NAME

DW_epqmi_read, DW_epqmi_list_params, DW_epqmi_list_vars, DW_epqmi_bind, DW_epqmi_exec, DW_epqmi_sols, DW_epqmi_sol_vars, DW_epqmi_sol_occurs, DW_epqmi_sol_obj, DW_epqmi_free

LIBRARY

dw library (libepqmi.a, -lepqmi)

SYNOPSIS

```
#include <epqmi.h>

DW_epqmi *DW_epqmi_read(char *epqmi_file);

int DW_epqmi_list_params(DW_epqmi *epqmi, char ***param_names,
int *params);

int DW_epqmi_list_vars(DW_epqmi *epqmi, char ***var_names, int
*vars);

int DW_epqmi_bind(DW_epqmi *epqmi, float *param_values);

int DW_epqmi_exec(DW_epqmi *epqmi, int num_reads);

int DW_epqmi_sols(DW_epqmi *epqmi, int *solutions);

int DW_epqmi_sol_vars(DW_epqmi *epqmi, int solnum, char *var, int
*valid);

int DW_epqmi_sol_occurs(DW_epqmi *epqmi, int solnum, int
*occurrences);

int DW_epqmi_sol_obj(DW_epqmi *epqmi, int solnum, float
*objective);

int DW_epqmi_free(DW_epqmi *epqmi);
```

OVERVIEW

The DW_epqmi library contains functions designed to provide a simple interface for running pre-embedded parameterized QUBOs on the D-Wave simulator and hardware. DW_epqmi is an opaque type whose definition contains a single ‘magic’ field. The only valid way to allocate a DW_epqmi object is to use the DW_epqmi_read

function, which returns a pointer to a properly allocated object of this type.

When DW_epqmi_read is called, the environment of the current process is examined for environment variables which contain the current setting of the dw workspace. The workspace should contain a pre-embedded parameterized QUBO in the form of a .epqmi file. For details on this process, see dw(1).

DW_epqmi_read reads the QUBO and prepares it for execution. The QUBO is defined over a set of variables, each of which has a name. Use DW_epqmi_list_vars to retrieve a list of the variable names from the QUBO. The QUBO may have parameters whose value must be specified before the QUBO can be executed. Use DW_epqmi_list_params to retrieve the parameters. Before execution, assign values to all parameters using DW_epqmi_bind. Execute the embedded QUBO using DW_epqmi_exec. Note that the library is single threaded – once parameter values have been assigned, a single generated QMI is ready for execution.

The result of executing the generated QMI can be inspected via DW_epqmi_sols which returns the number of distinct solutions, DW_epqmi_sol_vars which returns values of the variables over which the QUBO is defined (not qubit values), DW_epqmi_sol_occurs which returns the number of times a particular solution appeared, and DW_epqmi_sol_obj which returns the objective value for a solution.

Multiple executions are possible. If the current set of parameter values does not need to be updated, repeat the call of DW_epqmi_exec without calling DW_epqmi_bind. If the current set of parameter values needs to be updated before the next execution, call DW_epqmi_bind.

DESCRIPTION

DW_epqmi *DW_epqmi_read(char *epqmi_file);

Use DW_epqmi_read(...) to create a DW_epqmi structure by reading information from the current dw workspace. If the epqmi_file argument is a NULL pointer, assume the epqmi file is named "default.epqmi". Otherwise, assume that epqmi_file names an epqmi file in the current workspace. This returns NULL if there is a problem and a non-NULL pointer if all is successful.

int DW_epqmi_list_params(DW_epqmi *epqmi, char ***param_names, int *params);

Use DW_epqmi_list_params(...) to list the parameters in the epqmi. The param_names argument should be the address of a char ** variable. The variable's value will be overwritten with the address of an array of pointers to char. The params argument should be the address of an int which will be overwritten with the number of parameters in the underlying QUBO. At bind time, a value must be specified for each parameter and the order of parameter values must match the order of the parameter names returned by this function.

int DW_epqmi_list_vars(DW_epqmi *epqmi, char ***var_names, int *vars);

Use DW_epqmi_list_vars(...) to list the variables in the epqmi. The var_names argument should be the address of a char ** so that the variable's value can be overwritten with the address of an array of pointers to char. The vars argument should be the address of an int, which will be overwritten with the number of variables in the underlying QUBO. After execution, the DW_epqmi_sol_vars function reports the value of each variable in each sample. The ordering of variables reported by that function matches the ordering of variable names returned by this function.

int DW_epqmi_bind(DW_epqmi *epqmi, float *param_values);

Use DW_epqmi_bind(...) to assign a value to each parameter in an epqmi and create a QMI. The parameter values are provided in an array whose address is the second argument to the function. The order of parameter values in the array corresponds to the order of parameter names returned by DW_epqmi_list_params(...).

int DW_epqmi_exec(DW_epqmi *epqmi, int num_reads);

Use DW_epqmi_exec(...) to execute the QMI created by DW_epqmi_bind(...). Specify the number of samples via the num_reads argument.

int DW_epqmi_sols(DW_epqmi *epqmi, int *solutions);

Use DW_epqmi_sols(...) after executing a QMI to determine the number of unique samples in the distribution. The solutions argument is a pointer to a variable defined by the calling program whose value will be overwritten with the number of distinct samples.

int DW_epqmi_sol_vars(DW_epqmi *epqmi, int solnum, char *var, int *valid);

Use DW_epqmi_sol_vars(...) after executing a QMI to determine the variable values in each variable in a specific solution. Solnum is the solution number. Var is a pointer to an array allocated by the calling program whose size in bytes equals the number of variables. This array will be overwritten with the 0 and 1 values from the specified solution. Additionally, the assertions associated with the epqmi will be evaluated. If all assertions are true, the valid variable will be overwritten with 1, otherwise it will be overwritten with 0.

int DW_epqmi_sol_occurs(DW_epqmi *epqmi, int solnum, int *occurrences);

Use DW_epqmi_sol_occurs(...) after executing a QMI to determine how many times a specific sample occurs. Solnum names the solution and the number of occurrences will be overwritten into the variable whose address is passed in the occurrrences argument.

int DW_epqmi_sol_obj(DW_epqmi *epqmi, int solnum, float *objective);

Use DW_epqmi_sol_obj(...) to get the objective value of a specific sample. The objective argument should be the address of a value defined in the calling program which will be overwritten with the objective of the specified solution.

int DW_epqmi_free(DW_epqmi *epqmi);

Use DW_epqmi_free(...) to free the storage associated with an eqpqmi.

RETURN VALUES

DW_epqmi_read returns a valid pointer to DW_epqmi if successful and a NULL pointer otherwise. All other functions in the library return 0 if successful and a non-zero value otherwise. In case of error, an message may be written to stderr. All functions in the library with the exception of DW_epqmi_read check their first argument. If this argument is not a valid pointer to DW_epqmi or is NULL, the function will return 1 and no error message will be displayed.

EXAMPLE

This C program uses the dw library to open the default epqmi in the current workspace, list its parameters and variables, assign

random values to the parameters, execute the generated QMI and examine the generated samples:

```
#include <stdio.h>
#include <stdlib.h>

#include "epqmi.h"

int main(int argc, char *argv[])
{
  DW_epqmi *epqmi;

  epqmi = DW_epqmi_read(NULL);

  if (epqmi == NULL)
    {
      printf("error: epqmi is NULL\n");
      return 1;
    }

  char **param_names;
  int params;
  DW_epqmi_list_params(epqmi, &param_names, &params);

  int i;
  for (i=0; i<params; ++i)
     printf("param[%d] is \"%s\"\n", i, param_names[i]);

  float *param_values;
  param_values = (float *) malloc( params * sizeof(float) );
  for (i=0; i<params; ++i) param_values[i] = (-10 + rand() % 20);

  DW_epqmi_bind(epqmi, param_values);

  char **var_names;
  int vars;
  DW_epqmi_list_vars(epqmi, &var_names, &vars);

  for (i=0; i<vars; ++i)
     printf("var[%d] is \"%s\"\n", i, var_names[i]);

  if (DW_epqmi_exec(epqmi, 100))
    {
      fprintf(stderr, "execution error\n");
      return 1;
    }

  int solutions;
  if (DW_epqmi_sols(epqmi, &solutions))
    {
      fprintf(stderr, "solution error\n");
```

```
      return 1;
    }
  printf("**** num_solutions=%d ****", solutions);

  char *var = malloc(vars);

  int s;
  for (s=0; s<solutions; ++s)
    {
      printf("**** solution=%d ****\n", s);
      int valid;
      if (DW_epqmi_sol_vars(epqmi, s, var, &valid))
        {
          fprintf(stderr, "variable error\n");
          return 1;
        }
      printf("     valid=%d\n", valid);
      int occurrences;
      DW_epqmi_sol_occurs(epqmi, s, &occurrences);
      float obj;
      DW_epqmi_sol_obj(epqmi, s, &obj);
      printf("     occurs=%d\n", occurrences);
      printf("     objective=%g\n", obj);
      for (i=0; i<vars; ++i)
         printf("%10s <== %d\n", var_names[i], var[i]);
    }

  return 0;
}
```

BUGS

Please report bugs to dwsupport@dwavesys.com.

COPYRIGHT

SEE ALSO

dw(1)