0123, then the sum QMI will have a weight of 2.4 for Q0123.

dw trans transform.t transformed.qmi original.qmi

dw trans transform.t original.sol transformed.sol

The **dw trans** command implements both graph and gauge transformations on either QMI or SOL files. Transformations are specified via a simple text format file with suffix .t. Each line of the transformation file specifies a target and source qubit as follows:

```
Q0000 <== Q0001
```

This line means that source qubit Q0001 in the original QMI will be sent to target qubit Q0000 in the transformed QMI. To specify a gauge transformation in addition to the qubit mapping, include a tilde symbol on the right hand side of the assignment arrow:

```
Q0000 <== ~Q0001
```

This means that the 0 (resp. 1) value of qubit Q0001 will correspond to the 1 (resp. 0) value of qubit Q0000 in the transformed problem.

When applied to a QMI file, weights and strengths move from source to target. For example, if the first specification line above appears in the transform file, then the weight for qubit Q0001 in original.qmi will match the weight for qubit Q0000 in transformed.qmi.

When applied to an SOL file, qubit values move from target to source. For example, if qubit Q0000 is 1 in the seventh solution in transformed.sol, then qubit Q0001 will be 1 in the seventh solution in original.sol.

dw trans moves qubit weights and coupler strengths (from QMI files) in the opposite direction to qubit values (from SOL files) for the following reason: we can either execute the original QMI directly, or transform it with **dw trans**, execute the transformed QMI, and then transform the solutions again with **dw trans**. The

latter route employs symmetry averaging to eliminate hardware bias.

dw trans expects a single .t file argument, which is the transform file. This is a text file and resides in the local directory. If **dw trans** has two .qmi file arguments, the first is treated as the transformed QMI and the second is treated as the original QMI. If **dw trans** has two .sol file arguments, the first is treated as the output SOL and the second is the input SOL. Both .qmi and .sol files reside in the remote workspace.

EXAMPLE

The first example is a complete Bash shell script that uses **dw** to create a connection, obtain a solver, define a QMI, execute the QMI, and examine the solutions:

```
#!/bin/bash

dw set connection laptop

dw set solver c4-sw_sample

dw set first.qmi <<EOF

    Q0000 <== 5

    Q0004 <== 5

    C0000 <== -10

EOF

dw exec first.qmi

dw get first.sol
```

The next example shows the naming and creation of a new workspace in **dw**. A new workspace is created whenever the user sets a connection and solver which has a distinct geometry from any that the user has worked with before. This may happen either via the **dw set solver ...** or **dw get solver** commands. In the following snippet taken from a shell session, the user invokes **dw get solver**, and **dw** responds by listing the one available solver. The user chooses solver **DW2X_SYS4**, and **dw** determines that this is a new geometry. **dw** responds by prompting the user for a new workspace name. The user provides the name sys4 and the new

workspace is created using that name. Finally, the solver is set to DW2X_SYS4:

```
Documents> dw get solver

VALUE('solver') IS UNSET

Available solvers:

1) DW2X_SYS4

SELECT A SOLVER (OR Ctrl-D TO SKIP SELECTION): 1

NO WORKSPACE ASSOCIATED WITH SOLVER 'DW2X_SYS4'

ENTER A NEW WORKSPACE NAME OR HIT <ENTER> TO ABORT: sys4

CREATED WORKSPACE 'workspace.sys4

SETTING VALUE('solver') TO 'DW2X_SYS4'
```

The third example demonstrates usage of the constraint compiler in **dw**. The example declares a single parameter, four variables and two assertions. It compiles and produces an output QUBO, which can be embedded or solved using qbsolv:

```
dw init

dw set param p

dw set variable x

dw set variable y

dw set variable z

dw set variable w

dw set assert 'x + y + z - 1'

dw set assert 'z * w'

dw compile
```

The QUBO output is generated by the **dw compile** command. All earlier commands record their results in the current workspace directory. The **dw** commands for establishing the connection, solver and workspace directory determine the location of the temporary files written by the **dw set** commands.

BUGS

Please report bugs to dwsupport@dwavesys.com.

COPYRIGHT

SEE ALSO

The file format of **dw** QUBO files is described in Q.pdf. Built-in macros and user-defined macros for the **dw** constraint compiler are described in DW-MACRO.pdf.