Arrow

2.0

Generated by Doxygen 1.5.5

Fri Jul 4 13:27:16 2008

Contents

Chapter 1

Arrow Callable Library

1.1 Introduction

Arrow is one part callable library, one part collection of programs for solving the bottleneck traveling salesman problem and other closely related problems.

It's still very much under development, but someday a stable release will be present that will include better documentation that what's provided here.

1.2 Installation

Is still super hard. Sorry about that. Will describe it someday.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Module Documentation

5.1 Binary Programs

Files

- file 2mb.c

  2-Max Bound solver.

- file abtsp.c

  Asymmetric Bottleneck TSP heuristic.

- file bap.c

  Bottleneck Assignment Problem solver.

- file bbssp.c

  Bottleneck Biconnected Spanning Subgraph solver.

- file bscssp.c

  Bottleneck strongly connected spanning subgraph problem solver.

- file btsp.c

  Bottleneck TSP heuristic.

- file cbtsp.c

  Constrained Bottleneck TSP heuristic.

- file dcbpb.c

  Degree Constrained Bottleneck Path Bound solver.

- file hash.c

  Hash testing.

- file histogram_data.c

  Edge length histogram data collector.

- file linkern.c

  Lin-Kernighan TSP heuristic.

- file subproblem.c

  Sub-problem generator.

- file tour_info.c

  Tour information.

- file tsp.c

  Traveling Salesman Problem solver.

5.2 Callable Library

Files

- file 2mb.c

  2-max bound implemenation.

- file arrow.h

  Header file for the Arrow callable library.

- file bap.c

  Bottleneck assignment problem (BAP) implemenation.

- file bbssp.c

  Bottleneck biconnected spanning subgraph problem implemenation.

- file bintree.c

  Binary tree implementation.

- file bscssp.c

  Bottleneck strongly connected spanning subgraph problem implemenation.

- file btsp.c

  Bottleneck traveling salesman problem (BTSP) methods.

- file btsp_fun.c

  Cost matrix transformation functions.

- file dcbpb.c

  Degree constarined bottleneck paths bound.

- file options.c

  Helper for parsing program options.

- file problem.c

  Functions for working with problem data.

- file tsp.c

  TSP solver and Lin-Kernighan heuristic.

- file util.c

  Useful utility functions.

Chapter 6

Data Structure Documentation

6.1 arrow_bintree Struct Reference

Binary tree data structure.

`#include <arrow.h>`

Collaboration diagram for arrow_bintree:

Data Fields

- struct arrow_bintree_node * root_node
- int size

6.1.1 Detailed Description

Binary tree data structure.

Definition at line 88 of file arrow.h.

6.1.2 Field Documentation

6.1.2.1 struct arrow_bintree_node* arrow_bintree::root_node [read]

root node of tree

Definition at line 90 of file arrow.h.

Referenced by arrow_bintree_destruct(), arrow_bintree_init(), arrow_bintree_insert(), and arrow_bintree_-to_array().

6.1.2.2 int arrow_bintree::size

size of tree

Definition at line 91 of file arrow.h.

Referenced by arrow_bintree_destruct(), arrow_bintree_init(), arrow_bintree_insert(), arrow_bintree_to_-array(), arrow_problem_info_get(), constrained_shake_deep_apply(), and insert_at().

The documentation for this struct was generated from the following file:

- lib/arrow.h

6.2 arrow_bintree_node Struct Reference

Binary tree node.

`#include <arrow.h>`

Collaboration diagram for arrow_bintree_node:

Data Fields

- int data
- int has_left_node
- int has_right_node
- struct arrow_bintree_node * left_node
- struct arrow_bintree_node * right_node

6.2.1 Detailed Description

Binary tree node.

Definition at line 97 of file arrow.h.

6.2.2 Field Documentation

6.2.2.1 int arrow_bintree_node::data

data contained in node

Definition at line 99 of file arrow.h.

Referenced by fill_array(), and insert_at().

6.2.2.2 int arrow_bintree_node::has_left_node

true if left node exists

Definition at line 100 of file arrow.h.

Referenced by destruct_node(), fill_array(), and insert_at().

6.2.2.3 int arrow_bintree_node::has_right_node

true if right node exists

Definition at line 101 of file arrow.h.

Referenced by destruct_node(), fill_array(), and insert_at().

6.2.2.4 struct arrow_bintree_node* arrow_bintree_node::left_node [read]

left node

Definition at line 102 of file arrow.h.

Referenced by destruct_node(), fill_array(), and insert_at().

6.2.2.5 struct arrow_bintree_node* arrow_bintree_node::right_node [read]

right node

Definition at line 103 of file arrow.h.

Referenced by destruct_node(), fill_array(), and insert_at().

The documentation for this struct was generated from the following file:

- lib/arrow.h

6.3 arrow_bound_result Struct Reference

A lower bound result.

`#include <arrow.h>`

Data Fields

- int obj_value
- double total_time

6.3.1 Detailed Description

A lower bound result.

Definition at line 79 of file arrow.h.

6.3.2 Field Documentation

6.3.2.1 int arrow_bound_result::obj_value

objective value

Definition at line 81 of file arrow.h.

Referenced by arrow_2mb_solve(), arrow_bap_solve(), arrow_bbssp_solve(), arrow_bscssp_solve(), arrow_dcbpb_solve(), and main().

6.3.2.2 double arrow_bound_result::total_time

total time

Definition at line 82 of file arrow.h.

Referenced by arrow_2mb_solve(), arrow_bap_solve(), arrow_bbssp_solve(), arrow_bscssp_solve(), arrow_dcbpb_solve(), and main().

The documentation for this struct was generated from the following file:

- lib/arrow.h

6.4 arrow_btsp_fun Struct Reference

BTSP Cost matrix function definition.

`#include <arrow.h>`

Data Fields

- void * data
- int shallow
- double feasible_length
- int(* apply)(struct arrow_btsp_fun *fun, arrow_problem *old_problem, int delta, arrow_problem *new_problem)

  Applies the function to the given problem.

- void(* destruct)(struct arrow_btsp_fun *fun)

  Destructs the function structure.

- int(* feasible)(struct arrow_btsp_fun *fun, arrow_problem *problem, int delta, double tour_length, int *tour)

  Determines if the given tour is feasible or not.

6.4.1 Detailed Description

BTSP Cost matrix function definition.

Definition at line 190 of file arrow.h.

6.4.2 Field Documentation

6.4.2.1 void* arrow_btsp_fun::data

data required by function

Definition at line 192 of file arrow.h.

Referenced by arrow_btsp_fun_basic(), arrow_btsp_fun_constrained(), arrow_btsp_fun_constrained_shake(), constrained_deep_apply(), constrained_destruct(), constrained_shake_deep_apply(), constrained_shake_destruct(), constrained_shake_feasible(), and constrained_shallow_apply().

6.4.2.2 int arrow_btsp_fun::shallow

indicates use of shallow copy of data

Definition at line 193 of file arrow.h.

Referenced by arrow_btsp_fun_apply(), arrow_btsp_fun_basic(), arrow_btsp_fun_basic_atsp(), arrow_btsp_fun_constrained(), and arrow_btsp_fun_constrained_shake().

6.4.2.3 double arrow_btsp_fun::feasible_length

the length of a feasible tour (normally 0)

Definition at line 194 of file arrow.h.

Referenced by arrow_btsp_fun_basic(), arrow_btsp_fun_basic_atsp(), arrow_btsp_fun_constrained(), arrow_btsp_fun_constrained_shake(), and constrained_shake_feasible().

6.4.2.4 int(∗ arrow_btsp_fun::apply)(struct arrow_btsp_fun ∗fun, arrow_problem ∗old_problem, int delta, arrow_problem ∗new_problem)

Applies the function to the given problem.

Parameters:

fun [in] function structure

Referenced by arrow_btsp_fun_apply(), arrow_btsp_fun_basic(), arrow_btsp_fun_basic_atsp(), arrow_btsp_fun_constrained(), and arrow_btsp_fun_constrained_shake().

6.4.2.5 void(∗ arrow_btsp_fun::destruct)(struct arrow_btsp_fun ∗fun)

Destructs the function structure.

Parameters:

fun [out] function structure

Referenced by arrow_btsp_fun_basic(), arrow_btsp_fun_basic_atsp(), arrow_btsp_fun_constrained(), arrow_btsp_fun_constrained_shake(), and arrow_btsp_fun_destruct().

6.4.2.6 int(∗ arrow_btsp_fun::feasible)(struct arrow_btsp_fun ∗fun, arrow_problem ∗problem, int delta, double tour_length, int ∗tour)

Determines if the given tour is feasible or not.

Parameters:

fun [in] function structure

problem [in] the problem to check against

delta [in] the delta parameter

tour_length [in] the length of the given tour

tour [in] the tour in node-node format

Returns:

ARROW_TRUE if the tour is feasible, ARROW_FALSE if not

Referenced by arrow_btsp_fun_basic(), arrow_btsp_fun_basic_atsp(), arrow_btsp_fun_constrained(), arrow_btsp_fun_constrained_shake(), and feasible().

The documentation for this struct was generated from the following file:

- lib/arrow.h

6.5 arrow_btsp_params Struct Reference

BTSP algorithm parameters.

`#include <arrow.h>`

Collaboration diagram for arrow_btsp_params:

Data Fields

- int confirm_sol
- int supress_ebst
- int find_short_tour
- int lower_bound
- int upper_bound
- int num_steps
- arrow_btsp_solve_plan * steps

6.5.1 Detailed Description

BTSP algorithm parameters.

Definition at line 241 of file arrow.h.

6.5.2 Field Documentation

6.5.2.1 int arrow_btsp_params::confirm_sol

confirm sol. with exact solver?

Definition at line 243 of file arrow.h.

Referenced by arrow_btsp_params_init(), arrow_btsp_solve(), and main().

6.5.2.2 int arrow_btsp_params::supress_ebst

supress EBST-heuristic?

Definition at line 244 of file arrow.h.

Referenced by arrow_btsp_params_init(), arrow_btsp_solve(), and main().

6.5.2.3 int arrow_btsp_params::find_short_tour

find short BSTP tour?

Definition at line 245 of file arrow.h.

Referenced by arrow_btsp_params_init(), arrow_btsp_solve(), and main().

6.5.2.4 int arrow_btsp_params::lower_bound

initial lower bound

Definition at line 246 of file arrow.h.

Referenced by arrow_btsp_params_init(), arrow_btsp_solve(), and main().

6.5.2.5 int arrow_btsp_params::upper_bound

initial upper bound

Definition at line 247 of file arrow.h.

Referenced by arrow_btsp_params_init(), arrow_btsp_solve(), and main().

6.5.2.6 int arrow_btsp_params::num_steps

the number of solve plan steps

Definition at line 248 of file arrow.h.

Referenced by arrow_btsp_params_destruct(), arrow_btsp_params_init(), arrow_btsp_solve(), and main().

6.5.2.7 arrow_btsp_solve_plan* arrow_btsp_params::steps

solve plan steps

Definition at line 249 of file arrow.h.

Referenced by arrow_btsp_params_destruct(), arrow_btsp_solve(), and main().

The documentation for this struct was generated from the following file:

- lib/arrow.h

6.6 arrow_btsp_result Struct Reference

BTSP result.

`#include <arrow.h>`

Data Fields

- int found_tour
- int obj_value
- double tour_length
- int * tour
- int optimal
- int bin_search_steps
- int linkern_attempts
- double linkern_time
- int exact_attempts
- double exact_time
- double total_time

6.6.1 Detailed Description

BTSP result.

Definition at line 172 of file arrow.h.

6.6.2 Field Documentation

6.6.2.1 int arrow_btsp_result::found_tour

true if a tour was found, false otherwise

Definition at line 174 of file arrow.h.

Referenced by arrow_btsp_result_init(), arrow_btsp_solve(), feasible(), and main().

6.6.2.2 int arrow_btsp_result::obj_value

objective value (largest cost in tour)

Definition at line 175 of file arrow.h.

Referenced by arrow_btsp_result_init(), arrow_btsp_solve(), feasible(), and main().

6.6.2.3 double arrow_btsp_result::tour_length

length of the tour found

Definition at line 176 of file arrow.h.

Referenced by arrow_btsp_result_init(), arrow_btsp_solve(), feasible(), and main().

6.6.2.4 int* arrow_btsp_result::tour

tour that was found in node-node format

Definition at line 177 of file arrow.h.

Referenced by arrow_btsp_result_destruct(), arrow_btsp_result_init(), arrow_btsp_solve(), feasible(), and main().

6.6.2.5 int arrow_btsp_result::optimal

indicates if the solution is optimal

Definition at line 178 of file arrow.h.

Referenced by arrow_btsp_solve(), and main().

6.6.2.6 int arrow_btsp_result::bin_search_steps

number of steps in binary search

Definition at line 179 of file arrow.h.

Referenced by arrow_btsp_result_init(), arrow_btsp_solve(), and main().

6.6.2.7 int arrow_btsp_result::linkern_attempts

number of calls to LK heuristic

Definition at line 180 of file arrow.h.

Referenced by arrow_btsp_result_init(), arrow_btsp_solve(), feasible(), and main().

6.6.2.8 double arrow_btsp_result::linkern_time

total time calling LK heuristic

Definition at line 181 of file arrow.h.

Referenced by arrow_btsp_result_init(), arrow_btsp_solve(), feasible(), and main().

6.6.2.9 int arrow_btsp_result::exact_attempts

number of calls to exact TSP solver

Definition at line 182 of file arrow.h.

Referenced by arrow_btsp_result_init(), arrow_btsp_solve(), feasible(), and main().

6.6.2.10 double arrow_btsp_result::exact_time

total time calling exact TSP solver

Definition at line 183 of file arrow.h.

Referenced by arrow_btsp_result_init(), arrow_btsp_solve(), feasible(), and main().

6.6.2.11 double arrow_btsp_result::total_time

total time

Definition at line 184 of file arrow.h.

Referenced by arrow_btsp_result_init(), arrow_btsp_solve(), feasible(), and main().

The documentation for this struct was generated from the following file:

- lib/arrow.h

6.7 arrow_btsp_solve_plan Struct Reference

BTSP feasibility solve step plan.

`#include <arrow.h>`

Collaboration diagram for arrow_btsp_solve_plan:

Data Fields

- int plan_type
- int use_exact_solver
- arrow_btsp_fun fun
- arrow_tsp_lk_params lk_params
- int upper_bound_update
- int attempts

6.7.1 Detailed Description

BTSP feasibility solve step plan.

Definition at line 228 of file arrow.h.

6.7.2 Field Documentation

6.7.2.1 int arrow_btsp_solve_plan::plan_type

the type of plan (see macros)

Definition at line 230 of file arrow.h.

Referenced by arrow_btsp_solve_plan_init(), and feasible().

6.7.2.2 int arrow_btsp_solve_plan::use_exact_solver

use exact TSP solver?

Definition at line 231 of file arrow.h.

Referenced by feasible().

6.7.2.3 arrow_btsp_fun arrow_btsp_solve_plan::fun

the cost matrix function to apply

Definition at line 232 of file arrow.h.

Referenced by arrow_btsp_solve_plan_destruct(), and feasible().

6.7.2.4 arrow_tsp_lk_params arrow_btsp_solve_plan::lk_params

LK params to use

Definition at line 233 of file arrow.h.

Referenced by arrow_btsp_solve_plan_destruct(), and feasible().

6.7.2.5 int arrow_btsp_solve_plan::upper_bound_update

Check for better upper bound?

Definition at line 234 of file arrow.h.

Referenced by feasible().

6.7.2.6 int arrow_btsp_solve_plan::attempts

number of attempts to perform

Definition at line 235 of file arrow.h.

Referenced by arrow_btsp_solve_plan_init(), and feasible().

The documentation for this struct was generated from the following file:

- lib/arrow.h

6.8 arrow_option Struct Reference

Program options structure.

`#include <arrow.h>`

Data Fields

- char short_option
- const char * long_option
- const char * help_message
- int data_type
- void * data_ptr
- int opt_required
- int arg_required

6.8.1 Detailed Description

Program options structure.

Definition at line 255 of file arrow.h.

6.8.2 Field Documentation

6.8.2.1 char arrow_option::short_option

short option (flag)

Definition at line 257 of file arrow.h.

Referenced by arrow_options_parse().

6.8.2.2 const char* arrow_option::long_option

long option

Definition at line 258 of file arrow.h.

Referenced by arrow_options_parse().

6.8.2.3 const char* arrow_option::help_message

help message to display for option

Definition at line 259 of file arrow.h.

6.8.2.4 int arrow_option::data_type

one of ARROW_OPTION_INT, ARROW_OPTION_DOUBLE, ARROW_OPTION_STRING

Definition at line 260 of file arrow.h.

6.8.2.5 void* arrow_option::data_ptr

pointer to variable to hold parameter

Definition at line 263 of file arrow.h.

6.8.2.6 int arrow_option::opt_required

if true ensures option is present

Definition at line 264 of file arrow.h.

6.8.2.7 int arrow_option::arg_required

if true, ensures argument for parameter passed, otherwise puts 1 into data_ptr if parameter is present

Definition at line 265 of file arrow.h.

The documentation for this struct was generated from the following file:

- lib/arrow.h

6.9 arrow_problem Struct Reference

Problem data structure.

`#include <arrow.h>`

Data Fields

- int size
- int symmetric
- CCdatagroup data
- int shallow
- char name [64]
- int(* get_cost)(struct arrow_problem *this, int i, int j)

  Returns the cost between node i and node j.

6.9.1 Detailed Description

Problem data structure.

Definition at line 109 of file arrow.h.

6.9.2 Field Documentation

6.9.2.1 int arrow_problem::size

problem size

Definition at line 111 of file arrow.h.

Referenced by arrow_2mb_solve(), arrow_bap_solve(), arrow_bbssp_biconnected(), arrow_btsp_fun_-apply(), arrow_btsp_result_init(), arrow_btsp_solve(), arrow_dcbpb_solve(), arrow_problem_abtsp_to_-sbtsp(), arrow_problem_info_get(), arrow_problem_print(), arrow_problem_read(), arrow_tsp_exact_-solve(), arrow_tsp_lk_params_init(), arrow_tsp_lk_solve(), arrow_tsp_result_init(), arrow_util_sbtsp_-to_abstp_tour(), arrow_util_write_tour(), basic_atsp_deep_apply(), basic_atsp_feasible(), basic_deep_-apply(), bottleneck_paths(), build_initial_tour(), constrained_deep_apply(), constrained_shake_deep_-apply(), constrained_shake_feasible(), feasible(), find_art_points(), initialize_flow_data(), main(), read_-atsp(), strongly_connected(), and strongly_connected_dfs().

6.9.2.2 int arrow_problem::symmetric

indicates if cost matrix is symmetric

Definition at line 112 of file arrow.h.

Referenced by arrow_2mb_solve(), arrow_bbssp_solve(), arrow_btsp_solve(), arrow_problem_abtsp_to_-sbtsp(), arrow_problem_info_get(), arrow_problem_read(), and main().

6.9.2.3 CCdatagroup arrow_problem::data

Concorde data structure for problem.

Definition at line 113 of file arrow.h.

Referenced by arrow_btsp_fun_apply(), arrow_problem_abtsp_to_sbtsp(), arrow_problem_destruct(), arrow_problem_get_cost(), arrow_problem_read(), arrow_tsp_exact_solve(), arrow_tsp_lk_solve(), basic_atsp_deep_apply(), basic_deep_apply(), basic_shallow_apply(), build_initial_tour(), constrained_deep_apply(), constrained_shake_deep_apply(), constrained_shallow_apply(), and read_atsp().

6.9.2.4 int arrow_problem::shallow

indicates use of shallow copy of data

Definition at line 114 of file arrow.h.

Referenced by arrow_btsp_fun_apply(), arrow_problem_abtsp_to_sbtsp(), arrow_problem_destruct(), and arrow_problem_read().

6.9.2.5 char arrow_problem::name[64]

name of problem (can be NULL)

Definition at line 115 of file arrow.h.

Referenced by arrow_btsp_fun_apply(), arrow_problem_abtsp_to_sbtsp(), arrow_problem_read(), arrow_tsp_exact_solve(), arrow_util_write_tour(), and main().

6.9.2.6 int(∗ arrow_problem::get_cost)(struct arrow_problem ∗this, int i, int j)

Returns the cost between node i and node j.

Parameters:

this [in] problem data

i [in] node i

j [in] node j

Returns:

cost between node i and j.

Referenced by arrow_2mb_solve(), arrow_btsp_fun_apply(), arrow_dcbpb_solve(), arrow_problem_abtsp_to_sbtsp(), arrow_problem_info_get(), arrow_problem_print(), arrow_problem_read(), arrow_util_sbtsp_to_abstp_tour(), basic_atsp_deep_apply(), basic_atsp_feasible(), basic_deep_apply(), bottleneck_paths(), constrained_deep_apply(), constrained_shake_deep_apply(), constrained_shake_feasible(), feasible(), find_art_points(), initialize_flow_data(), main(), and strongly_connected_dfs().

The documentation for this struct was generated from the following file:

- lib/arrow.h

6.10 arrow_problem_info Struct Reference

Problem information data structure.

`#include <arrow.h>`

Data Fields

- int * cost_list
- int cost_list_length
- int min_cost
- int max_cost

6.10.1 Detailed Description

Problem information data structure.

Definition at line 131 of file arrow.h.

6.10.2 Field Documentation

6.10.2.1 int* arrow_problem_info::cost_list

sorted list of unique costs from problem.

Definition at line 133 of file arrow.h.

Referenced by arrow_bap_solve(), arrow_bbssp_solve(), arrow_bscssp_solve(), arrow_btsp_solve(), arrow_problem_info_destruct(), arrow_problem_info_get(), constrained_shake_deep_apply(), and main().

6.10.2.2 int arrow_problem_info::cost_list_length

length of cost list.

Definition at line 134 of file arrow.h.

Referenced by arrow_bap_solve(), arrow_bbssp_solve(), arrow_bscssp_solve(), arrow_btsp_solve(), arrow_problem_info_get(), constrained_shake_deep_apply(), and main().

6.10.2.3 int arrow_problem_info::min_cost

smallest cost in problem.

Definition at line 135 of file arrow.h.

Referenced by arrow_problem_info_get(), and main().

6.10.2.4 int arrow_problem_info::max_cost

largest cost in problem.

Definition at line 136 of file arrow.h.

Referenced by arrow_problem_info_get(), and main().

The documentation for this struct was generated from the following file:

- lib/arrow.h

6.11 arrow_tsp_lk_params Struct Reference

LK algorithm parameters.

`#include <arrow.h>`

Data Fields

- int random_restarts
- int stall_count
- int kicks
- int kick_type
- double time_bound
- double length_bound
- int * initial_tour

6.11.1 Detailed Description

LK algorithm parameters.

Definition at line 142 of file arrow.h.

6.11.2 Field Documentation

6.11.2.1 int arrow_tsp_lk_params::random_restarts

the number of random restarts to perform

Definition at line 144 of file arrow.h.

Referenced by arrow_tsp_lk_params_init(), arrow_tsp_lk_solve(), and main().

6.11.2.2 int arrow_tsp_lk_params::stall_count

the max number of 4-swap kicks to perform without making progress

Definition at line 145 of file arrow.h.

Referenced by arrow_tsp_lk_params_init(), arrow_tsp_lk_solve(), and main().

6.11.2.3 int arrow_tsp_lk_params::kicks

the number of 4-swap kicks to perform

Definition at line 147 of file arrow.h.

Referenced by arrow_tsp_lk_params_init(), arrow_tsp_lk_solve(), and main().

6.11.2.4 int arrow_tsp_lk_params::kick_type

the type of kick: one of CC_LK_RANDOM_KICK, CC_LK_GEOMETRIC_KICK, or CC_LK_CLOSE_-KICK; see Concorde documentation for more info

Definition at line 148 of file arrow.h.

Referenced by arrow_tsp_lk_params_init(), and arrow_tsp_lk_solve().

6.11.2.5 double arrow_tsp_lk_params::time_bound

stop after this running time reached; set to 0 to have no time bound, must be positive

Definition at line 151 of file arrow.h.

Referenced by arrow_tsp_lk_params_init(), and arrow_tsp_lk_solve().

6.11.2.6 double arrow_tsp_lk_params::length_bound

stop after finding tour of this length; must be non-negative

Definition at line 153 of file arrow.h.

Referenced by arrow_tsp_lk_params_init(), arrow_tsp_lk_solve(), and main().

6.11.2.7 int* arrow_tsp_lk_params::initial_tour

initial tour (may be NULL)

Definition at line 155 of file arrow.h.

Referenced by arrow_tsp_lk_params_destruct(), arrow_tsp_lk_params_init(), and arrow_tsp_lk_solve().

The documentation for this struct was generated from the following file:

- lib/arrow.h

6.12 arrow_tsp_result Struct Reference

TSP result (including result from LK heuristic).

`#include <arrow.h>`

Data Fields

- int found_tour
- double obj_value
- int * tour
- double total_time

6.12.1 Detailed Description

TSP result (including result from LK heuristic).

Definition at line 161 of file arrow.h.

6.12.2 Field Documentation

6.12.2.1 int arrow_tsp_result::found_tour

true if a tour was found, false otherwise

Definition at line 163 of file arrow.h.

Referenced by arrow_tsp_exact_solve(), arrow_tsp_lk_solve(), arrow_tsp_result_init(), and main().

6.12.2.2 double arrow_tsp_result::obj_value

objective value (tour length)

Definition at line 164 of file arrow.h.

Referenced by arrow_tsp_exact_solve(), arrow_tsp_lk_solve(), arrow_tsp_result_init(), feasible(), and main().

6.12.2.3 int* arrow_tsp_result::tour

tour that was found in node-node format

Definition at line 165 of file arrow.h.

Referenced by arrow_tsp_exact_solve(), arrow_tsp_lk_solve(), arrow_tsp_result_destruct(), arrow_tsp_-result_init(), and feasible().

6.12.2.4 double arrow_tsp_result::total_time

total time

Definition at line 166 of file arrow.h.

Referenced by arrow_tsp_exact_solve(), arrow_tsp_lk_solve(), arrow_tsp_result_init(), feasible(), and main().

The documentation for this struct was generated from the following file:

- lib/arrow.h

6.13 basic_data Struct Reference

Concorde userdat structure for basic cost matrix function.

Data Fields

- CCdatagroup * dat
- int delta

6.13.1 Detailed Description

Concorde userdat structure for basic cost matrix function.

Definition at line 178 of file btsp_fun.c.

6.13.2 Field Documentation

6.13.2.1 CCdatagroup* basic_data::dat

existing Concorde data structure

Definition at line 180 of file btsp_fun.c.

Referenced by basic_edgelen(), and basic_shallow_apply().

6.13.2.2 int basic_data::delta

delta value

Definition at line 181 of file btsp_fun.c.

Referenced by basic_shallow_apply().

The documentation for this struct was generated from the following file:

- lib/btsp_fun.c

6.14 constrained_data Struct Reference

Concorde userdat structure for constrained cost matrix function.

Data Fields

- CCdatagroup * dat
- int infinity
- int delta

6.14.1 Detailed Description

Concorde userdat structure for constrained cost matrix function.

Definition at line 187 of file btsp_fun.c.

6.14.2 Field Documentation

6.14.2.1 CCdatagroup* constrained_data::dat

existing Concorde data structure

Definition at line 189 of file btsp_fun.c.

Referenced by constrained_edgelen(), and constrained_shallow_apply().

6.14.2.2 int constrained_data::infinity

value to use for "infinity"

Definition at line 190 of file btsp_fun.c.

Referenced by constrained_edgelen(), and constrained_shallow_apply().

6.14.2.3 int constrained_data::delta

delta value

Definition at line 191 of file btsp_fun.c.

Referenced by constrained_shallow_apply().

The documentation for this struct was generated from the following file:

- lib/btsp_fun.c

6.15 constrained_shake_data Struct Reference

Concorde userdat structure for constrained shake cost matrix function.

Collaboration diagram for constrained_shake_data:

Data Fields

- arrow_problem * problem
- arrow_problem_info * info
- int rand_min
- int rand_max
- int random_list_length
- int infinity

6.15.1 Detailed Description

Concorde userdat structure for constrained shake cost matrix function.

Definition at line 198 of file btsp_fun.c.

6.15.2 Field Documentation

6.15.2.1 arrow_problem* constrained_shake_data::problem

original problem data

Definition at line 200 of file btsp_fun.c.

Referenced by arrow_btsp_fun_constrained_shake().

6.15.2.2 arrow_problem_info* constrained_shake_data::info

original problem info

Definition at line 201 of file btsp_fun.c.

Referenced by arrow_btsp_fun_constrained_shake(), and constrained_shake_deep_apply().

6.15.2.3 int constrained_shake_data::rand_min

smallest random number to generate

Definition at line 202 of file btsp_fun.c.

Referenced by arrow_btsp_fun_constrained_shake(), and constrained_shake_deep_apply().

6.15.2.4 int constrained_shake_data::rand_max

largest random number to generate

Definition at line 203 of file btsp_fun.c.

Referenced by arrow_btsp_fun_constrained_shake(), and constrained_shake_deep_apply().

6.15.2.5 int constrained_shake_data::random_list_length

the number of random numbers in list

Definition at line 204 of file btsp_fun.c.

6.15.2.6 int constrained_shake_data::infinity

value to use for "infinity"

Definition at line 205 of file btsp_fun.c.

Referenced by arrow_btsp_fun_constrained_shake(), and constrained_shake_deep_apply().

The documentation for this struct was generated from the following file:

- lib/btsp_fun.c

Chapter 7

File Documentation

7.1 /Users/johnlarusic/Dev/arrow/global.dox File Reference

7.2 bin/2mb.c File Reference

2-Max Bound solver.

```
#include "arrow.h"
```

Defines

- #define NUM_OPTS 2

Functions

- int main (int argc, char *argv[ ])

Variables

- char * program_name
- char * input_file = NULL
- char * xml_file = NULL
- arrow_option options [NUM_OPTS]
- char * desc = "2-Max Bound solver"
- char * usage = "-i tsplib.tsp [options] "

7.2.1 Detailed Description

2-Max Bound solver.

Solves the 2-max bound (2MB) on the given input file.

Author:

John LaRusic

Definition in file 2mb.c.

7.2.2 Define Documentation

7.2.2.1 #define NUM_OPTS 2

Definition at line 17 of file 2mb.c.

Referenced by main().

7.2.3 Function Documentation

7.2.3.1 int main (int *argc*, char * *argv*[ ])

Definition at line 30 of file 2mb.c.

References arrow_2mb_solve(), ARROW_FAILURE, arrow_options_parse(), arrow_print_error, arrow_problem_destruct(), arrow_problem_read(), ARROW_SUCCESS, arrow_util_print_program_args(), desc,

input_file, NUM_OPTS, arrow_bound_result::obj_value, arrow_bound_result::total_time, usage, and xml_file.

7.2.4 Variable Documentation

7.2.4.1 char* desc = "2-Max Bound solver"

Definition at line 25 of file 2mb.c.

Referenced by main().

7.2.4.2 char* input_file = NULL

Given input TSPLIB file

Definition at line 13 of file 2mb.c.

Referenced by main(), and read_args().

7.2.4.3 arrow_option options[NUM_OPTS]

Initial value:

```
{
    {'i', "input", "TSPLIB input file",
        ARROW_OPTION_STRING, &input_file, ARROW_TRUE, ARROW_TRUE},
    {'x', "xml", "File to write XML output to",
        ARROW_OPTION_STRING, &xml_file, ARROW_FALSE, ARROW_TRUE}
}
```

Definition at line 18 of file 2mb.c.

7.2.4.4 char* program_name

Program name

Definition at line 12 of file 2mb.c.

Referenced by main(), print_help(), print_usage(), and print_version().

7.2.4.5 char* usage = "-i tsplib.tsp [options] "

Definition at line 26 of file 2mb.c.

Referenced by main().

7.2.4.6 char* xml_file = NULL

Definition at line 14 of file 2mb.c.

Referenced by main().

7.3 lib/2mb.c File Reference

2-max bound implemenation.

```
#include "arrow.h"
```

Functions

- int arrow_2mb_solve (arrow_problem *problem, arrow_bound_result *result)
  Solves the 2-max bound (2MB) on the given problem.

7.3.1 Detailed Description

2-max bound implemenation.

Implemenation of the 2-max bound (2MB) used as a lower bound for the Bottleneck TSP objective value.

Author:

John LaRusic

Definition in file 2mb.c.

7.3.2 Function Documentation

7.3.2.1 int arrow_2mb_solve (arrow_problem * *problem*, arrow_bound_result * *result*)

Solves the 2-max bound (2MB) on the given problem.

Parameters:

problem [in] problem data

result [out] 2MB solution

Definition at line 16 of file 2mb.c.

References ARROW_SUCCESS, arrow_util_zeit(), arrow_problem::get_cost, max(), arrow_bound_result::obj_value, arrow_problem::size, arrow_problem::symmetric, and arrow_bound_result::total_time.

Referenced by main().

7.4 bin/abtsp.c File Reference

Asymmetric Bottleneck TSP heuristic.

```
#include "arrow.h"
```

Defines

- #define NUM_OPTS 12
- #define SOLVE_STEPS 1

Functions

- int main (int argc, char *argv[ ])

Variables

- char * input_file = NULL
- char * xml_file = NULL
- char * tour_file = NULL
- int random_restarts = -1
- int stall_count = -1
- int kicks = -1
- int confirm_sol = ARROW_FALSE
- int supress_ebst = ARROW_FALSE
- int find_short_tour = ARROW_FALSE
- int lower_bound = -1
- int upper_bound = INT_MAX
- int basic_attempts = ARROW_DEFAULT_BASIC_ATTEMPTS
- arrow_option options [NUM_OPTS]
- char * desc = "Bottleneck traveling salesman problem (BTSP) solver"
- char * usage = "-i tsplib.tsp [options]"

7.4.1 Detailed Description

Asymmetric Bottleneck TSP heuristic.

Runs the Bottleneck TSP heuristic on the given asymmetric input file.

Author:

John LaRusic

Definition in file abtsp.c.

7.4.2 Define Documentation

7.4.2.1 #define NUM_OPTS 12

Definition at line 26 of file abtsp.c.

7.4.2.2 #define SOLVE_STEPS 1

Referenced by main().

7.4.3 Function Documentation

7.4.3.1 int main (int *argc*, char * *argv*[ ])

Definition at line 61 of file abtsp.c.

References arrow_btsp_fun_basic_atsp(), arrow_btsp_fun_destruct(), arrow_btsp_params_destruct(), arrow_btsp_params_init(), arrow_btsp_result_destruct(), arrow_btsp_result_init(), arrow_btsp_solve(), ARROW_BTSP_SOLVE_PLAN_BASIC, arrow_debug, ARROW_FAILURE, ARROW_FALSE, arrow_options_parse(), arrow_print_error, arrow_problem_abtsp_to_sbtsp(), arrow_problem_destruct(), arrow_problem_info_get(), arrow_problem_read(), ARROW_SUCCESS, ARROW_TRUE, arrow_tsp_lk_params_destruct(), arrow_tsp_lk_params_init(), arrow_util_create_int_array(), arrow_util_print_program_args(), arrow_util_sbtsp_to_abstp_tour(), arrow_util_write_tour(), arrow_util_zeit(), basic_attempts, arrow_btsp_result::bin_search_steps, confirm_sol, arrow_btsp_params::confirm_sol, arrow_problem_info::cost_list_length, desc, arrow_btsp_result::exact_attempts, arrow_btsp_result::exact_time, find_short_tour, arrow_btsp_params::find_short_tour, arrow_btsp_result::found_tour, arrow_problem::get_cost, input_file, arrow_tsp_lk_params::kicks, kicks, arrow_tsp_lk_params::length_bound, arrow_btsp_result::linkern_attempts, arrow_btsp_result::linkern_time, arrow_btsp_params::lower_bound, lower_bound, arrow_problem_info::max_cost, arrow_problem_info::min_cost, NUM_OPTS, arrow_btsp_params::num_steps, arrow_btsp_result::obj_value, arrow_btsp_result::optimal, arrow_tsp_lk_params::random_restarts, random_restarts, arrow_problem::size, SOLVE_STEPS, arrow_tsp_lk_params::stall_count, stall_count, arrow_btsp_params::steps, supress_ebst, arrow_btsp_params::supress_ebst, arrow_problem::symmetric, arrow_btsp_result::total_time, arrow_btsp_result::tour, tour_file, arrow_btsp_result::tour_length, arrow_btsp_params::upper_bound, upper_bound, usage, and xml_file.

7.4.4 Variable Documentation

7.4.4.1 int basic_attempts = ARROW_DEFAULT_BASIC_ATTEMPTS

Definition at line 23 of file abtsp.c.

Referenced by main().

7.4.4.2 int confirm_sol = ARROW_FALSE

Definition at line 18 of file abtsp.c.

Referenced by main().

7.4.4.3 char* desc = "Bottleneck traveling salesman problem (BTSP) solver"

Definition at line 56 of file abtsp.c.

7.4.4.4 int find_short_tour = ARROW_FALSE

Definition at line 20 of file abtsp.c.

Referenced by main().

7.4.4.5 char* input_file = NULL

Definition at line 12 of file abtsp.c.

7.4.4.6 int kicks = -1

Definition at line 17 of file abtsp.c.

Referenced by main(), and read_args().

7.4.4.7 int lower_bound = -1

Definition at line 21 of file abtsp.c.

Referenced by main().

7.4.4.8 arrow_option options[NUM_OPTS]

Initial value:

```
{
    {'i', "input", "TSPLIB input file",
        ARROW_OPTION_STRING, &input_file, ARROW_TRUE, ARROW_TRUE},
    {'x', "xml", "file to write XML output to",
        ARROW_OPTION_STRING, &xml_file, ARROW_FALSE, ARROW_TRUE},
    {'T', "tour", "file to write tour to",
        ARROW_OPTION_STRING, &tour_file, ARROW_FALSE, ARROW_TRUE},

    {'r', "restarts", "number of random restarts",
        ARROW_OPTION_INT, &random_restarts, ARROW_FALSE, ARROW_TRUE},
    {'s', "stall-count", "max number of 4-swaps w/o progress",
        ARROW_OPTION_INT, &stall_count, ARROW_FALSE, ARROW_TRUE},
    {'k', "kicks", "number of 4-swap kicks",
        ARROW_OPTION_INT, &kicks, ARROW_FALSE, ARROW_TRUE},
    {'c', "confirm-solution", "confirm solution with exact solver",
        ARROW_OPTION_INT, &confirm_sol, ARROW_FALSE, ARROW_FALSE},
    {'e', "supress-ebst", "supress binary search",
        ARROW_OPTION_INT, &supress_ebst, ARROW_FALSE, ARROW_FALSE},
    {'S', "find-short-tour", "finds a (relatively) short BTSP tour",
        ARROW_OPTION_INT, &find_short_tour, ARROW_FALSE, ARROW_FALSE},

    {'l', "lower-bound", "initial lower bound",
        ARROW_OPTION_INT, &lower_bound, ARROW_FALSE, ARROW_TRUE},
    {'u', "upper-bound", "initial upper bound",
        ARROW_OPTION_INT, &upper_bound, ARROW_FALSE, ARROW_TRUE},
    {'a', "basic-attempts", "number of basic attempts",
        ARROW_OPTION_INT, &basic_attempts, ARROW_FALSE, ARROW_TRUE}
}
```

Definition at line 27 of file abtsp.c.

7.4.4.9 int random_restarts = -1

Definition at line 15 of file abtsp.c.

Referenced by main(), and read_args().

7.4.4.10 int stall_count = -1

Definition at line 16 of file abtsp.c.

Referenced by main(), and read_args().

7.4.4.11 int supress_ebst = ARROW_FALSE

Definition at line 19 of file abtsp.c.

Referenced by main().

7.4.4.12 char* tour_file = NULL

Definition at line 14 of file abtsp.c.

Referenced by main(), and print_xml_output().

7.4.4.13 int upper_bound = INT_MAX

Definition at line 22 of file abtsp.c.

Referenced by arrow_btsp_solve(), and main().

7.4.4.14 char* usage = "-i tsplib.tsp [options]"

Definition at line 57 of file abtsp.c.

7.4.4.15 char* xml_file = NULL

Definition at line 13 of file abtsp.c.

7.5 bin/bap.c File Reference

Bottleneck Assignment Problem solver.

```
#include "arrow.h"
```

Defines

- #define NUM_OPTS 2

Functions

- int main (int argc, char ∗argv[ ])

Variables

- char ∗ program_name
- char ∗ input_file = NULL
- char ∗ xml_file = NULL
- arrow_option options [NUM_OPTS]
- char ∗ desc = "Bottleneck assignment problem solver"
- char ∗ usage = "-i tsplib.tsp [options] "

7.5.1 Detailed Description

Bottleneck Assignment Problem solver.

Solves the bottleneck assignment problem (BAP) on the given input file.

Author:

John LaRusic

Definition in file bap.c.

7.5.2 Define Documentation

7.5.2.1 #define NUM_OPTS 2

Definition at line 17 of file bap.c.

7.5.3 Function Documentation

7.5.3.1 int main (int *argc*, char ∗ *argv*[ ])

Definition at line 30 of file bap.c.

References arrow_bap_solve(), ARROW_FAILURE, arrow_options_parse(), arrow_print_error, arrow_problem_destruct(), arrow_problem_info_destruct(), arrow_problem_info_get(), arrow_problem_read(), ARROW_SUCCESS, arrow_util_print_program_args(), desc, input_file, NUM_OPTS, arrow_bound_result::obj_value, arrow_bound_result::total_time, usage, and xml_file.

7.5.4 Variable Documentation

7.5.4.1 char* desc = "Bottleneck assignment problem solver"

Definition at line 25 of file bap.c.

7.5.4.2 char* input_file = NULL

Given input TSPLIB file

Definition at line 13 of file bap.c.

7.5.4.3 arrow_option options[NUM_OPTS]

Initial value:

```
{
    {'i', "input", "TSPLIB input file",
        ARROW_OPTION_STRING, &input_file, ARROW_TRUE, ARROW_TRUE},
    {'x', "xml", "File to write XML output to",
        ARROW_OPTION_STRING, &xml_file, ARROW_FALSE, ARROW_TRUE}
}
```

Definition at line 18 of file bap.c.

7.5.4.4 char* program_name

Program name

Definition at line 12 of file bap.c.

7.5.4.5 char* usage = "-i tsplib.tsp [options] "

Definition at line 26 of file bap.c.

7.5.4.6 char* xml_file = NULL

Definition at line 14 of file bap.c.

7.6 lib/bap.c File Reference

Bottleneck assignment problem (BAP) implemenation.

```
#include "arrow.h"
```

Functions

- void initialize_flow_data (arrow_problem *problem, int delta, int s, int t, int **res, int *m, int *dist, int *pred)
- void shortest_augmenting_path (int n, int s, int t, int stop, int **res, int *dist, int *pred, int *flow)
- void ford_fulkerson_labeling (int n, int s, int t, int **res, int *label, int *pred, int *flow, int *list)
- int arrow_bap_solve (arrow_problem *problem, arrow_problem_info *info, arrow_bound_result *result)

7.6.1 Detailed Description

Bottleneck assignment problem (BAP) implemenation.

Implemenation of the bottleneck assignment problem (BAP) that's used as a lower bound for the Bottleneck TSP objective value.

Author:

John LaRusic

Definition in file bap.c.

7.6.2 Function Documentation

7.6.2.1 int arrow_bap_solve (arrow_problem * *problem*, arrow_problem_info * *info*, arrow_bound_result * *result*)

Definition at line 67 of file bap.c.

References ARROW_FAILURE, ARROW_SUCCESS, arrow_util_create_int_array(), arrow_util_create_int_matrix(), arrow_util_zeit(), arrow_problem_info::cost_list, arrow_problem_info::cost_list_length, ford_fulkerson_labeling(), initialize_flow_data(), arrow_bound_result::obj_value, shortest_augmenting_path(), arrow_problem::size, and arrow_bound_result::total_time.

Referenced by main().

7.6.2.2 void ford_fulkerson_labeling (int *n*, int *s*, int *t*, int ** *res*, int * *label*, int * *pred*, int * *flow*, int * *list*)

Definition at line 251 of file bap.c.

References ARROW_FALSE, and ARROW_TRUE.

Referenced by arrow_bap_solve().

7.6.2.3 void initialize_flow_data (arrow_problem * *problem*, int *delta*, int *s*, int *t*, int ** *res*, int * *m*, int * *dist*, int * *pred*)

Definition at line 152 of file bap.c.

References arrow_problem::get_cost, and arrow_problem::size.

Referenced by arrow_bap_solve().

7.6.2.4 void shortest_augmenting_path (int *n*, int *s*, int *t*, int *stop*, int ** *res*, int * *dist*, int * *pred*, int * *flow*)

Definition at line 193 of file bap.c.

References ARROW_FALSE, ARROW_SUCCESS, and ARROW_TRUE.

Referenced by arrow_bap_solve().

7.7 bin/bbssp.c File Reference

Bottleneck Biconnected Spanning Subgraph solver.

```
#include "arrow.h"
```

Defines

- #define NUM_OPTS 2

Functions

- int main (int argc, char *argv[ ])

Variables

- char * program_name
- char * input_file = NULL
- char * xml_file = NULL
- arrow_option options [NUM_OPTS]
- char * desc = "Bottleneck biconnected spanning subgraph solver"
- char * usage = "-i tsplib.tsp [options] "

7.7.1 Detailed Description

Bottleneck Biconnected Spanning Subgraph solver.

Solves the bottleneck biconnected spanning subgraph problem (BBSSP) problem on the given input file.

Author:

John LaRusic

Definition in file bbssp.c.

7.7.2 Define Documentation

7.7.2.1 #define NUM_OPTS 2

Definition at line 18 of file bbssp.c.

7.7.3 Function Documentation

7.7.3.1 int main (int *argc*, char * *argv*[ ])

Definition at line 31 of file bbssp.c.

References arrow_bbssp_solve(), ARROW_FAILURE, arrow_options_parse(), arrow_print_error, arrow_problem_abtsp_to_sbtsp(), arrow_problem_destruct(), arrow_problem_info_destruct(), arrow_problem_info_get(), arrow_problem_read(), ARROW_SUCCESS, arrow_util_print_program_args(), desc, input_file, arrow_problem_info::max_cost, NUM_OPTS, arrow_bound_result::obj_value, arrow_problem::size, arrow_problem::symmetric, arrow_bound_result::total_time, usage, and xml_file.

7.7.4 Variable Documentation

7.7.4.1 char* desc = "Bottleneck biconnected spanning subgraph solver"

Definition at line 26 of file bbssp.c.

7.7.4.2 char* input_file = NULL

Given input TSPLIB file

Definition at line 14 of file bbssp.c.

7.7.4.3 arrow_option options[NUM_OPTS]

Initial value:

```
{
    {'i', "input", "TSPLIB input file",
        ARROW_OPTION_STRING, &input_file, ARROW_TRUE, ARROW_TRUE},
    {'x', "xml", "File to write XML output to",
        ARROW_OPTION_STRING, &xml_file, ARROW_FALSE, ARROW_TRUE}
}
```

Definition at line 19 of file bbssp.c.

7.7.4.4 char* program_name

Program name

Definition at line 13 of file bbssp.c.

7.7.4.5 char* usage = "-i tsplib.tsp [options] "

Definition at line 27 of file bbssp.c.

7.7.4.6 char* xml_file = NULL

Definition at line 15 of file bbssp.c.

7.8 lib/bbssp.c File Reference

Bottleneck biconnected spanning subgraph problem implemenation.

```
#include "arrow.h"
```

Functions

- int find_art_points (arrow_problem *problem, int max_cost, int node, int depth_num, int root_children, int *visited, int *depth, int *low, int *parent, int *art_point)

  Recursively searches for articulation points in the graph using only costs less than or equal to 'max_cost' out from the given node.

- int arrow_bbssp_solve (arrow_problem *problem, arrow_problem_info *info, arrow_bound_result *result)

  Solves the bottleneck biconnected spanning subgraph problem (BBSSP) on the given problem.

- int arrow_bbssp_biconnected (arrow_problem *problem, int max_cost, int *result)

  Determines if the graph is biconnected using only edges with costs less than or equal to the given value.

7.8.1 Detailed Description

Bottleneck biconnected spanning subgraph problem implemenation.

Implemenation of the bottleneck biconnected spanning subgraph problem (BBSSP) that's used as a lower bound for the Bottleneck TSP objective value.

Author:

John LaRusic

Definition in file bbssp.c.

7.8.2 Function Documentation

7.8.2.1 int arrow_bbssp_biconnected (arrow_problem * *problem*, int *max_cost*, int * *result*)

Determines if the graph is biconnected using only edges with costs less than or equal to the given value.

Parameters:

problem [in] problem data

max_cost [in] value to check biconnectivity question against

result [out] ARROW_TRUE if biconnected, ARROW_FALSE otherwise.

Definition at line 94 of file bbssp.c.

References ARROW_ERROR_FATAL, ARROW_FALSE, ARROW_TRUE, arrow_util_create_int_array(), find_art_points(), and arrow_problem::size.

Referenced by arrow_bbssp_solve().

7.8.2.2 int arrow_bbssp_solve (arrow_problem * *problem*, arrow_problem_info * *info*, arrow_bound_result * *result*)

Solves the bottleneck biconnected spanning subgraph problem (BBSSP) on the given problem.

Parameters:

problem [in] problem data

info [in] problem info

result [out] BBSSP solution

Definition at line 44 of file bbssp.c.

References arrow_bbssp_biconnected(), ARROW_ERROR_FATAL, ARROW_FAILURE, arrow_print_error, ARROW_SUCCESS, ARROW_TRUE, arrow_util_zeit(), arrow_problem_info::cost_list, arrow_problem_info::cost_list_length, arrow_bound_result::obj_value, arrow_problem::symmetric, and arrow_bound_result::total_time.

Referenced by main().

7.8.2.3 int find_art_points (arrow_problem * *problem*, int *max_cost*, int *node*, int *depth_num*, int *root_children*, int * *visited*, int * *depth*, int * *low*, int * *parent*, int * *art_point*)

Recursively searches for articulation points in the graph using only costs less than or equal to 'max_cost' out from the given node.

Parameters:

problem [in] problem data structure

max_cost [in] the largest cost to consider as being in the graph

node [in] the node to search outward from

depth_num [in] level at which the given node was first discovered

root_children [in] count of the number of children of the root

visited [out] indicates if a node has been visited or not

depth [out] indicates the discovery depth of a node

low [out] indicates a back edge for some descendent of a node (e.g. the discovery depth of the node closest to the root and reachable from the given node by following zero or more edges downward and then at most one back edge)

parent [out] indicates the closest "parent" of a node

art_point [out] indicates if the node is an articulation point

Definition at line 168 of file bbssp.c.

References ARROW_SUCCESS, arrow_problem::get_cost, and arrow_problem::size.

Referenced by arrow_bbssp_biconnected().

7.9 bin/bscssp.c File Reference

Bottleneck strongly connected spanning subgraph problem solver.

```
#include "arrow.h"
```

Defines

- #define NUM_OPTS 2

Functions

- int main (int argc, char *argv[ ])

Variables

- char * program_name
- char * input_file = NULL
- char * xml_file = NULL
- arrow_option options [NUM_OPTS]
- char * desc = "Bottleneck strongly connected spanning subgraph problem solver"
- char * usage = "-i tsplib.tsp [options] "

7.9.1 Detailed Description

Bottleneck strongly connected spanning subgraph problem solver.

Solves the bottleneck strongly connected spanning subgraph problem (BSCSSP) on the given input file.

Author:

John LaRusic

Definition in file bscssp.c.

7.9.2 Define Documentation

7.9.2.1 #define NUM_OPTS 2

Definition at line 18 of file bscssp.c.

7.9.3 Function Documentation

7.9.3.1 int main (int *argc*, char * *argv*[ ])

Definition at line 31 of file bscssp.c.

References arrow_bscssp_solve(), ARROW_FAILURE, arrow_options_parse(), arrow_print_error, arrow_problem_destruct(), arrow_problem_info_destruct(), arrow_problem_info_get(), arrow_problem_read(), ARROW_SUCCESS, arrow_util_print_program_args(), desc, input_file, NUM_OPTS, arrow_bound_result::obj_value, arrow_bound_result::total_time, usage, and xml_file.

7.9.4 Variable Documentation

7.9.4.1 char* desc = "Bottleneck strongly connected spanning subgraph problem solver"

Definition at line 26 of file bscssp.c.

7.9.4.2 char* input_file = NULL

Given input TSPLIB file

Definition at line 14 of file bscssp.c.

7.9.4.3 arrow_option options[NUM_OPTS]

Initial value:

```
{
    {'i', "input", "TSPLIB input file",
        ARROW_OPTION_STRING, &input_file, ARROW_TRUE, ARROW_TRUE},
    {'x', "xml", "File to write XML output to",
        ARROW_OPTION_STRING, &xml_file, ARROW_FALSE, ARROW_TRUE}
}
```

Definition at line 19 of file bscssp.c.

7.9.4.4 char* program_name

Program name

Definition at line 13 of file bscssp.c.

7.9.4.5 char* usage = "-i tsplib.tsp [options] "

Definition at line 27 of file bscssp.c.

7.9.4.6 char* xml_file = NULL

Definition at line 15 of file bscssp.c.

7.10 lib/bscssp.c File Reference

Bottleneck strongly connected spanning subgraph problem implemenation.

`#include "arrow.h"`

Functions

- int strongly_connected (arrow_problem *problem, int delta, int *result)
  Determines if the given graph is strongly connected.
- void strongly_connected_dfs (arrow_problem *problem, int delta, int i, int transpose, int *visited)
  Performs a recursive depth-first search to test for connectivity.
- int arrow_bscssp_solve (arrow_problem *problem, arrow_problem_info *info, arrow_bound_result *result)
  Solves the bottleneck strongly connected spanning subgraph problem (BSCSSP) on the given graph.

7.10.1 Detailed Description

Bottleneck strongly connected spanning subgraph problem implemenation.

Implemenation of the bottleneck strongly connected spanning subgraph problem (BSCSSP) that's used as a lower bound for the Bottleneck TSP objective value.

Author:

John LaRusic

Definition in file bscssp.c.

7.10.2 Function Documentation

7.10.2.1 int arrow_bscssp_solve (arrow_problem * *problem*, arrow_problem_info * *info*, arrow_bound_result * *result*)

Solves the bottleneck strongly connected spanning subgraph problem (BSCSSP) on the given graph.

Parameters:

problem [in] problem data

info [in] problem info

result [out] BSCSSP solution

Definition at line 43 of file bscssp.c.

References ARROW_ERROR_FATAL, ARROW_FAILURE, ARROW_SUCCESS, ARROW_TRUE, arrow_util_zeit(), arrow_problem_info::cost_list, arrow_problem_info::cost_list_length, arrow_bound_result::obj_value, strongly_connected(), and arrow_bound_result::total_time.

Referenced by main().

7.10.2.2 int strongly_connected (arrow_problem * *problem*, int *delta*, int * *result*)

Determines if the given graph is strongly connected.

Parameters:

problem [in] the problem instance

delta [in] delta parameter

result [out] the result!

Definition at line 86 of file bscssp.c.

References ARROW_ERROR_FATAL, ARROW_FALSE, ARROW_SUCCESS, ARROW_TRUE, arrow_util_create_int_array(), arrow_problem::size, and strongly_connected_dfs().

Referenced by arrow_bscssp_solve().

7.10.2.3 void strongly_connected_dfs (arrow_problem * *problem*, int *delta*, int *i*, int *transpose*, int * *visited*)

Performs a recursive depth-first search to test for connectivity.

Parameters:

problem [in] the problem instance

delta [in] delta parameter

i [in] node index to search from

transpose [in] if true, calculates costs with transposed cost matrix

visited [out] array that marks nodes that have been visited

Definition at line 133 of file bscssp.c.

References ARROW_TRUE, arrow_problem::get_cost, and arrow_problem::size.

Referenced by strongly_connected().

7.11 bin/btsp.c File Reference

Bottleneck TSP heuristic.

```
#include "arrow.h"
```

Defines

- #define NUM_OPTS 11
- #define SOLVE_STEPS 1

Functions

- int main (int argc, char *argv[ ])

Variables

- char * input_file = NULL
- char * xml_file = NULL
- int random_restarts = -1
- int stall_count = -1
- int kicks = -1
- int confirm_sol = ARROW_FALSE
- int supress_ebst = ARROW_FALSE
- int find_short_tour = ARROW_FALSE
- int lower_bound = -1
- int upper_bound = INT_MAX
- int basic_attempts = ARROW_DEFAULT_BASIC_ATTEMPTS
- arrow_option options [NUM_OPTS]
- char * desc = "Bottleneck traveling salesman problem (BTSP) solver"
- char * usage = "-i tsplib.tsp [options]"

7.11.1 Detailed Description

Bottleneck TSP heuristic.

Runs the Bottleneck TSP heuristic on the given input file.

Author:

John LaRusic

Definition in file btsp.c.

7.11.2 Define Documentation

7.11.2.1 #define NUM_OPTS 11

Definition at line 25 of file btsp.c.

7.11.2.2 #define SOLVE_STEPS 1

7.11.3 Function Documentation

7.11.3.1 int main (int *argc*, char * *argv*[ ])

Definition at line 58 of file btsp.c.

References arrow_bbssp_solve(), arrow_btsp_fun_basic(), arrow_btsp_fun_destruct(), arrow_btsp_-params_destruct(), arrow_btsp_params_init(), arrow_btsp_result_destruct(), arrow_btsp_result_init(), arrow_btsp_solve(), ARROW_BTSP_SOLVE_PLAN_BASIC, ARROW_FAILURE, ARROW_FALSE, arrow_options_parse(), arrow_print_error, arrow_problem_destruct(), arrow_problem_info_get(), arrow_problem_read(), ARROW_SUCCESS, ARROW_TRUE, arrow_tsp_lk_params_destruct(), arrow_-tsp_lk_params_init(), arrow_util_print_program_args(), arrow_util_zeit(), basic_attempts, arrow_btsp_-result::bin_search_steps, confirm_sol, arrow_btsp_params::confirm_sol, arrow_problem_info::cost_list_-length, desc, arrow_btsp_result::exact_attempts, arrow_btsp_result::exact_time, find_short_tour, arrow_-btsp_params::find_short_tour, arrow_btsp_result::found_tour, input_file, arrow_tsp_lk_params::kicks, kicks, arrow_btsp_result::linkern_attempts, arrow_btsp_result::linkern_time, arrow_btsp_params::lower_-bound, lower_bound, NUM_OPTS, arrow_btsp_params::num_steps, arrow_btsp_result::obj_value, arrow_bound_result::obj_value, arrow_btsp_result::optimal, arrow_tsp_lk_params::random_restarts, random_restarts, SOLVE_STEPS, arrow_tsp_lk_params::stall_count, stall_count, arrow_btsp_-params::steps, supress_ebst, arrow_btsp_params::supress_ebst, arrow_problem::symmetric, arrow_-btsp_result::total_time, arrow_bound_result::total_time, arrow_btsp_result::tour_length, upper_bound, arrow_btsp_params::upper_bound, usage, and xml_file.

7.11.4 Variable Documentation

7.11.4.1 int basic_attempts = ARROW_DEFAULT_BASIC_ATTEMPTS

Definition at line 22 of file btsp.c.

7.11.4.2 int confirm_sol = ARROW_FALSE

Definition at line 17 of file btsp.c.

7.11.4.3 char* desc = "Bottleneck traveling salesman problem (BTSP) solver"

Definition at line 53 of file btsp.c.

7.11.4.4 int find_short_tour = ARROW_FALSE

Definition at line 19 of file btsp.c.

7.11.4.5 char* input_file = NULL

Definition at line 12 of file btsp.c.

7.11.4.6 int kicks = -1

Definition at line 16 of file btsp.c.

7.11.4.7 int lower_bound = -1

Definition at line 20 of file btsp.c.

7.11.4.8 arrow_option options[NUM_OPTS]

Initial value:

```
{
    {'i', "input", "TSPLIB input file",
        ARROW_OPTION_STRING, &input_file, ARROW_TRUE, ARROW_TRUE},
    {'x', "xml", "file to write XML output to",
        ARROW_OPTION_STRING, &xml_file, ARROW_FALSE, ARROW_TRUE},

    {'r', "restarts", "number of random restarts",
        ARROW_OPTION_INT, &random_restarts, ARROW_FALSE, ARROW_TRUE},
    {'s', "stall-count", "max number of 4-swaps w/o progress",
        ARROW_OPTION_INT, &stall_count, ARROW_FALSE, ARROW_TRUE},
    {'k', "kicks", "number of 4-swap kicks",
        ARROW_OPTION_INT, &kicks, ARROW_FALSE, ARROW_TRUE},
    {'c', "confirm-solution", "confirm solution with exact solver",
        ARROW_OPTION_INT, &confirm_sol, ARROW_FALSE, ARROW_FALSE},
    {'e', "supress-ebst", "supress binary search",
        ARROW_OPTION_INT, &supress_ebst, ARROW_FALSE, ARROW_FALSE},
    {'S', "find-short-tour", "finds a (relatively) short BTSP tour",
        ARROW_OPTION_INT, &find_short_tour, ARROW_FALSE, ARROW_FALSE},

    {'l', "lower-bound", "initial lower bound",
        ARROW_OPTION_INT, &lower_bound, ARROW_FALSE, ARROW_TRUE},
    {'u', "upper-bound", "initial upper bound",
        ARROW_OPTION_INT, &upper_bound, ARROW_FALSE, ARROW_TRUE},
    {'a', "basic-attempts", "number of basic attempts",
        ARROW_OPTION_INT, &basic_attempts, ARROW_FALSE, ARROW_TRUE}
}
```

Definition at line 26 of file btsp.c.

7.11.4.9 int random_restarts = -1

Definition at line 14 of file btsp.c.

7.11.4.10 int stall_count = -1

Definition at line 15 of file btsp.c.

7.11.4.11 int supress_ebst = ARROW_FALSE

Definition at line 18 of file btsp.c.

7.11.4.12 int upper_bound = INT_MAX

Definition at line 21 of file btsp.c.

7.11.4.13 char* usage = "-i tsplib.tsp [options]"

Definition at line 54 of file btsp.c.

7.11.4.14 char* xml_file = NULL

Definition at line 13 of file btsp.c.

7.12 lib/btsp.c File Reference

Bottleneck traveling salesman problem (BTSP) methods.

```
#include "arrow.h"
```

Functions

- int feasible (arrow_problem *problem, int num_steps, arrow_btsp_solve_plan *steps, int delta, int *tour_exists, arrow_btsp_result *result)

  Solves the feasibility problem which attempts to determine if there is a Hamiltonian cycle using costs <= delta.

- int arrow_btsp_result_init (arrow_problem *problem, arrow_btsp_result *result)

  Initializes the BTSP result structure.

- void arrow_btsp_result_destruct (arrow_btsp_result *result)

  Destructs a BTSP result structure.

- void arrow_btsp_params_init (arrow_btsp_params *params)

  Inititalizes BTSP parameter structure.

- void arrow_btsp_params_destruct (arrow_btsp_params *params)

  Destructs a BTSP parameters structure.

- void arrow_btsp_solve_plan_init (arrow_btsp_solve_plan *plan)

  Inititalizes BTSP solve plan structure.

- void arrow_btsp_solve_plan_destruct (arrow_btsp_solve_plan *plan)

  Destructs a BTSP solve plan structure.

- int arrow_btsp_solve (arrow_problem *problem, arrow_problem_info *info, arrow_btsp_params *params, arrow_btsp_result *result)

  Solves TSP with Concorde's exact solver.

7.12.1 Detailed Description

Bottleneck traveling salesman problem (BTSP) methods.

Heuristic for solving the bottleneck traveling salesman problem (BTSP).

Author:

John LaRusic

Definition in file btsp.c.

7.12.2 Function Documentation

7.12.2.1 void arrow_btsp_params_destruct (arrow_btsp_params * *params*)

Destructs a BTSP parameters structure.

Parameters:

params [out] BTSP parameters structure

Definition at line 71 of file btsp.c.

References arrow_btsp_solve_plan_destruct(), arrow_btsp_params::num_steps, and arrow_btsp_-params::steps.

Referenced by main().

7.12.2.2 void arrow_btsp_params_init (arrow_btsp_params * *params*)

Inititalizes BTSP parameter structure.

Parameters:

params [out] BTSP parameters structure

Definition at line 60 of file btsp.c.

References ARROW_FALSE, arrow_btsp_params::confirm_sol, arrow_btsp_params::find_short_tour, arrow_btsp_params::lower_bound, arrow_btsp_params::num_steps, arrow_btsp_params::supress_ebst, and arrow_btsp_params::upper_bound.

Referenced by main().

7.12.2.3 void arrow_btsp_result_destruct (arrow_btsp_result * *result*)

Destructs a BTSP result structure.

Parameters:

result [out] BTSP result structure

Definition at line 53 of file btsp.c.

References arrow_btsp_result::tour.

Referenced by arrow_btsp_solve(), and main().

7.12.2.4 int arrow_btsp_result_init (arrow_problem * *problem*, arrow_btsp_result * *result*)

Initializes the BTSP result structure.

Parameters:

problem [in] problem to solve

result [out] BTSP result structure

Definition at line 33 of file btsp.c.

References ARROW_FAILURE, ARROW_FALSE, ARROW_SUCCESS, arrow_util_create_int_array(), arrow_btsp_result::bin_search_steps, arrow_btsp_result::exact_attempts, arrow_btsp_result::exact_time, arrow_btsp_result::found_tour, arrow_btsp_result::linkern_attempts, arrow_btsp_result::linkern_time, arrow_btsp_result::obj_value, arrow_problem::size, arrow_btsp_result::total_time, arrow_btsp_result::tour, and arrow_btsp_result::tour_length.

Referenced by arrow_btsp_solve(), and main().

7.12.2.5 int arrow_btsp_solve (arrow_problem * *problem*, arrow_problem_info * *info*, arrow_btsp_params * *params*, arrow_btsp_result * *result*)

Solves TSP with Concorde's exact solver.

Parameters:

problem [in] problem to solve

info [in] extra problem info

params [in] parameters for solver (can be NULL)

result [out] BTSP solution

Definition at line 98 of file btsp.c.

References arrow_btsp_result_destruct(), arrow_btsp_result_init(), arrow_debug, ARROW_FAILURE, ARROW_FALSE, arrow_print_error, ARROW_SUCCESS, ARROW_TRUE, arrow_util_binary_search(), arrow_util_zeit(), arrow_btsp_result::bin_search_steps, arrow_btsp_params::confirm_sol, arrow_problem_info::cost_list, arrow_problem_info::cost_list_length, arrow_btsp_result::exact_attempts, arrow_btsp_result::exact_time, feasible(), arrow_btsp_params::find_short_tour, arrow_btsp_result::found_tour, arrow_btsp_result::linkern_attempts, arrow_btsp_result::linkern_time, arrow_btsp_params::lower_bound, arrow_btsp_params::num_steps, arrow_btsp_result::obj_value, arrow_btsp_result::optimal, arrow_problem::size, arrow_btsp_params::steps, arrow_btsp_params::supress_ebst, arrow_problem::symmetric, arrow_btsp_result::total_time, arrow_btsp_result::tour, arrow_btsp_result::tour_length, upper_bound, and arrow_btsp_params::upper_bound.

Referenced by main().

7.12.2.6 void arrow_btsp_solve_plan_destruct (arrow_btsp_solve_plan * *plan*)

Destructs a BTSP solve plan structure.

Parameters:

plan [out] BTSP solve plan structure

Definition at line 91 of file btsp.c.

References arrow_btsp_fun_destruct(), arrow_tsp_lk_params_destruct(), arrow_btsp_solve_plan::fun, and arrow_btsp_solve_plan::lk_params.

Referenced by arrow_btsp_params_destruct().

7.12.2.7 void arrow_btsp_solve_plan_init (arrow_btsp_solve_plan * *plan*)

Inititalizes BTSP solve plan structure.

Parameters:

plan [out] BTSP solve plan structure

Definition at line 84 of file btsp.c.

References arrow_btsp_solve_plan::attempts, and arrow_btsp_solve_plan::plan_type.

7.12.2.8 int feasible (arrow_problem * *problem*, int *num_steps*, arrow_btsp_solve_plan * *steps*, int *delta*, int * *tour_exists*, arrow_btsp_result * *result*)

Solves the feasibility problem which attempts to determine if there is a Hamiltonian cycle using costs <= delta.

Parameters:

problem [in] problem to solve

num_steps [in] total number of steps in solve plan

steps [in] solve plan step details

delta [in] delta parameter for feasibility problem.

tour_exists [out] true if a feasible tour exists, false otherwise

result [out] resulting BTSP tour found

Definition at line 275 of file btsp.c.

References arrow_btsp_fun_apply(), arrow_debug, ARROW_FAILURE, ARROW_FALSE, arrow_problem_destruct(), ARROW_SUCCESS, ARROW_TRUE, arrow_tsp_exact_solve(), arrow_tsp_lk_solve(), arrow_tsp_result_destruct(), arrow_tsp_result_init(), arrow_btsp_solve_plan::attempts, arrow_btsp_result::exact_attempts, arrow_btsp_result::exact_time, arrow_btsp_fun::feasible, arrow_btsp_result::found_tour, arrow_btsp_solve_plan::fun, arrow_problem::get_cost, arrow_btsp_result::linkern_attempts, arrow_btsp_result::linkern_time, arrow_btsp_solve_plan::lk_params, arrow_tsp_result::obj_value, arrow_btsp_result::obj_value, arrow_btsp_solve_plan::plan_type, arrow_problem::size, arrow_tsp_result::total_time, arrow_btsp_result::total_time, arrow_btsp_result::tour, arrow_tsp_result::tour, arrow_btsp_result::tour_length, arrow_btsp_solve_plan::upper_bound_update, and arrow_btsp_solve_plan::use_exact_solver.

Referenced by arrow_btsp_solve().

7.13 bin/cbtsp.c File Reference

Constrained Bottleneck TSP heuristic.

```
#include "arrow.h"
```

Defines

- #define NUM_OPTS 17
- #define SOLVE_STEPS 2

Functions

- int main (int argc, char *argv[ ])

Variables

- char * input_file = NULL
- char * xml_file = NULL
- char * tour_file = NULL
- double length = DBL_MAX
- int random_restarts = 5
- int stall_count = -1
- int kicks = -1
- int confirm_sol = ARROW_FALSE
- int supress_ebst = ARROW_FALSE
- int find_short_tour = ARROW_FALSE
- int lower_bound = -1
- int upper_bound = INT_MAX
- int basic_attempts = 3
- int shake_attempts = 2
- int shake_rand_min = 0
- int shake_rand_max = -1
- int random_seed = 0
- arrow_option options [NUM_OPTS]
- char * desc = "Bottleneck traveling salesman problem (BTSP) solver"
- char * usage = "-i tsplib.tsp -L max_length [options]"

7.13.1 Detailed Description

Constrained Bottleneck TSP heuristic.

Runs the Constrained Bottleneck TSP heuristic on the given input file.

Author:

John LaRusic

Definition in file cbtsp.c.

7.13.2 Define Documentation

7.13.2.1 #define NUM_OPTS 17

Definition at line 32 of file cbtsp.c.

7.13.2.2 #define SOLVE_STEPS 2

7.13.3 Function Documentation

7.13.3.1 int main (int *argc*, char * *argv*[ ])

Definition at line 81 of file cbtsp.c.

References arrow_bbssp_solve(), arrow_btsp_fun_constrained(), arrow_btsp_fun_constrained_shake(), arrow_btsp_params_destruct(), arrow_btsp_params_init(), arrow_btsp_result_destruct(), arrow_btsp_result_init(), arrow_btsp_solve(), ARROW_BTSP_SOLVE_PLAN_CONSTRAINED, ARROW_BTSP_SOLVE_PLAN_CONSTRAINED_SHAKE, ARROW_FAILURE, ARROW_FALSE, arrow_options_parse(), arrow_print_error, arrow_problem_destruct(), arrow_problem_info_get(), arrow_problem_read(), ARROW_TRUE, arrow_tsp_lk_params_destruct(), arrow_tsp_lk_params_init(), arrow_util_print_program_args(), arrow_util_random_seed(), arrow_util_write_tour(), arrow_util_zeit(), basic_attempts, arrow_btsp_result::bin_search_steps, confirm_sol, arrow_btsp_params::confirm_sol, arrow_problem_info::cost_list_length, desc, arrow_btsp_result::exact_attempts, arrow_btsp_result::exact_time, find_short_tour, arrow_btsp_params::find_short_tour, arrow_btsp_result::found_tour, arrow_problem::get_cost, input_file, arrow_tsp_lk_params::kicks, kicks, length, arrow_tsp_lk_params::length_bound, arrow_btsp_result::linkern_attempts, arrow_btsp_result::linkern_time, arrow_btsp_params::lower_bound, lower_bound, NUM_OPTS, arrow_btsp_params::num_steps, arrow_btsp_result::obj_value, arrow_bound_result::obj_value, arrow_btsp_result::optimal, arrow_tsp_lk_params::random_restarts, random_restarts, random_seed, shake_attempts, shake_rand_max, shake_rand_min, arrow_problem::size, SOLVE_STEPS, arrow_tsp_lk_params::stall_count, stall_count, arrow_btsp_params::steps, supress_ebst, arrow_btsp_params::supress_ebst, arrow_btsp_result::total_time, arrow_bound_result::total_time, arrow_btsp_result::tour, tour_file, arrow_btsp_result::tour_length, upper_bound, arrow_btsp_params::upper_bound, usage, and xml_file.

7.13.4 Variable Documentation

7.13.4.1 int basic_attempts = 3

Definition at line 24 of file cbtsp.c.

7.13.4.2 int confirm_sol = ARROW_FALSE

Definition at line 19 of file cbtsp.c.

7.13.4.3 char* desc = "Bottleneck traveling salesman problem (BTSP) solver"

Definition at line 76 of file cbtsp.c.

7.13.4.4 int find_short_tour = ARROW_FALSE

Definition at line 21 of file cbtsp.c.

7.13.4.5 char* input_file = NULL

Definition at line 12 of file cbtsp.c.

7.13.4.6 int kicks = -1

Definition at line 18 of file cbtsp.c.

7.13.4.7 double length = DBL_MAX

Definition at line 15 of file cbtsp.c.

Referenced by main().

7.13.4.8 int lower_bound = -1

Definition at line 22 of file cbtsp.c.

7.13.4.9 arrow_option options[NUM_OPTS]

Definition at line 33 of file cbtsp.c.

7.13.4.10 int random_restarts = 5

Definition at line 16 of file cbtsp.c.

7.13.4.11 int random_seed = 0

Definition at line 28 of file cbtsp.c.

Referenced by main().

7.13.4.12 int shake_attempts = 2

Definition at line 25 of file cbtsp.c.

Referenced by main().

7.13.4.13 int shake_rand_max = -1

Definition at line 27 of file cbtsp.c.

Referenced by main().

7.13.4.14 int shake_rand_min = 0

Definition at line 26 of file cbtsp.c.

Referenced by main().

7.13.4.15 int stall_count = -1

Definition at line 17 of file cbtsp.c.

7.13.4.16 int supress_ebst = ARROW_FALSE

Definition at line 20 of file cbtsp.c.

7.13.4.17 char* tour_file = NULL

Definition at line 14 of file cbtsp.c.

7.13.4.18 int upper_bound = INT_MAX

Definition at line 23 of file cbtsp.c.

7.13.4.19 char* usage = "-i tsplib.tsp -L max_length [options]"

Definition at line 77 of file cbtsp.c.

7.13.4.20 char* xml_file = NULL

Definition at line 13 of file cbtsp.c.

7.14 bin/dcbpb.c File Reference

Degree Constrained Bottleneck Path Bound solver.

```
#include "arrow.h"
```

Defines

- #define NUM_OPTS 2

Functions

- int main (int argc, char *argv[ ])

Variables

- char * program_name
- char * input_file = NULL
- char * xml_file = NULL
- arrow_option options [NUM_OPTS]
- char * desc = "Degree Constrained Bottleneck Path Bound solver"
- char * usage = "-i tsplib.tsp [options] "

7.14.1 Detailed Description

Degree Constrained Bottleneck Path Bound solver.

Solves the Degree Constrained Bottleneck Path Bound (DCBPB) on the given input file.

Author:

John LaRusic

Definition in file dcbpb.c.

7.14.2 Define Documentation

7.14.2.1 #define NUM_OPTS 2

Definition at line 18 of file dcbpb.c.

7.14.3 Function Documentation

7.14.3.1 int main (int *argc*, char * *argv*[ ])

Definition at line 31 of file dcbpb.c.

References arrow_dcbpb_solve(), ARROW_FAILURE, arrow_options_parse(), arrow_print_error, arrow_problem_destruct(), arrow_problem_read(), ARROW_SUCCESS, arrow_util_print_program_-args(), desc, input_file, NUM_OPTS, arrow_bound_result::obj_value, arrow_bound_result::total_time, usage, and xml_file.

7.14.4 Variable Documentation

7.14.4.1 char* desc = "Degree Constrained Bottleneck Path Bound solver"

Definition at line 26 of file dcbpb.c.

7.14.4.2 char* input_file = NULL

Given input TSPLIB file

Definition at line 14 of file dcbpb.c.

7.14.4.3 arrow_option options[NUM_OPTS]

Initial value:

```
{
    {'i', "input", "TSPLIB input file",
        ARROW_OPTION_STRING, &input_file, ARROW_TRUE, ARROW_TRUE},
    {'x', "xml", "File to write XML output to",
        ARROW_OPTION_STRING, &xml_file, ARROW_FALSE, ARROW_TRUE}
}
```

Definition at line 19 of file dcbpb.c.

7.14.4.4 char* program_name

Program name

Definition at line 13 of file dcbpb.c.

7.14.4.5 char* usage = "-i tsplib.tsp [options] "

Definition at line 27 of file dcbpb.c.

7.14.4.6 char* xml_file = NULL

Definition at line 15 of file dcbpb.c.

7.15 lib/dcbpb.c File Reference

Degree constarined bottleneck paths bound.

```
#include "arrow.h"
```

Functions

- void bottleneck_paths (arrow_problem *problem, int ignore, int **b)

  Solves the all-pairs bottleneck paths problem (a simple modification of the Floyd-Warshall alg for all-pairs shortest paths).

- int max (int i, int j, int k)
- int arrow_dcbpb_solve (arrow_problem *problem, arrow_bound_result *result)

  Solves the degree constrained bottleneck paths bound (DCBPB).

7.15.1 Detailed Description

Degree constarined bottleneck paths bound.

Implemenation of the degree constrained bottleneck paths bound (DCBPB) used as a lower bound for the Bottleneck TSP objective value.

Author:

John LaRusic

Definition in file dcbpb.c.

7.15.2 Function Documentation

7.15.2.1 int arrow_dcbpb_solve (arrow_problem * *problem*, arrow_bound_result * *result*)

Solves the degree constrained bottleneck paths bound (DCBPB).

Parameters:

problem [in] problem data

result [out] BPB solution

Definition at line 38 of file dcbpb.c.

References ARROW_FAILURE, ARROW_SUCCESS, arrow_util_create_int_matrix(), arrow_util_zeit(), bottleneck_paths(), arrow_problem::get_cost, max(), arrow_bound_result::obj_value, arrow_problem::size, and arrow_bound_result::total_time.

Referenced by main().

7.15.2.2 void bottleneck_paths (arrow_problem * *problem*, int *ignore*, int ** *b*)

Solves the all-pairs bottleneck paths problem (a simple modification of the Floyd-Warshall alg for all-pairs shortest paths).

Parameters:

problem [in] problem data

ignore [in] vertex number to ignore

b [out] array will hold bottleneck path value for each pair of source/sink nodes

Definition at line 107 of file dcbpb.c.

References arrow_problem::get_cost, max(), and arrow_problem::size.

Referenced by arrow_dcbpb_solve().

7.15.2.3 int max (int *i*, int *j*, int *k*)

Definition at line 144 of file dcbpb.c.

Referenced by arrow_2mb_solve(), arrow_dcbpb_solve(), and bottleneck_paths().

7.16 bin/hash.c File Reference

Hash testing.

```
#include "arrow.h"
```

Defines

- #define NUM_OPTS 1

Functions

- int main (int argc, char ∗argv[ ])

Variables

- char ∗ program_name
- char ∗ input_file = NULL
- arrow_option options [NUM_OPTS]
- char ∗ desc = "Tests the hashing functions."
- char ∗ usage = "-i tsplib.tsp"

7.16.1 Detailed Description

Hash testing.

Tests hashing functions.

Author:

John LaRusic

Definition in file hash.c.

7.16.2 Define Documentation

7.16.2.1 #define NUM_OPTS 1

Definition at line 16 of file hash.c.

7.16.3 Function Documentation

7.16.3.1 int main (int *argc*, char ∗ *argv*[ ])

Definition at line 27 of file hash.c.

References arrow_options_parse(), arrow_print_error, arrow_problem_destruct(), arrow_problem_info_destruct(), arrow_problem_info_get(), arrow_problem_read(), arrow_problem_info::cost_list, arrow_problem_info::cost_list_length, desc, arrow_problem::get_cost, input_file, NUM_OPTS, arrow_problem::size, and usage.

7.16.4 Variable Documentation

7.16.4.1 char* desc = "Tests the hashing functions."

Definition at line 22 of file hash.c.

7.16.4.2 char* input_file = NULL

Given input TSPLIB file

Definition at line 13 of file hash.c.

7.16.4.3 arrow_option options[NUM_OPTS]

Initial value:

```
{
    {'i', "input", "TSPLIB input file",
        ARROW_OPTION_STRING, &input_file, ARROW_TRUE, ARROW_TRUE}
}
```

Definition at line 17 of file hash.c.

7.16.4.4 char* program_name

Program name

Definition at line 12 of file hash.c.

7.16.4.5 char* usage = "-i tsplib.tsp"

Definition at line 23 of file hash.c.

7.17 bin/histogram_data.c File Reference

Edge length histogram data collector.

```
#include "arrow.h"
```

Defines

- #define NUM_OPTS 1

Functions

- int main (int argc, char *argv[ ])

Variables

- char * program_name
- char * input_file = NULL
- arrow_option options [NUM_OPTS]
- char * desc = "Prints a list of every cost in problem (for histogram.py)"
- char * usage = "-i tsplib.tsp"

7.17.1 Detailed Description

Edge length histogram data collector.

Prints out a list of every edge length present in given problem. Used in conjunction with a Python script for generating a histogram plot.

Author:

John LaRusic

Definition in file histogram_data.c.

7.17.2 Define Documentation

7.17.2.1 #define NUM_OPTS 1

Definition at line 17 of file histogram_data.c.

7.17.3 Function Documentation

7.17.3.1 int main (int *argc*, char * *argv*[ ])

Definition at line 28 of file histogram_data.c.

References ARROW_DEV_NULL, arrow_options_parse(), arrow_problem_destruct(), arrow_problem_read(), arrow_util_redirect_stdout_to_file(), arrow_util_restore_stdout(), desc, arrow_problem::get_cost, input_file, NUM_OPTS, arrow_problem::size, and usage.

7.17.4 Variable Documentation

7.17.4.1 char* desc = "Prints a list of every cost in problem (for histogram.py)"

Definition at line 23 of file histogram_data.c.

7.17.4.2 char* input_file = NULL

Given input TSPLIB file

Definition at line 14 of file histogram_data.c.

7.17.4.3 arrow_option options[NUM_OPTS]

Initial value:

```
{
    {'i', "input", "TSPLIB input file",
        ARROW_OPTION_STRING, &input_file, ARROW_TRUE, ARROW_TRUE}
}
```

Definition at line 18 of file histogram_data.c.

7.17.4.4 char* program_name

Program name

Definition at line 13 of file histogram_data.c.

7.17.4.5 char* usage = "-i tsplib.tsp"

Definition at line 24 of file histogram_data.c.

7.18 bin/linkern.c File Reference

Lin-Kernighan TSP heuristic.

```
#include "arrow.h"
#include <getopt.h>
```

Functions

- void print_help ()
  Prints help/usage message.
- void print_version ()
  Prints version message.
- void print_usage ()
  Prints usage message.
- void read_args (int argc, char *argv[ ])
  Reads program arguments.
- int main (int argc, char *argv[ ])

Variables

- char * program_name
- char * input_file
- int random_restarts = -1
- int stall_count = -1
- int kicks = -1

7.18.1 Detailed Description

Lin-Kernighan TSP heuristic.

Runs the Lin-Kernighan TSP heuristic on the given input file. This is really nothing more than a wrapper to Concorde, and is for testing purposes only. Use Concorde's executable for access to solve options.

Author:

John LaRusic

Definition in file linkern.c.

7.18.2 Function Documentation

7.18.2.1 int main (int *argc*, char * *argv*[ ])

Definition at line 49 of file linkern.c.

References arrow_print_error, arrow_problem_destruct(), arrow_problem_read(), arrow_tsp_lk_params_-destruct(), arrow_tsp_lk_params_init(), arrow_tsp_lk_solve(), arrow_tsp_result_destruct(), arrow_tsp_-result_init(), arrow_tsp_result::found_tour, input_file, arrow_tsp_lk_params::kicks, kicks, arrow_tsp_-result::obj_value, program_name, arrow_tsp_lk_params::random_restarts, random_restarts, read_args(), arrow_tsp_lk_params::stall_count, stall_count, and arrow_tsp_result::total_time.

7.18.2.2 void print_help ()

Prints help/usage message.

Definition at line 90 of file linkern.c.

References print_usage(), and program_name.

Referenced by arrow_options_parse(), and read_args().

7.18.2.3 void print_usage ()

Prints usage message.

Definition at line 120 of file linkern.c.

References program_name.

Referenced by arrow_options_parse(), print_help(), and read_args().

7.18.2.4 void print_version ()

Prints version message.

Definition at line 105 of file linkern.c.

References program_name.

Referenced by arrow_options_parse(), and read_args().

7.18.2.5 void read_args (int *argc*, char * *argv*[ ])

Reads program arguments.

Definition at line 126 of file linkern.c.

References arrow_print_error, input_file, kicks, print_help(), print_usage(), print_version(), random_-restarts, and stall_count.

Referenced by main().

7.18.3 Variable Documentation

7.18.3.1 char* input_file

Given input TSPLIB file

Definition at line 40 of file linkern.c.

7.18.3.2 int kicks = -1

Definition at line 43 of file linkern.c.

7.18.3.3 char* program_name

Program name

Definition at line 39 of file linkern.c.

7.18.3.4 int random_restarts = -1

Definition at line 41 of file linkern.c.

7.18.3.5 int stall_count = -1

Definition at line 42 of file linkern.c.

7.19 bin/subproblem.c File Reference

Sub-problem generator.

```
#include "arrow.h"
```

Defines

- #define NUM_OPTS 3

Functions

- int main (int argc, char *argv[ ])

Variables

- char * program_name
- char * input_file = NULL
- int start = 0
- int end = 1
- arrow_option options [NUM_OPTS]
- char * desc = "Generates a sub-problem from a larger one"
- char * usage = "-i tsplib.tsp -s # -e #"

7.19.1 Detailed Description

Sub-problem generator.

Generates a sub-problem from a larger one by outputing data for nodes on the interval [start, end].

Author:

John LaRusic

Definition in file subproblem.c.

7.19.2 Define Documentation

7.19.2.1 #define NUM_OPTS 3

Definition at line 19 of file subproblem.c.

7.19.3 Function Documentation

7.19.3.1 int main (int *argc*, char * *argv*[ ])

Definition at line 34 of file subproblem.c.

References ARROW_DEV_NULL, arrow_options_parse(), arrow_print_error, arrow_problem_destruct(), arrow_problem_read(), arrow_util_redirect_stdout_to_file(), arrow_util_restore_stdout(), desc, end,

arrow_problem::get_cost, input_file, arrow_problem::name, NUM_OPTS, arrow_problem::size, start, arrow_problem::symmetric, and usage.

7.19.4 Variable Documentation

7.19.4.1 char* desc = "Generates a sub-problem from a larger one"

Definition at line 29 of file subproblem.c.

7.19.4.2 int end = 1

Ending city index

Definition at line 16 of file subproblem.c.

Referenced by main().

7.19.4.3 char* input_file = NULL

Given input TSPLIB file

Definition at line 14 of file subproblem.c.

7.19.4.4 arrow_option options[NUM_OPTS]

Initial value:

```
{
    {'i', "input", "TSPLIB input file",
        ARROW_OPTION_STRING, &input_file, ARROW_TRUE, ARROW_TRUE},
    {'s', "start", "TSPLIB input file",
        ARROW_OPTION_INT, &start, ARROW_FALSE, ARROW_TRUE},
    {'e', "end", "TSPLIB input file",
        ARROW_OPTION_INT, &end, ARROW_TRUE, ARROW_TRUE}
}
```

Definition at line 20 of file subproblem.c.

7.19.4.5 char* program_name

Program name

Definition at line 13 of file subproblem.c.

7.19.4.6 int start = 0

Starting city index

Definition at line 15 of file subproblem.c.

Referenced by main().

7.19.4.7 char* usage = "-i tsplib.tsp -s # -e #"

Definition at line 30 of file subproblem.c.

7.20 bin/tour_info.c File Reference

Tour information.

```
#include "arrow.h"
#include <getopt.h>
```

Functions

- void print_help ()

  Prints help/usage message.

- void print_version ()

  Prints version message.

- void print_usage ()

  Prints usage message.

- void read_args (int argc, char ∗argv[ ])

  Reads program arguments.

- void print_xml_output (double length, int max_cost, int min_cost, int argc, char ∗argv[ ])

  Prints output in XML format.

- int main (int argc, char ∗argv[ ])

Variables

- char ∗ program_name
- char ∗ problem_file
- char ∗ tour_file
- int xml_output = ARROW_FALSE

7.20.1 Detailed Description

Tour information.

Displays tour information for a given problem input and tour input

Author:

John LaRusic

Definition in file tour_info.c.

7.20.2 Function Documentation

7.20.2.1 int main (int *argc*, char * *argv*[ ])

Definition at line 53 of file tour_info.c.

References ARROW_DEV_NULL, arrow_print_error, arrow_problem_destruct(), arrow_problem_read(), arrow_problem_read_tour(), ARROW_SUCCESS, arrow_util_create_int_array(), arrow_util_redirect_stdout_to_file(), arrow_util_restore_stdout(), arrow_problem::get_cost, length, print_xml_output(), problem_file, program_name, read_args(), arrow_problem::size, tour_file, and xml_output.

7.20.2.2 void print_help ()

Prints help/usage message.

7.20.2.3 void print_usage ()

Prints usage message.

7.20.2.4 void print_version ()

Prints version message.

7.20.2.5 void print_xml_output (double *length*, int *max_cost*, int *min_cost*, int *argc*, char * *argv*[ ])

Prints output in XML format.

Definition at line 214 of file tour_info.c.

References problem_file, and tour_file.

Referenced by main().

7.20.2.6 void read_args (int *argc*, char * *argv*[ ])

Reads program arguments.

7.20.3 Variable Documentation

7.20.3.1 char* problem_file

Given TSPLIB problem file

Definition at line 45 of file tour_info.c.

Referenced by main(), and print_xml_output().

7.20.3.2 char* program_name

Program name

Definition at line 44 of file tour_info.c.

7.20.3.3 char* tour_file

Given TSPLIB tour file

Definition at line 46 of file tour_info.c.

7.20.3.4 int xml_output = ARROW_FALSE

Output output in XML format (or not)

Definition at line 47 of file tour_info.c.

Referenced by main().

7.21 bin/tsp.c File Reference

Traveling Salesman Problem solver.

```
#include "arrow.h"
#include <getopt.h>
```

Functions

- void print_help ()
  Prints help/usage message.
- void print_version ()
  Prints version message.
- void print_usage ()
  Prints usage message.
- void read_args (int argc, char *argv[ ])
  Reads program arguments.
- int main (int argc, char *argv[ ])

Variables

- char * program_name
- char * input_file

7.21.1 Detailed Description

Traveling Salesman Problem solver.

Solves the traveling salesman problem (TSP) on the given input file. This is really nothing more than a wrapper to Concorde, and is for testing purposes only. Use Concorde's executable for access to solve options.

Author:

John LaRusic

Definition in file tsp.c.

7.21.2 Function Documentation

7.21.2.1 int main (int *argc*, char * *argv*[ ])

Definition at line 46 of file tsp.c.

References arrow_print_error, arrow_problem_destruct(), arrow_problem_read(), arrow_tsp_exact_solve(), arrow_tsp_result_destruct(), arrow_tsp_result_init(), arrow_tsp_result::found_tour, input_file, arrow_tsp_result::obj_value, program_name, read_args(), and arrow_tsp_result::total_time.

7.21.2.2 void print_help ()

Prints help/usage message.

7.21.2.3 void print_usage ()

Prints usage message.

7.21.2.4 void print_version ()

Prints version message.

7.21.2.5 void read_args (int *argc*, char ∗ *argv*[ ])

Reads program arguments.

7.21.3 Variable Documentation

7.21.3.1 char∗ input_file

Given input TSPLIB file

Definition at line 40 of file tsp.c.

7.21.3.2 char∗ program_name

Program name

Definition at line 39 of file tsp.c.

7.22 lib/tsp.c File Reference

TSP solver and Lin-Kernighan heuristic.

```
#include "arrow.h"
```

Functions

- int build_initial_tour (arrow_problem *problem, CCedgegengroup *plan, CCrandstate *rstate, int *initial_tour)
- int arrow_tsp_result_init (arrow_problem *problem, arrow_tsp_result *result)

  Initializes the TSP result structure.

- void arrow_tsp_result_destruct (arrow_tsp_result *result)

  Destructs a TSP result structure.

- void arrow_tsp_lk_params_init (arrow_problem *problem, arrow_tsp_lk_params *params)

  Sets default parameters for Lin-Kernighan heuristic:
  - *random_restarts = 0*
  - *stall_count = problem->size*
  - *kicks = (problem->size / 2), at least 500*
  - *kick_type = CC_LK_GEOMETRIC_KICK*
  - *time_bound = 0.0*
  - *length_bound = 0.0*
  - *initial_tour = NULL.*

- void arrow_tsp_lk_params_destruct (arrow_tsp_lk_params *params)

  Destructs a LK parameters structure.

- int arrow_tsp_exact_solve (arrow_problem *problem, int *initial_tour, arrow_tsp_result *result)

  Solves TSP with Concorde's exact solver.

- int arrow_tsp_lk_solve (arrow_problem *problem, arrow_tsp_lk_params *params, arrow_tsp_result *result)

  Solves TSP with Concorde's Lin-Kernighan heuristic.

7.22.1 Detailed Description

TSP solver and Lin-Kernighan heuristic.

Wrapper for calling Concorde's TSP solver and Lin-Kernighan heuristic.

Author:

John LaRusic

Definition in file tsp.c.

7.22.2 Function Documentation

7.22.2.1 int arrow_tsp_exact_solve (arrow_problem * *problem*, int * *initial_tour*, arrow_tsp_result * *result*)

Solves TSP with Concorde's exact solver.

Parameters:

problem [in] problem to solve

initial_tour [in] an initial tour (can be NULL)

result [out] TSP solution

Definition at line 66 of file tsp.c.

References ARROW_FAILURE, ARROW_SUCCESS, arrow_util_zeit(), arrow_problem::data, arrow_tsp_result::found_tour, arrow_problem::name, arrow_tsp_result::obj_value, arrow_problem::size, arrow_tsp_result::total_time, and arrow_tsp_result::tour.

Referenced by feasible(), and main().

7.22.2.2 void arrow_tsp_lk_params_destruct (arrow_tsp_lk_params * *params*)

Destructs a LK parameters structure.

Parameters:

params [out] LK parameters structure

Definition at line 59 of file tsp.c.

References arrow_tsp_lk_params::initial_tour.

Referenced by arrow_btsp_solve_plan_destruct(), arrow_tsp_lk_solve(), and main().

7.22.2.3 void arrow_tsp_lk_params_init (arrow_problem * *problem*, arrow_tsp_lk_params * *params*)

Sets default parameters for Lin-Kernighan heuristic:

- random_restarts = 0
- stall_count = problem->size
- kicks = (problem->size / 2), at least 500
- kick_type = CC_LK_GEOMETRIC_KICK
- time_bound = 0.0
- length_bound = 0.0
- initial_tour = NULL.

Parameters:

problem [in] problem to solve

params [out] LK parameters structure

Definition at line 47 of file tsp.c.

References arrow_tsp_lk_params::initial_tour, arrow_tsp_lk_params::kick_type, arrow_tsp_lk_params::kicks, arrow_tsp_lk_params::length_bound, arrow_tsp_lk_params::random_restarts, arrow_problem::size, arrow_tsp_lk_params::stall_count, and arrow_tsp_lk_params::time_bound.

Referenced by arrow_tsp_lk_solve(), and main().

7.22.2.4 int arrow_tsp_lk_solve (arrow_problem * *problem*, arrow_tsp_lk_params * *params*, arrow_tsp_result * *result*)

Solves TSP with Concorde's Lin-Kernighan heuristic.

Parameters:

problem [in] problem to solve

params [in] Lin-Kernighan params (can be NULL)

result [out] TSP solution

Definition at line 102 of file tsp.c.

References arrow_debug, ARROW_FAILURE, arrow_print_error, ARROW_SUCCESS, ARROW_TRUE, arrow_tsp_lk_params_destruct(), arrow_tsp_lk_params_init(), arrow_util_zeit(), build_initial_tour(), CONCORDE_FAILURE, arrow_problem::data, arrow_tsp_result::found_tour, arrow_tsp_lk_params::initial_tour, arrow_tsp_lk_params::kick_type, arrow_tsp_lk_params::kicks, arrow_tsp_lk_params::length_bound, arrow_tsp_result::obj_value, arrow_tsp_lk_params::random_restarts, arrow_problem::size, arrow_tsp_lk_params::stall_count, arrow_tsp_lk_params::time_bound, arrow_tsp_result::total_time, and arrow_tsp_result::tour.

Referenced by feasible(), and main().

7.22.2.5 void arrow_tsp_result_destruct (arrow_tsp_result * *result*)

Destructs a TSP result structure.

Parameters:

result [out] TSP result structure

Definition at line 37 of file tsp.c.

References arrow_tsp_result::tour.

Referenced by feasible(), and main().

7.22.2.6 int arrow_tsp_result_init (arrow_problem * *problem*, arrow_tsp_result * *result*)

Initializes the TSP result structure.

Parameters:

problem [in] problem to solve

result [out] TSP result structure

Definition at line 23 of file tsp.c.

References ARROW_ERROR_FATAL, ARROW_FALSE, ARROW_SUCCESS, arrow_util_create_int_-array(), arrow_tsp_result::found_tour, arrow_tsp_result::obj_value, arrow_problem::size, arrow_tsp_-result::total_time, and arrow_tsp_result::tour.

Referenced by feasible(), and main().

7.22.2.7 int build_initial_tour (arrow_problem * *problem*, CCedgegengroup * *plan*, CCrandstate * *rstate*, int * *initial_tour*)

Definition at line 273 of file tsp.c.

References ARROW_FAILURE, ARROW_FALSE, arrow_print_error, ARROW_SUCCESS, CONCORDE_FAILURE, arrow_problem::data, and arrow_problem::size.

Referenced by arrow_tsp_lk_solve().

7.23 lib/arrow.h File Reference

Header file for the Arrow callable library.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <limits.h>
#include <unistd.h>
#include <float.h>
#include <regex.h>
#include <getopt.h>
#include "concorde.h"
```

Data Structures

- struct arrow_bound_result
  A lower bound result.
- struct arrow_bintree
  Binary tree data structure.
- struct arrow_bintree_node
  Binary tree node.
- struct arrow_problem
  Problem data structure.
- struct arrow_problem_info
  Problem information data structure.
- struct arrow_tsp_lk_params
  LK algorithm parameters.
- struct arrow_tsp_result
  TSP result (including result from LK heuristic).
- struct arrow_btsp_result
  BTSP result.
- struct arrow_btsp_fun
  BTSP Cost matrix function definition.
- struct arrow_btsp_solve_plan
  BTSP feasibility solve step plan.

- struct arrow_btsp_params

  BTSP algorithm parameters.

- struct arrow_option

  Program options structure.

Defines

- #define ARROW_DEBUG
- #define arrow_debug printf
- #define ARROW_VERSION "1.0"
- #define ARROW_DEV_NULL "/dev/null"
- #define ARROW_SUCCESS 1
- #define ARROW_FAILURE 0
- #define ARROW_ERROR_INPUT 0
- #define ARROW_ERROR_FATAL -1
- #define ARROW_ERROR_NON_FATAL -2
- #define ARROW_TRUE 1
- #define ARROW_FALSE 0
- #define ARROW_BTSP_SOLVE_PLAN_BASIC 1
- #define ARROW_BTSP_SOLVE_PLAN_CONSTRAINED 2
- #define ARROW_BTSP_SOLVE_PLAN_CONSTRAINED_SHAKE 3
- #define ARROW_DEFAULT_BASIC_ATTEMPTS 3
- #define ARROW_OPTION_INT 1
- #define ARROW_OPTION_DOUBLE 2
- #define ARROW_OPTION_STRING 3
- #define CONCORDE_SUCCESS 0
- #define CONCORDE_FAILURE 1
- #define arrow_print_error(message) arrow_util_print_error(__FILE__, __LINE__, message)

Functions

- int arrow_2mb_solve (arrow_problem *problem, arrow_bound_result *result)

  Solves the 2-max bound (2MB) on the given problem.

- int arrow_dcbpb_solve (arrow_problem *problem, arrow_bound_result *result)

  Solves the degree constrained bottleneck paths bound (DCBPB).

- int arrow_bbssp_solve (arrow_problem *problem, arrow_problem_info *info, arrow_bound_result *result)

  Solves the bottleneck biconnected spanning subgraph problem (BBSSP) on the given problem.

- int arrow_bbssp_biconnected (arrow_problem *problem, int max_cost, int *result)

  Determines if the graph is biconnected using only edges with costs less than or equal to the given value.

- void arrow_bintree_init (arrow_bintree *tree)

  Initializes the binary tree data structure.

- void arrow_bintree_destruct (arrow_bintree *tree)

  Destructs a binary tree data structure.

- int arrow_bintree_insert (arrow_bintree *tree, int value)

  Inserts a value into the binary tree.

- int arrow_bintree_to_array (arrow_bintree *tree, int **array)

  Initializes the binary tree data structure.

- void arrow_bintree_print (arrow_bintree *tree)

  Prints out the values of the binary tree.

- int arrow_bscssp_solve (arrow_problem *problem, arrow_problem_info *info, arrow_bound_result *result)

  Solves the bottleneck strongly connected spanning subgraph problem (BSCSSP) on the given graph.

- int arrow_btsp_result_init (arrow_problem *problem, arrow_btsp_result *result)

  Initializes the BTSP result structure.

- void arrow_btsp_result_destruct (arrow_btsp_result *result)

  Destructs a BTSP result structure.

- int arrow_btsp_solve (arrow_problem *problem, arrow_problem_info *info, arrow_btsp_params *params, arrow_btsp_result *result)

  Solves TSP with Concorde's exact solver.

- void arrow_btsp_params_init (arrow_btsp_params *params)

  Inititalizes BTSP parameter structure.

- void arrow_btsp_params_destruct (arrow_btsp_params *params)

  Destructs a BTSP parameters structure.

- void arrow_btsp_solve_plan_init (arrow_btsp_solve_plan *plan)

  Inititalizes BTSP solve plan structure.

- void arrow_btsp_solve_plan_destruct (arrow_btsp_solve_plan *plan)

  Destructs a BTSP solve plan structure.

- int arrow_btsp_fun_apply (arrow_btsp_fun *fun, arrow_problem *old_problem, int delta, arrow_problem *new_problem)

  Applies the given function to the given problem to create a new problem.

- void arrow_btsp_fun_destruct (arrow_btsp_fun *fun)

  Destructs a function structure.

- int arrow_btsp_fun_basic (int shallow, arrow_btsp_fun *fun)

  Basic BTSP to TSP function.

- int arrow_btsp_fun_basic_atsp (int shallow, arrow_btsp_fun *fun)

  Basic BTSP to TSP function for asymmetric problem instances.

- int arrow_btsp_fun_constrained (int shallow, double feasible_length, int infinity, arrow_btsp_fun *fun)

  Constrained BTSP to TSP function.

- int arrow_btsp_fun_constrained_shake (int shallow, double feasible_length, int infinity, int rand_min, int rand_max, arrow_problem *problem, arrow_problem_info *info, arrow_btsp_fun *fun)

  Constrained "Shake" BTSP to TSP function.

- int arrow_options_parse (int num_opts, arrow_option options[ ], char *description, char *usage, int argc, char *argv[ ], int *opt_ind)
- int arrow_problem_read (char *file_name, arrow_problem *problem)

  Reads a problem from a TSPLIB file.

- void arrow_problem_destruct (arrow_problem *problem)

  Deallocates problem data structure.

- int arrow_problem_info_get (arrow_problem *problem, arrow_problem_info *info)

  Builds ordered cost list and finds min/max cost in a problem.

- void arrow_problem_info_destruct (arrow_problem_info *info)

  Deallocates problem info data structure.

- void arrow_problem_print (arrow_problem *problem)

  Prints out information about a problem.

- int arrow_problem_get_cost (arrow_problem *problem, int i, int j)

  Retrieves cost between nodes i and j.

- int arrow_problem_read_tour (char *file_name, int size, int *tour)

  Reads a TSPLIB tour file.

- int arrow_problem_abtsp_to_sbtsp (arrow_problem *old_problem, int infinity, arrow_problem *new_problem)

  Transforms an asymmetric BTSP problem of n nodes into a symmetric BTSP problem with 2n nodes.

- int arrow_tsp_result_init (arrow_problem *problem, arrow_tsp_result *result)

  Initializes the TSP result structure.

- void arrow_tsp_result_destruct (arrow_tsp_result *result)

  Destructs a TSP result structure.

- void arrow_tsp_lk_params_init (arrow_problem *problem, arrow_tsp_lk_params *params)

  Sets default parameters for Lin-Kernighan heuristic:
  - *random_restarts = 0*
  - *stall_count = problem->size*
  - *kicks = (problem->size / 2), at least 500*
  - *kick_type = CC_LK_GEOMETRIC_KICK*
  - *time_bound = 0.0*
  - *length_bound = 0.0*

- *initial_tour = NULL.*

- void arrow_tsp_lk_params_destruct (arrow_tsp_lk_params *params)

  Destructs a LK parameters structure.

- int arrow_tsp_exact_solve (arrow_problem *problem, int *initial_tour, arrow_tsp_result *result)

  Solves TSP with Concorde's exact solver.

- int arrow_tsp_lk_solve (arrow_problem *problem, arrow_tsp_lk_params *params, arrow_tsp_result *result)

  Solves TSP with Concorde's Lin-Kernighan heuristic.

- int arrow_util_create_int_array (int size, int **array)

  Creates an integer array.

- int arrow_util_create_int_matrix (int rows, int cols, int ***matrix, int **space)

  Creates a full integer matrix.

- void arrow_util_print_error (const char *file_name, int line_num, const char *message)

  Prints an error message to stderr with consistent formatting.

- double arrow_util_zeit ()

  Used to measure timings.

- void arrow_util_redirect_stdout_to_file (const char *filename, int *old_stream)

  Redirects STDOUT stream to a file (can be used to completely surpress output by directing to /dev/null).

- void arrow_util_restore_stdout (int old_stream)

  Restores STDOUT stream that's been redirected.

- void arrow_util_CCdatagroup_shallow_copy (CCdatagroup *from, CCdatagroup *to)

  Makes a shallow copy of the Concorde CCdatagroup structure.

- int arrow_util_CCdatagroup_init_matrix (int size, CCdatagroup *dat)

  Initializes an upper-diagonal matrix norm structure for Concorde that is ready to be filled in with values.

- int arrow_util_binary_search (int *array, int size, int element, int *pos)

  Performs a binary search to find the wanted element in a sorted integer array.

- int arrow_util_regex_match (char *string, char *pattern)

  Determines if the given string turns up a match for the given regular expression pattern.

- void arrow_util_print_program_args (int argc, char *argv[ ], FILE *out)

  Prints out the given program arguments to the specified file.

- void arrow_util_random_seed (int seed)

  Seeds the random number generator. Pass a value of 0 to seed with the current time.

- int arrow_util_random ()

  Returns a random number between 0 and RAND_MAX (normally, RAND_MAX = INT_MAX).

- int arrow_util_random_between (int min, int max)

  Returns a random number between min and max.

- void arrow_util_write_tour (arrow_problem *problem, char *comment, int *tour, FILE *out)
- void arrow_util_sbtsp_to_abstp_tour (arrow_problem *problem, int *old_tour, int *new_tour)

7.23.1 Detailed Description

Header file for the Arrow callable library.

Function prototypes and structures exposed by the callable library.

Author:

John LaRusic

Definition in file arrow.h.

7.23.2 Define Documentation

7.23.2.1 #define ARROW_BTSP_SOLVE_PLAN_BASIC 1

Definition at line 53 of file arrow.h.

Referenced by main().

7.23.2.2 #define ARROW_BTSP_SOLVE_PLAN_CONSTRAINED 2

Definition at line 54 of file arrow.h.

Referenced by main().

7.23.2.3 #define ARROW_BTSP_SOLVE_PLAN_CONSTRAINED_SHAKE 3

Definition at line 55 of file arrow.h.

Referenced by main().

7.23.2.4 #define arrow_debug printf

Definition at line 33 of file arrow.h.

Referenced by arrow_btsp_solve(), arrow_problem_print(), arrow_problem_read(), arrow_tsp_lk_solve(), feasible(), main(), and read_atsp().

7.23.2.5 #define ARROW_DEBUG

Definition at line 31 of file arrow.h.

7.23.2.6 #define ARROW_DEFAULT_BASIC_ATTEMPTS 3

Definition at line 57 of file arrow.h.

7.23.2.7 #define ARROW_DEV_NULL "/dev/null"

Definition at line 42 of file arrow.h.

Referenced by main().

7.23.2.8 #define ARROW_ERROR_FATAL -1

Definition at line 48 of file arrow.h.

Referenced by arrow_bbssp_biconnected(), arrow_bbssp_solve(), arrow_bscssp_solve(), arrow_-btsp_fun_apply(), arrow_btsp_fun_constrained(), arrow_problem_read_tour(), arrow_tsp_result_init(), construct_node(), insert_at(), read_atsp(), and strongly_connected().

7.23.2.9 #define ARROW_ERROR_INPUT 0

Definition at line 47 of file arrow.h.

7.23.2.10 #define ARROW_ERROR_NON_FATAL -2

Definition at line 49 of file arrow.h.

7.23.2.11 #define ARROW_FAILURE 0

Definition at line 45 of file arrow.h.

Referenced by arrow_bap_solve(), arrow_bbssp_solve(), arrow_bscssp_solve(), arrow_btsp_fun_apply(), arrow_btsp_fun_basic_atsp(), arrow_btsp_fun_constrained_shake(), arrow_btsp_result_init(), arrow_-btsp_solve(), arrow_dcbpb_solve(), arrow_options_parse(), arrow_problem_abtsp_to_sbtsp(), arrow_-problem_read(), arrow_tsp_exact_solve(), arrow_tsp_lk_solve(), arrow_util_binary_search(), arrow_-util_CCdatagroup_init_matrix(), arrow_util_create_int_array(), arrow_util_create_int_matrix(), basic_-shallow_apply(), build_initial_tour(), constrained_shake_deep_apply(), constrained_shallow_apply(), fea-sible(), and main().

7.23.2.12 #define ARROW_FALSE 0

Definition at line 51 of file arrow.h.

Referenced by arrow_bbssp_biconnected(), arrow_btsp_fun_basic(), arrow_btsp_fun_basic_atsp(), arrow_btsp_fun_constrained(), arrow_btsp_fun_constrained_shake(), arrow_btsp_params_init(), arrow_-btsp_result_init(), arrow_btsp_solve(), arrow_options_parse(), arrow_problem_abtsp_to_sbtsp(), arrow_problem_read(), arrow_tsp_result_init(), arrow_util_regex_match(), basic_atsp_feasible(), basic_feasible(), build_initial_tour(), constrained_shake_feasible(), construct_node(), feasible(), ford_-fulkerson_labeling(), insert_at(), main(), shortest_augmenting_path(), and strongly_connected().

7.23.2.13 #define ARROW_OPTION_DOUBLE 2

Definition at line 60 of file arrow.h.

Referenced by arrow_options_parse().

7.23.2.14 #define ARROW_OPTION_INT 1

Definition at line 59 of file arrow.h.

Referenced by arrow_options_parse().

7.23.2.15 #define ARROW_OPTION_STRING 3

Definition at line 61 of file arrow.h.

Referenced by arrow_options_parse().

7.23.2.16 #define arrow_print_error(message) arrow_util_print_error(__FILE__, __LINE__, message)

Definition at line 70 of file arrow.h.

Referenced by arrow_bbssp_solve(), arrow_btsp_fun_apply(), arrow_btsp_fun_basic_atsp(), arrow_btsp_fun_constrained(), arrow_btsp_fun_constrained_shake(), arrow_btsp_solve(), arrow_options_parse(), arrow_problem_abtsp_to_sbtsp(), arrow_problem_read(), arrow_tsp_lk_solve(), arrow_util_CCdatagroup_init_matrix(), arrow_util_create_int_array(), arrow_util_create_int_matrix(), basic_shallow_apply(), build_initial_tour(), constrained_shake_deep_apply(), constrained_shallow_apply(), construct_node(), main(), read_args(), and read_atsp().

7.23.2.17 #define ARROW_SUCCESS 1

Definition at line 44 of file arrow.h.

Referenced by arrow_2mb_solve(), arrow_bap_solve(), arrow_bbssp_solve(), arrow_bintree_to_array(), arrow_bscssp_solve(), arrow_btsp_fun_apply(), arrow_btsp_fun_basic(), arrow_btsp_fun_basic_atsp(), arrow_btsp_fun_constrained(), arrow_btsp_fun_constrained_shake(), arrow_btsp_result_init(), arrow_btsp_solve(), arrow_dcbpb_solve(), arrow_options_parse(), arrow_problem_abtsp_to_sbtsp(), arrow_problem_info_get(), arrow_problem_read(), arrow_problem_read_tour(), arrow_tsp_exact_solve(), arrow_tsp_lk_solve(), arrow_tsp_result_init(), arrow_util_binary_search(), arrow_util_CCdatagroup_init_matrix(), arrow_util_create_int_array(), arrow_util_create_int_matrix(), basic_atsp_deep_apply(), basic_deep_apply(), basic_shallow_apply(), build_initial_tour(), constrained_deep_apply(), constrained_shake_deep_apply(), constrained_shallow_apply(), construct_node(), feasible(), find_art_points(), insert_at(), main(), read_atsp(), shortest_augmenting_path(), and strongly_connected().

7.23.2.18 #define ARROW_TRUE 1

Definition at line 50 of file arrow.h.

Referenced by arrow_bbssp_biconnected(), arrow_bbssp_solve(), arrow_bscssp_solve(), arrow_btsp_fun_basic(), arrow_btsp_fun_constrained(), arrow_btsp_solve(), arrow_options_parse(), arrow_problem_abtsp_to_sbtsp(), arrow_problem_info_get(), arrow_problem_read(), arrow_tsp_lk_solve(), arrow_util_regex_match(), basic_atsp_feasible(), basic_feasible(), constrained_shake_feasible(), destruct_node(), fea-

sible(), ford_fulkerson_labeling(), insert_at(), main(), shortest_augmenting_path(), strongly_connected(), and strongly_connected_dfs().

7.23.2.19 #define ARROW_VERSION "1.0"

Definition at line 41 of file arrow.h.

Referenced by print_version().

7.23.2.20 #define CONCORDE_FAILURE 1

Definition at line 64 of file arrow.h.

Referenced by arrow_tsp_lk_solve(), and build_initial_tour().

7.23.2.21 #define CONCORDE_SUCCESS 0

Definition at line 63 of file arrow.h.

Referenced by arrow_problem_read_tour().

7.23.3 Function Documentation

7.23.3.1 int arrow_2mb_solve (arrow_problem * *problem*, arrow_bound_result * *result*)

Solves the 2-max bound (2MB) on the given problem.

Parameters:

problem [in] problem data
result [out] 2MB solution

Definition at line 16 of file 2mb.c.

References ARROW_SUCCESS, arrow_util_zeit(), arrow_problem::get_cost, max(), arrow_bound_result::obj_value, arrow_problem::size, arrow_problem::symmetric, and arrow_bound_result::total_time.

Referenced by main().

7.23.3.2 int arrow_bbssp_biconnected (arrow_problem * *problem*, int *max_cost*, int * *result*)

Determines if the graph is biconnected using only edges with costs less than or equal to the given value.

Parameters:

problem [in] problem data
max_cost [in] value to check biconnectivity question against
result [out] ARROW_TRUE if biconnected, ARROW_FALSE otherwise.

Definition at line 94 of file bbssp.c.

References ARROW_ERROR_FATAL, ARROW_FALSE, ARROW_TRUE, arrow_util_create_int_array(), find_art_points(), and arrow_problem::size.

Referenced by arrow_bbssp_solve().

7.23.3.3 int arrow_bbssp_solve (arrow_problem * *problem*, arrow_problem_info * *info*, arrow_bound_result * *result*)

Solves the bottleneck biconnected spanning subgraph problem (BBSSP) on the given problem.

Parameters:

problem [in] problem data

info [in] problem info

result [out] BBSSP solution

Definition at line 44 of file bbssp.c.

References arrow_bbssp_biconnected(), ARROW_ERROR_FATAL, ARROW_FAILURE, arrow_print_error, ARROW_SUCCESS, ARROW_TRUE, arrow_util_zeit(), arrow_problem_info::cost_list, arrow_problem_info::cost_list_length, arrow_bound_result::obj_value, arrow_problem::symmetric, and arrow_bound_result::total_time.

Referenced by main().

7.23.3.4 void arrow_bintree_destruct (arrow_bintree * *tree*)

Destructs a binary tree data structure.

Parameters:

tree [out] binary tree structure

Definition at line 60 of file bintree.c.

References destruct_node(), arrow_bintree::root_node, and arrow_bintree::size.

Referenced by arrow_problem_info_get(), and constrained_shake_deep_apply().

7.23.3.5 void arrow_bintree_init (arrow_bintree * *tree*)

Initializes the binary tree data structure.

Parameters:

tree [out] binary tree structure

Definition at line 53 of file bintree.c.

References arrow_bintree::root_node, and arrow_bintree::size.

Referenced by arrow_problem_info_get(), and constrained_shake_deep_apply().

7.23.3.6 int arrow_bintree_insert (arrow_bintree * *tree*, int *value*)

Inserts a value into the binary tree.

Parameters:

tree [out] binary tree structure

value [in] value to insert into tree

Definition at line 67 of file bintree.c.

References construct_node(), insert_at(), arrow_bintree::root_node, and arrow_bintree::size.

Referenced by arrow_problem_info_get(), and constrained_shake_deep_apply().

7.23.3.7 void arrow_bintree_print (arrow_bintree * *tree*)

Prints out the values of the binary tree.

Parameters:

tree [in] binary tree structure

7.23.3.8 int arrow_bintree_to_array (arrow_bintree * *tree*, int ** *array*)

Initializes the binary tree data structure.

Parameters:

tree [out] binary tree structure

array [out] array to be created and filled

Definition at line 87 of file bintree.c.

References ARROW_SUCCESS, arrow_util_create_int_array(), fill_array(), arrow_bintree::root_node, and arrow_bintree::size.

Referenced by arrow_problem_info_get(), and constrained_shake_deep_apply().

7.23.3.9 int arrow_bscssp_solve (arrow_problem * *problem*, arrow_problem_info * *info*, arrow_bound_result * *result*)

Solves the bottleneck strongly connected spanning subgraph problem (BSCSSP) on the given graph.

Parameters:

problem [in] problem data

info [in] problem info

result [out] BSCSSP solution

Definition at line 43 of file bscssp.c.

References ARROW_ERROR_FATAL, ARROW_FAILURE, ARROW_SUCCESS, ARROW_TRUE, arrow_util_zeit(), arrow_problem_info::cost_list, arrow_problem_info::cost_list_length, arrow_bound_result::obj_value, strongly_connected(), and arrow_bound_result::total_time.

Referenced by main().

7.23.3.10 int arrow_btsp_fun_apply (arrow_btsp_fun * *fun*, arrow_problem * *old_problem*, int *delta*, arrow_problem * *new_problem*)

Applies the given function to the given problem to create a new problem.

Parameters:

fun [in] function structure

old_problem [in] existing problem

delta [in] delta parameter

new_problem [out] new problem to create

Definition at line 213 of file btsp_fun.c.

References arrow_btsp_fun::apply, ARROW_ERROR_FATAL, ARROW_FAILURE, arrow_print_error, ARROW_SUCCESS, arrow_util_CCdatagroup_init_matrix(), arrow_problem::data, arrow_problem::get_cost, arrow_problem::name, arrow_btsp_fun::shallow, arrow_problem::shallow, and arrow_problem::size.

Referenced by feasible().

7.23.3.11 int arrow_btsp_fun_basic (int *shallow*, arrow_btsp_fun * *fun*)

Basic BTSP to TSP function.

Parameters:

shallow [in] ARROW_TRUE for shallow copy, ARROW_FALSE for deep

fun [out] function structure

Definition at line 249 of file btsp_fun.c.

References arrow_btsp_fun::apply, ARROW_FALSE, ARROW_SUCCESS, ARROW_TRUE, basic_deep_apply(), basic_destruct(), basic_feasible(), basic_shallow_apply(), arrow_btsp_fun::data, arrow_btsp_fun::destruct, arrow_btsp_fun::feasible, arrow_btsp_fun::feasible_length, and arrow_btsp_fun::shallow.

Referenced by main().

7.23.3.12 int arrow_btsp_fun_basic_atsp (int *shallow*, arrow_btsp_fun * *fun*)

Basic BTSP to TSP function for asymmetric problem instances.

Parameters:

shallow [in] ARROW_TRUE for shallow copy, ARROW_FALSE for deep

fun [out] function structure

Definition at line 270 of file btsp_fun.c.

References arrow_btsp_fun::apply, ARROW_FAILURE, ARROW_FALSE, arrow_print_error, ARROW_SUCCESS, basic_atsp_deep_apply(), basic_atsp_destruct(), basic_atsp_feasible(), arrow_btsp_fun::destruct, arrow_btsp_fun::feasible, arrow_btsp_fun::feasible_length, and arrow_btsp_fun::shallow.

Referenced by main().

7.23.3.13 int arrow_btsp_fun_constrained (int *shallow*, double *feasible_length*, int *infinity*, arrow_btsp_fun * *fun*)

Constrained BTSP to TSP function.

Parameters:

shallow [in] ARROW_TRUE for shallow copy, ARROW_FALSE for deep

feasible_length [in] length of feasible tour

infinity [in] value to use as "infinity"

fun [out] function structure

Definition at line 291 of file btsp_fun.c.

References arrow_btsp_fun::apply, ARROW_ERROR_FATAL, ARROW_FALSE, arrow_print_error, ARROW_SUCCESS, ARROW_TRUE, basic_feasible(), constrained_deep_apply(), constrained_destruct(), constrained_shallow_apply(), arrow_btsp_fun::data, arrow_btsp_fun::destruct, arrow_btsp_fun::feasible, arrow_btsp_fun::feasible_length, and arrow_btsp_fun::shallow.

Referenced by main().

7.23.3.14 int arrow_btsp_fun_constrained_shake (int *shallow*, double *feasible_length*, int *infinity*, int *rand_min*, int *rand_max*, arrow_problem * *problem*, arrow_problem_info * *info*, arrow_btsp_fun * *fun*)

Constrained "Shake" BTSP to TSP function.

Parameters:

shallow [in] ARROW_TRUE for shallow copy, ARROW_FALSE for deep

feasible_length [in] length of feasible tour

infinity [in] value to use as "infinity"

rand_min [in] minimum random value to generate

rand_max [in] maximum random value to generate

problem [in] the problem the shake is based upon

info [in] information about the original problem

fun [out] function structure

Definition at line 319 of file btsp_fun.c.

References arrow_btsp_fun::apply, ARROW_FAILURE, ARROW_FALSE, arrow_print_error, ARROW_SUCCESS, constrained_shake_deep_apply(), constrained_shake_destruct(), constrained_shake_feasible(), arrow_btsp_fun::data, arrow_btsp_fun::destruct, arrow_btsp_fun::feasible, arrow_btsp_fun::feasible_length, constrained_shake_data::infinity, constrained_shake_data::info, constrained_shake_data::problem, constrained_shake_data::rand_max, constrained_shake_data::rand_min, and arrow_btsp_fun::shallow.

Referenced by main().

7.23.3.15 void arrow_btsp_fun_destruct (arrow_btsp_fun * *fun*)

Destructs a function structure.

Parameters:

fun [out] function structure

Definition at line 243 of file btsp_fun.c.

References arrow_btsp_fun::destruct.

Referenced by arrow_btsp_solve_plan_destruct(), and main().

7.23.3.16 void arrow_btsp_params_destruct (arrow_btsp_params * *params*)

Destructs a BTSP parameters structure.

Parameters:

params [out] BTSP parameters structure

Definition at line 71 of file btsp.c.

References arrow_btsp_solve_plan_destruct(), arrow_btsp_params::num_steps, and arrow_btsp_-params::steps.

Referenced by main().

7.23.3.17 void arrow_btsp_params_init (arrow_btsp_params * *params*)

Inititalizes BTSP parameter structure.

Parameters:

params [out] BTSP parameters structure

Definition at line 60 of file btsp.c.

References ARROW_FALSE, arrow_btsp_params::confirm_sol, arrow_btsp_params::find_short_tour, arrow_btsp_params::lower_bound, arrow_btsp_params::num_steps, arrow_btsp_params::supress_ebst, and arrow_btsp_params::upper_bound.

Referenced by main().

7.23.3.18 void arrow_btsp_result_destruct (arrow_btsp_result * *result*)

Destructs a BTSP result structure.

Parameters:

result [out] BTSP result structure

Definition at line 53 of file btsp.c.

References arrow_btsp_result::tour.

Referenced by arrow_btsp_solve(), and main().

7.23.3.19 int arrow_btsp_result_init (arrow_problem * *problem*, arrow_btsp_result * *result*)

Initializes the BTSP result structure.

Parameters:

problem [in] problem to solve

result [out] BTSP result structure

Definition at line 33 of file btsp.c.

References ARROW_FAILURE, ARROW_FALSE, ARROW_SUCCESS, arrow_util_create_int_array(), arrow_btsp_result::bin_search_steps, arrow_btsp_result::exact_attempts, arrow_btsp_result::exact_time, arrow_btsp_result::found_tour, arrow_btsp_result::linkern_attempts, arrow_btsp_result::linkern_time, arrow_btsp_result::obj_value, arrow_problem::size, arrow_btsp_result::total_time, arrow_btsp_result::tour, and arrow_btsp_result::tour_length.

Referenced by arrow_btsp_solve(), and main().

7.23.3.20 int arrow_btsp_solve (arrow_problem * *problem*, arrow_problem_info * *info*, arrow_btsp_params * *params*, arrow_btsp_result * *result*)

Solves TSP with Concorde's exact solver.

Parameters:

problem [in] problem to solve

info [in] extra problem info

params [in] parameters for solver (can be NULL)

result [out] BTSP solution

Definition at line 98 of file btsp.c.

References arrow_btsp_result_destruct(), arrow_btsp_result_init(), arrow_debug, ARROW_FAILURE, ARROW_FALSE, arrow_print_error, ARROW_SUCCESS, ARROW_TRUE, arrow_util_binary_search(), arrow_util_zeit(), arrow_btsp_result::bin_search_steps, arrow_btsp_params::confirm_sol, arrow_problem_info::cost_list, arrow_problem_info::cost_list_length, arrow_btsp_result::exact_attempts, arrow_btsp_result::exact_time, feasible(), arrow_btsp_params::find_short_tour, arrow_btsp_result::found_tour, arrow_btsp_result::linkern_attempts, arrow_btsp_result::linkern_time, arrow_btsp_params::lower_bound, arrow_btsp_params::num_steps, arrow_btsp_result::obj_value, arrow_btsp_result::optimal, arrow_problem::size, arrow_btsp_params::steps, arrow_btsp_params::supress_ebst, arrow_problem::symmetric, arrow_btsp_result::total_time, arrow_btsp_result::tour, arrow_btsp_result::tour_length, upper_bound, and arrow_btsp_params::upper_bound.

Referenced by main().

7.23.3.21 void arrow_btsp_solve_plan_destruct (arrow_btsp_solve_plan * *plan*)

Destructs a BTSP solve plan structure.

Parameters:

plan [out] BTSP solve plan structure

Definition at line 91 of file btsp.c.

References arrow_btsp_fun_destruct(), arrow_tsp_lk_params_destruct(), arrow_btsp_solve_plan::fun, and arrow_btsp_solve_plan::lk_params.

Referenced by arrow_btsp_params_destruct().

7.23.3.22 void arrow_btsp_solve_plan_init (arrow_btsp_solve_plan * *plan*)

Inititalizes BTSP solve plan structure.

Parameters:

plan [out] BTSP solve plan structure

Definition at line 84 of file btsp.c.

References arrow_btsp_solve_plan::attempts, and arrow_btsp_solve_plan::plan_type.

7.23.3.23 int arrow_dcbpb_solve (arrow_problem * *problem*, arrow_bound_result * *result*)

Solves the degree constrained bottleneck paths bound (DCBPB).

Parameters:

problem [in] problem data

result [out] BPB solution

Definition at line 38 of file dcbpb.c.

References ARROW_FAILURE, ARROW_SUCCESS, arrow_util_create_int_matrix(), arrow_util_zeit(), bottleneck_paths(), arrow_problem::get_cost, max(), arrow_bound_result::obj_value, arrow_problem::size, and arrow_bound_result::total_time.

Referenced by main().

7.23.3.24 int arrow_options_parse (int *num_opts*, arrow_option *options*[ ], char * *description*, char * *usage*, int *argc*, char * *argv*[ ], int * *opt_ind*)

Definition at line 24 of file options.c.

References ARROW_FAILURE, ARROW_FALSE, ARROW_OPTION_DOUBLE, ARROW_OPTION_INT, ARROW_OPTION_STRING, arrow_print_error, ARROW_SUCCESS, ARROW_TRUE, arrow_option::long_option, print_help(), print_usage(), print_version(), and arrow_option::short_option.

Referenced by main().

7.23.3.25 int arrow_problem_abtsp_to_sbtsp (arrow_problem * *old_problem*, int *infinity*, arrow_problem * *new_problem*)

Transforms an asymmetric BTSP problem of n nodes into a symmetric BTSP problem with 2n nodes.

Parameters:

old_problem [in] the asymmetric problem

infinity [in] value to use as "infinity"
new_problem [out] the new symmetric problem

Definition at line 228 of file problem.c.

References ARROW_FAILURE, ARROW_FALSE, arrow_print_error, arrow_problem_get_cost(), ARROW_SUCCESS, ARROW_TRUE, arrow_util_CCdatagroup_init_matrix(), arrow_problem::data, arrow_problem::get_cost, arrow_problem::name, arrow_problem::shallow, arrow_problem::size, and arrow_problem::symmetric.

Referenced by main().

7.23.3.26 void arrow_problem_destruct (arrow_problem * *problem*)

Deallocates problem data structure.

Parameters:

problem [out] problem data structure

Definition at line 98 of file problem.c.

References arrow_problem::data, and arrow_problem::shallow.

Referenced by feasible(), and main().

7.23.3.27 int arrow_problem_get_cost (arrow_problem * *problem*, int *i*, int *j*) [inline]

Retrieves cost between nodes i and j.

Parameters:

problem [in] pointer to arrow_problem structure
i [in] id of starting node
j [in] id of ending node

Returns:

cost between node i and node j

Definition at line 213 of file problem.c.

References arrow_problem::data.

Referenced by arrow_problem_abtsp_to_sbtsp(), and arrow_problem_read().

7.23.3.28 void arrow_problem_info_destruct (arrow_problem_info * *info*)

Deallocates problem info data structure.

Parameters:

info [in] problem info data structure

Definition at line 158 of file problem.c.

References arrow_problem_info::cost_list.

Referenced by main().

7.23.3.29 int arrow_problem_info_get (arrow_problem * *problem*, arrow_problem_info * *info*)

Builds ordered cost list and finds min/max cost in a problem.

Parameters:

problem [in] problem data structure

info [out] problem info data structure

Definition at line 115 of file problem.c.

References arrow_bintree_destruct(), arrow_bintree_init(), arrow_bintree_insert(), arrow_bintree_to_array(), ARROW_SUCCESS, ARROW_TRUE, arrow_problem_info::cost_list, arrow_problem_info::cost_list_length, arrow_problem::get_cost, arrow_problem_info::max_cost, arrow_problem_info::min_cost, arrow_bintree::size, arrow_problem::size, and arrow_problem::symmetric.

Referenced by main().

7.23.3.30 void arrow_problem_print (arrow_problem * *problem*)

Prints out information about a problem.

Parameters:

problem [in] problem data structure

Definition at line 165 of file problem.c.

References arrow_debug, arrow_problem::get_cost, and arrow_problem::size.

7.23.3.31 int arrow_problem_read (char * *file_name*, arrow_problem * *problem*)

Reads a problem from a TSPLIB file.

Parameters:

file_name [in] path to TSPLIB file

problem [out] problem data structure

Definition at line 54 of file problem.c.

References arrow_debug, ARROW_FAILURE, ARROW_FALSE, arrow_print_error, arrow_problem_get_cost(), ARROW_SUCCESS, ARROW_TRUE, arrow_problem::data, arrow_problem::get_cost, is_asymmetric(), is_symmetric(), arrow_problem::name, read_atsp(), arrow_problem::shallow, arrow_problem::size, and arrow_problem::symmetric.

Referenced by main().

7.23.3.32 int arrow_problem_read_tour (char * *file_name*, int *size*, int * *tour*)

Reads a TSPLIB tour file.

Parameters:

file_name [in] the TSPLIB tour file to read

size [in] the number of cities in the tour

tour [out] an array to hold the tour in node-node format

Definition at line 219 of file problem.c.

References ARROW_ERROR_FATAL, ARROW_SUCCESS, and CONCORDE_SUCCESS.

Referenced by main().

7.23.3.33 int arrow_tsp_exact_solve (arrow_problem * *problem*, int * *initial_tour*, arrow_tsp_result * *result*)

Solves TSP with Concorde's exact solver.

Parameters:

problem [in] problem to solve

initial_tour [in] an initial tour (can be NULL)

result [out] TSP solution

Definition at line 66 of file tsp.c.

References ARROW_FAILURE, ARROW_SUCCESS, arrow_util_zeit(), arrow_problem::data, arrow_tsp_result::found_tour, arrow_problem::name, arrow_tsp_result::obj_value, arrow_problem::size, arrow_tsp_result::total_time, and arrow_tsp_result::tour.

Referenced by feasible(), and main().

7.23.3.34 void arrow_tsp_lk_params_destruct (arrow_tsp_lk_params * *params*)

Destructs a LK parameters structure.

Parameters:

params [out] LK parameters structure

Definition at line 59 of file tsp.c.

References arrow_tsp_lk_params::initial_tour.

Referenced by arrow_btsp_solve_plan_destruct(), arrow_tsp_lk_solve(), and main().

7.23.3.35 void arrow_tsp_lk_params_init (arrow_problem * *problem*, arrow_tsp_lk_params * *params*)

Sets default parameters for Lin-Kernighan heuristic:

- random_restarts = 0
- stall_count = problem->size
- kicks = (problem->size / 2), at least 500
- kick_type = CC_LK_GEOMETRIC_KICK
- time_bound = 0.0

- length_bound = 0.0
- initial_tour = NULL.

Parameters:

problem [in] problem to solve

params [out] LK parameters structure

Definition at line 47 of file tsp.c.

References arrow_tsp_lk_params::initial_tour, arrow_tsp_lk_params::kick_type, arrow_tsp_lk_params::kicks, arrow_tsp_lk_params::length_bound, arrow_tsp_lk_params::random_restarts, arrow_problem::size, arrow_tsp_lk_params::stall_count, and arrow_tsp_lk_params::time_bound.

Referenced by arrow_tsp_lk_solve(), and main().

7.23.3.36 int arrow_tsp_lk_solve (arrow_problem * *problem*, arrow_tsp_lk_params * *params*, arrow_tsp_result * *result*)

Solves TSP with Concorde's Lin-Kernighan heuristic.

Parameters:

problem [in] problem to solve

params [in] Lin-Kernighan params (can be NULL)

result [out] TSP solution

Definition at line 102 of file tsp.c.

References arrow_debug, ARROW_FAILURE, arrow_print_error, ARROW_SUCCESS, ARROW_TRUE, arrow_tsp_lk_params_destruct(), arrow_tsp_lk_params_init(), arrow_util_zeit(), build_initial_tour(), CONCORDE_FAILURE, arrow_problem::data, arrow_tsp_result::found_tour, arrow_tsp_lk_params::initial_tour, arrow_tsp_lk_params::kick_type, arrow_tsp_lk_params::kicks, arrow_tsp_lk_params::length_bound, arrow_tsp_result::obj_value, arrow_tsp_lk_params::random_restarts, arrow_problem::size, arrow_tsp_lk_params::stall_count, arrow_tsp_lk_params::time_bound, arrow_tsp_result::total_time, and arrow_tsp_result::tour.

Referenced by feasible(), and main().

7.23.3.37 void arrow_tsp_result_destruct (arrow_tsp_result * *result*)

Destructs a TSP result structure.

Parameters:

result [out] TSP result structure

Definition at line 37 of file tsp.c.

References arrow_tsp_result::tour.

Referenced by feasible(), and main().

7.23.3.38 int arrow_tsp_result_init (arrow_problem * *problem*, arrow_tsp_result * *result*)

Initializes the TSP result structure.

Parameters:

problem [in] problem to solve

result [out] TSP result structure

Definition at line 23 of file tsp.c.

References ARROW_ERROR_FATAL, ARROW_FALSE, ARROW_SUCCESS, arrow_util_create_int_-array(), arrow_tsp_result::found_tour, arrow_tsp_result::obj_value, arrow_problem::size, arrow_tsp_-result::total_time, and arrow_tsp_result::tour.

Referenced by feasible(), and main().

7.23.3.39 int arrow_util_binary_search (int * *array*, int *size*, int *element*, int * *pos*)

Performs a binary search to find the wanted element in a sorted integer array.

Parameters:

array [in] the array to search (note: must be sorted in non-increasing order)

size [in] size of the array

element [in] the element to find in the array

pos [out] the index where the element can be found in the array

Definition at line 145 of file util.c.

References ARROW_FAILURE, and ARROW_SUCCESS.

Referenced by arrow_btsp_solve(), and constrained_shake_deep_apply().

7.23.3.40 int arrow_util_CCdatagroup_init_matrix (int *size*, CCdatagroup * *dat*)

Initializes an upper-diagonal matrix norm structure for Concorde that is ready to be filled in with values.

Parameters:

size [in] the number of cities/vertices

dat [out] the CCdatagroup structure to create

Definition at line 115 of file util.c.

References ARROW_FAILURE, arrow_print_error, and ARROW_SUCCESS.

Referenced by arrow_btsp_fun_apply(), and arrow_problem_abtsp_to_sbtsp().

7.23.3.41 void arrow_util_CCdatagroup_shallow_copy (CCdatagroup * *from*, CCdatagroup * *to*)

Makes a shallow copy of the Concorde CCdatagroup structure.

Parameters:

from [in] the CCdatagroup structure to copy from

to [out] the CCdatagroup structure to copy to

Definition at line 89 of file util.c.

7.23.3.42 int arrow_util_create_int_array (int *size*, int ** *array*) [inline]

Creates an integer array.

Parameters:

size [in] size of array

array [out] pointer to array that will be created

Definition at line 15 of file util.c.

References ARROW_FAILURE, arrow_print_error, and ARROW_SUCCESS.

Referenced by arrow_bap_solve(), arrow_bbssp_biconnected(), arrow_bintree_to_array(), arrow_btsp_result_init(), arrow_tsp_result_init(), arrow_util_create_int_matrix(), main(), and strongly_connected().

7.23.3.43 int arrow_util_create_int_matrix (int *rows*, int *cols*, int *** *matrix*, int ** *space*) [inline]

Creates a full integer matrix.

Parameters:

rows [in] number of rows

cols [in] number of columns

matrix [out] pointer to matrix that will be created

space [out] pointer to matrix space that will be created

Definition at line 27 of file util.c.

References ARROW_FAILURE, arrow_print_error, ARROW_SUCCESS, and arrow_util_create_int_array().

Referenced by arrow_bap_solve(), and arrow_dcbpb_solve().

7.23.3.44 void arrow_util_print_error (const char * *file_name*, int *line_num*, const char * *message*) [inline]

Prints an error message to stderr with consistent formatting.

Parameters:

file_name [in] file error occured in

line_num [in] line number error occured at

message [in] error message to write

Definition at line 56 of file util.c.

7.23.3.45 void arrow_util_print_program_args (int *argc*, char * *argv*[ ], FILE * *out*)

Prints out the given program arguments to the specified file.

Parameters:

argc [in] the number of arguments

argv [in] the program arugment array

out [in] the file handle to print out to

Definition at line 195 of file util.c.

Referenced by main().

7.23.3.46 int arrow_util_random () [inline]

Returns a random number between 0 and RAND_MAX (normally, RAND_MAX = INT_MAX).

Returns:

a random integer.

Definition at line 215 of file util.c.

7.23.3.47 int arrow_util_random_between (int *min*, int *max*) [inline]

Returns a random number between min and max.

Parameters:

min [in] the minimum random number to return

max [in] the maximum random number to return

Returns:

a random integer in the range [min, max]

Definition at line 221 of file util.c.

Referenced by constrained_shake_deep_apply().

7.23.3.48 void arrow_util_random_seed (int *seed*)

Seeds the random number generator. Pass a value of 0 to seed with the current time.

Parameters:

seed [in] the random number seed.

Definition at line 206 of file util.c.

Referenced by main().

7.23.3.49 void arrow_util_redirect_stdout_to_file (const char * *filename*, int * *old_stream*)

Redirects STDOUT stream to a file (can be used to completely surpress output by directing to /dev/null).

Parameters:

filename [in] name of file to direct STDOUT to

old_stream [out] existing file handle for STDOUT stream (necessary for restoring stream afterwards)

Definition at line 69 of file util.c.

Referenced by main().

7.23.3.50 int arrow_util_regex_match (char * *string*, char * *pattern*)

Determines if the given string turns up a match for the given regular expression pattern.

Parameters:

string [in] the string to match against

pattern [in] the regular expression pattern to match

Returns:

ARROW_TRUE if a match is found, ARROW_FALSE if not.

Definition at line 177 of file util.c.

References ARROW_FALSE, and ARROW_TRUE.

Referenced by is_asymmetric(), and is_symmetric().

7.23.3.51 void arrow_util_restore_stdout (int *old_stream*)

Restores STDOUT stream that's been redirected.

Parameters:

old_stream [in] existing file handle for STDOUT stream

Definition at line 78 of file util.c.

Referenced by main().

7.23.3.52 void arrow_util_sbtsp_to_abstp_tour (arrow_problem * *problem*, int * *old_tour*, int * *new_tour*)

Definition at line 246 of file util.c.

References arrow_problem::get_cost, and arrow_problem::size.

Referenced by main().

7.23.3.53 void arrow_util_write_tour (arrow_problem * *problem*, char * *comment*, int * *tour*, FILE * *out*)

Definition at line 227 of file util.c.

References arrow_problem::name, and arrow_problem::size.

Referenced by main().

7.23.3.54 double arrow_util_zeit () [inline]

Used to measure timings.

Returns:

a value representing the CPU time in seconds

Definition at line 63 of file util.c.

Referenced by arrow_2mb_solve(), arrow_bap_solve(), arrow_bbssp_solve(), arrow_bscssp_solve(), arrow_btsp_solve(), arrow_dcbpb_solve(), arrow_tsp_exact_solve(), arrow_tsp_lk_solve(), and main().

7.24 lib/bintree.c File Reference

Binary tree implementation.

```
#include "arrow.h"
```

Functions

- int construct_node (arrow_bintree_node **node, int value)
  Constructs a new node structure with the given value.
- void destruct_node (arrow_bintree_node *node)
  Frees the memory of the given node and its child nodes.
- int insert_at (arrow_bintree *tree, arrow_bintree_node *node, int value)
  Inserts a given value into the tree at the given node, or one of its child nodes.
- void fill_array (arrow_bintree_node *node, int **array, int *pos)
  Recursive helper function to fill an array in nondecreasing order.
- void arrow_bintree_init (arrow_bintree *tree)
  Initializes the binary tree data structure.
- void arrow_bintree_destruct (arrow_bintree *tree)
  Destructs a binary tree data structure.
- int arrow_bintree_insert (arrow_bintree *tree, int value)
  Inserts a value into the binary tree.
- int arrow_bintree_to_array (arrow_bintree *tree, int **array)
  Initializes the binary tree data structure.

7.24.1 Detailed Description

Binary tree implementation.

Methods for working with Arrow's binary tree data structure.

Author:

John LaRusic

Definition in file bintree.c.

7.24.2 Function Documentation

7.24.2.1 void arrow_bintree_destruct (arrow_bintree * *tree*)

Destructs a binary tree data structure.

Parameters:

tree [out] binary tree structure

Definition at line 60 of file bintree.c.

References destruct_node(), arrow_bintree::root_node, and arrow_bintree::size.

Referenced by arrow_problem_info_get(), and constrained_shake_deep_apply().

7.24.2.2 void arrow_bintree_init (arrow_bintree * *tree*)

Initializes the binary tree data structure.

Parameters:

tree [out] binary tree structure

Definition at line 53 of file bintree.c.

References arrow_bintree::root_node, and arrow_bintree::size.

Referenced by arrow_problem_info_get(), and constrained_shake_deep_apply().

7.24.2.3 int arrow_bintree_insert (arrow_bintree * *tree*, int *value*)

Inserts a value into the binary tree.

Parameters:

tree [out] binary tree structure

value [in] value to insert into tree

Definition at line 67 of file bintree.c.

References construct_node(), insert_at(), arrow_bintree::root_node, and arrow_bintree::size.

Referenced by arrow_problem_info_get(), and constrained_shake_deep_apply().

7.24.2.4 int arrow_bintree_to_array (arrow_bintree * *tree*, int ** *array*)

Initializes the binary tree data structure.

Parameters:

tree [out] binary tree structure

array [out] array to be created and filled

Definition at line 87 of file bintree.c.

References ARROW_SUCCESS, arrow_util_create_int_array(), fill_array(), arrow_bintree::root_node, and arrow_bintree::size.

Referenced by arrow_problem_info_get(), and constrained_shake_deep_apply().

7.24.2.5 int construct_node (arrow_bintree_node ** *node*, int *value*)

Constructs a new node structure with the given value.

Parameters:

node [out] pointer to node structure

value [in] value to assign to new node

Definition at line 105 of file bintree.c.

References ARROW_ERROR_FATAL, ARROW_FALSE, arrow_print_error, and ARROW_SUCCESS.

Referenced by arrow_bintree_insert(), and insert_at().

7.24.2.6 void destruct_node (arrow_bintree_node * *node*)

Frees the memory of the given node and its child nodes.

Parameters:

node [out] node structure

Definition at line 124 of file bintree.c.

References ARROW_TRUE, arrow_bintree_node::has_left_node, arrow_bintree_node::has_right_node, arrow_bintree_node::left_node, and arrow_bintree_node::right_node.

Referenced by arrow_bintree_destruct().

7.24.2.7 void fill_array (arrow_bintree_node * *node*, int ** *array*, int * *pos*)

Recursive helper function to fill an array in nondecreasing order.

Parameters:

node [out] pointer to an node structure

array [out] pointer to array to fill

pos [out] current position in array

Definition at line 190 of file bintree.c.

References arrow_bintree_node::data, arrow_bintree_node::has_left_node, arrow_bintree_node::has_right_node, arrow_bintree_node::left_node, and arrow_bintree_node::right_node.

Referenced by arrow_bintree_to_array().

7.24.2.8 int insert_at (arrow_bintree * *tree*, arrow_bintree_node * *node*, int *value*)

Inserts a given value into the tree at the given node, or one of its child nodes.

Parameters:

tree [in] pointer to tree structure

node [out] pointer to node structure

value [in] value to assign to new node

Definition at line 141 of file bintree.c.

References ARROW_ERROR_FATAL, ARROW_FALSE, ARROW_SUCCESS, ARROW_TRUE, construct_node(), arrow_bintree_node::data, arrow_bintree_node::has_left_node, arrow_bintree_node::has_right_node, arrow_bintree_node::left_node, arrow_bintree_node::right_node, and arrow_bintree::size.

Referenced by arrow_bintree_insert().

7.25 lib/btsp_fun.c File Reference

Cost matrix transformation functions.

```
#include "arrow.h"
```

Data Structures

- struct basic_data

  Concorde userdat structure for basic cost matrix function.

- struct constrained_data

  Concorde userdat structure for constrained cost matrix function.

- struct constrained_shake_data

  Concorde userdat structure for constrained shake cost matrix function.

Functions

- int basic_shallow_apply (arrow_btsp_fun *fun, arrow_problem *old_problem, int delta, arrow_problem *new_problem)

  Applies basic BTSP function to the cost matrix of the old problem to create the new problem (shallow copy).

- int basic_deep_apply (arrow_btsp_fun *fun, arrow_problem *old_problem, int delta, arrow_problem *new_problem)

  Applies basic BTSP function to the cost matrix of the old problem to create the new problem (deep copy).

- void basic_destruct (arrow_btsp_fun *fun)

  Destructs a basic BTSP function structure.

- int basic_feasible (arrow_btsp_fun *fun, arrow_problem *problem, int delta, double tour_length, int *tour)

  Determines if the given tour is feasible or not.

- static int basic_edgelen (int i, int j, struct CCdatagroup *dat)

  Concorde edge length function for the basic cost matrix function. Returns the cost C[i,j].

- int basic_atsp_deep_apply (arrow_btsp_fun *fun, arrow_problem *old_problem, int delta, arrow_problem *new_problem)

  Applies basic ABTSP function to the cost matrix of the old problem to create the new problem (deep copy).

- void basic_atsp_destruct (arrow_btsp_fun *fun)

  Destructs a basic ABTSP function structure.

- int basic_atsp_feasible (arrow_btsp_fun *fun, arrow_problem *problem, int delta, double tour_length, int *tour)

  Determines if the given tour is feasible or not for the basic ATSP transformation.

- int constrained_shallow_apply (arrow_btsp_fun *fun, arrow_problem *old_problem, int delta, arrow_problem *new_problem)

  Applies constrained BTSP function to the cost matrix of the old problem to create the new problem (shallow copy).

- int constrained_deep_apply (arrow_btsp_fun *fun, arrow_problem *old_problem, int delta, arrow_problem *new_problem)

  Applies constrained BTSP function to the cost matrix of the old problem to create the new problem (deep copy).

- void constrained_destruct (arrow_btsp_fun *fun)

  Destructs constrained BTSP function structure.

- static int constrained_edgelen (int i, int j, struct CCdatagroup *dat)

  Concorde edge length function for the constrained cost matrix function. Returns the cost C[i,j].

- int constrained_shake_deep_apply (arrow_btsp_fun *fun, arrow_problem *old_problem, int delta, arrow_problem *new_problem)

  Applies constrained shake BTSP function to the cost matrix of the old problem to create the new problem (deep copy).

- void constrained_shake_destruct (arrow_btsp_fun *fun)

  Destructs constrained shake BTSP function structure.

- int constrained_shake_feasible (arrow_btsp_fun *fun, arrow_problem *problem, int delta, double tour_length, int *tour)

  Determines if the given tour is feasible or not.

- int arrow_btsp_fun_apply (arrow_btsp_fun *fun, arrow_problem *old_problem, int delta, arrow_problem *new_problem)

  Applies the given function to the given problem to create a new problem.

- void arrow_btsp_fun_destruct (arrow_btsp_fun *fun)

  Destructs a function structure.

- int arrow_btsp_fun_basic (int shallow, arrow_btsp_fun *fun)

  Basic BTSP to TSP function.

- int arrow_btsp_fun_basic_atsp (int shallow, arrow_btsp_fun *fun)

  Basic BTSP to TSP function for asymmetric problem instances.

- int arrow_btsp_fun_constrained (int shallow, double feasible_length, int infinity, arrow_btsp_fun *fun)

  Constrained BTSP to TSP function.

- int arrow_btsp_fun_constrained_shake (int shallow, double feasible_length, int infinity, int rand_min, int rand_max, arrow_problem *problem, arrow_problem_info *info, arrow_btsp_fun *fun)

  Constrained "Shake" BTSP to TSP function.

7.25.1 Detailed Description

Cost matrix transformation functions.

Cost matrix transformation functions for the bottleneck traveling salesman problem (BTSP).

Author:

John LaRusic

Definition in file btsp_fun.c.

7.25.2 Function Documentation

7.25.2.1 int arrow_btsp_fun_apply (arrow_btsp_fun * *fun*, arrow_problem * *old_problem*, int *delta*, arrow_problem * *new_problem*)

Applies the given function to the given problem to create a new problem.

Parameters:

fun [in] function structure

old_problem [in] existing problem

delta [in] delta parameter

new_problem [out] new problem to create

Definition at line 213 of file btsp_fun.c.

References arrow_btsp_fun::apply, ARROW_ERROR_FATAL, ARROW_FAILURE, arrow_print_error, ARROW_SUCCESS, arrow_util_CCdatagroup_init_matrix(), arrow_problem::data, arrow_problem::get_cost, arrow_problem::name, arrow_btsp_fun::shallow, arrow_problem::shallow, and arrow_problem::size.

Referenced by feasible().

7.25.2.2 int arrow_btsp_fun_basic (int *shallow*, arrow_btsp_fun * *fun*)

Basic BTSP to TSP function.

Parameters:

shallow [in] ARROW_TRUE for shallow copy, ARROW_FALSE for deep

fun [out] function structure

Definition at line 249 of file btsp_fun.c.

References arrow_btsp_fun::apply, ARROW_FALSE, ARROW_SUCCESS, ARROW_TRUE, basic_deep_apply(), basic_destruct(), basic_feasible(), basic_shallow_apply(), arrow_btsp_fun::data, arrow_btsp_fun::destruct, arrow_btsp_fun::feasible, arrow_btsp_fun::feasible_length, and arrow_btsp_fun::shallow.

Referenced by main().

7.25.2.3 int arrow_btsp_fun_basic_atsp (int *shallow*, arrow_btsp_fun * *fun*)

Basic BTSP to TSP function for asymmetric problem instances.

Parameters:

shallow [in] ARROW_TRUE for shallow copy, ARROW_FALSE for deep

fun [out] function structure

Definition at line 270 of file btsp_fun.c.

References arrow_btsp_fun::apply, ARROW_FAILURE, ARROW_FALSE, arrow_print_error, ARROW_SUCCESS, basic_atsp_deep_apply(), basic_atsp_destruct(), basic_atsp_feasible(), arrow_btsp_fun::destruct, arrow_btsp_fun::feasible, arrow_btsp_fun::feasible_length, and arrow_btsp_fun::shallow.

Referenced by main().

7.25.2.4 int arrow_btsp_fun_constrained (int *shallow*, double *feasible_length*, int *infinity*, arrow_btsp_fun * *fun*)

Constrained BTSP to TSP function.

Parameters:

shallow [in] ARROW_TRUE for shallow copy, ARROW_FALSE for deep

feasible_length [in] length of feasible tour

infinity [in] value to use as "infinity"

fun [out] function structure

Definition at line 291 of file btsp_fun.c.

References arrow_btsp_fun::apply, ARROW_ERROR_FATAL, ARROW_FALSE, arrow_print_error, ARROW_SUCCESS, ARROW_TRUE, basic_feasible(), constrained_deep_apply(), constrained_destruct(), constrained_shallow_apply(), arrow_btsp_fun::data, arrow_btsp_fun::destruct, arrow_btsp_fun::feasible, arrow_btsp_fun::feasible_length, and arrow_btsp_fun::shallow.

Referenced by main().

7.25.2.5 int arrow_btsp_fun_constrained_shake (int *shallow*, double *feasible_length*, int *infinity*, int *rand_min*, int *rand_max*, arrow_problem * *problem*, arrow_problem_info * *info*, arrow_btsp_fun * *fun*)

Constrained "Shake" BTSP to TSP function.

Parameters:

shallow [in] ARROW_TRUE for shallow copy, ARROW_FALSE for deep

feasible_length [in] length of feasible tour

infinity [in] value to use as "infinity"

rand_min [in] minimum random value to generate

rand_max [in] maximum random value to generate

problem [in] the problem the shake is based upon

info [in] information about the original problem

fun [out] function structure

Definition at line 319 of file btsp_fun.c.

References arrow_btsp_fun::apply, ARROW_FAILURE, ARROW_FALSE, arrow_print_error, ARROW_SUCCESS, constrained_shake_deep_apply(), constrained_shake_destruct(), constrained_shake_feasible(), arrow_btsp_fun::data, arrow_btsp_fun::destruct, arrow_btsp_fun::feasible, arrow_btsp_fun::feasible_length, constrained_shake_data::infinity, constrained_shake_data::info, constrained_shake_data::problem, constrained_shake_data::rand_max, constrained_shake_data::rand_min, and arrow_btsp_fun::shallow.

Referenced by main().

7.25.2.6 void arrow_btsp_fun_destruct (arrow_btsp_fun * *fun*)

Destructs a function structure.

Parameters:

fun [out] function structure

Definition at line 243 of file btsp_fun.c.

References arrow_btsp_fun::destruct.

Referenced by arrow_btsp_solve_plan_destruct(), and main().

7.25.2.7 int basic_atsp_deep_apply (arrow_btsp_fun * *fun*, arrow_problem * *old_problem*, int *delta*, arrow_problem * *new_problem*)

Applies basic ABTSP function to the cost matrix of the old problem to create the new problem (deep copy).

Parameters:

fun [in] the cost matrix function

old_problem [in] the problem to apply the function to

delta [in] delta parameter

new_problem [out] the resulting new problem

Definition at line 422 of file btsp_fun.c.

References ARROW_SUCCESS, arrow_problem::data, arrow_problem::get_cost, and arrow_problem::size.

Referenced by arrow_btsp_fun_basic_atsp().

7.25.2.8 void basic_atsp_destruct (arrow_btsp_fun * *fun*)

Destructs a basic ABTSP function structure.

Parameters:

fun [out] the function structure to destruct

Definition at line 442 of file btsp_fun.c.

Referenced by arrow_btsp_fun_basic_atsp().

7.25.2.9 int basic_atsp_feasible (arrow_btsp_fun * *fun*, arrow_problem * *problem*, int *delta*, double *tour_length*, int * *tour*)

Determines if the given tour is feasible or not for the basic ATSP transformation.

Parameters:

fun [in] function structure

problem [in] the problem to check against

delta [in] delta parameter

tour_length [in] the length of the given tour

tour [in] the tour in node-node format

Returns:

ARROW_TRUE if the tour is feasible, ARROW_FALSE if not

Definition at line 446 of file btsp_fun.c.

References ARROW_FALSE, ARROW_TRUE, arrow_problem::get_cost, and arrow_problem::size.

Referenced by arrow_btsp_fun_basic_atsp().

7.25.2.10 int basic_deep_apply (arrow_btsp_fun * *fun*, arrow_problem * *old_problem*, int *delta*, arrow_problem * *new_problem*)

Applies basic BTSP function to the cost matrix of the old problem to create the new problem (deep copy).

Parameters:

fun [in] the cost matrix function

old_problem [in] the problem to apply the function to

delta [in] delta parameter

new_problem [out] the resulting new problem

Definition at line 387 of file btsp_fun.c.

References ARROW_SUCCESS, arrow_problem::data, arrow_problem::get_cost, and arrow_problem::size.

Referenced by arrow_btsp_fun_basic().

7.25.2.11 void basic_destruct (arrow_btsp_fun * *fun*)

Destructs a basic BTSP function structure.

Parameters:

fun [out] the function structure to destruct

Definition at line 403 of file btsp_fun.c.

Referenced by arrow_btsp_fun_basic().

7.25.2.12 static int basic_edgelen (int *i*, int *j*, struct CCdatagroup * *dat*) [static]

Concorde edge length function for the basic cost matrix function. Returns the cost C[i,j].

Parameters:

i [in] node i

j [in] node j

dat [in] Concorde data structure.

Definition at line 414 of file btsp_fun.c.

References basic_data::dat.

Referenced by basic_shallow_apply().

7.25.2.13 int basic_feasible (arrow_btsp_fun * *fun*, arrow_problem * *problem*, int *delta*, double *tour_length*, int * *tour*)

Determines if the given tour is feasible or not.

Parameters:

fun [in] function structure

problem [in] the problem to check against

delta [in] delta parameter

tour_length [in] the length of the given tour

tour [in] the tour in node-node format

Returns:

ARROW_TRUE if the tour is feasible, ARROW_FALSE if not

Definition at line 407 of file btsp_fun.c.

References ARROW_FALSE, and ARROW_TRUE.

Referenced by arrow_btsp_fun_basic(), and arrow_btsp_fun_constrained().

7.25.2.14 int basic_shallow_apply (arrow_btsp_fun * *fun*, arrow_problem * *old_problem*, int *delta*, arrow_problem * *new_problem*)

Applies basic BTSP function to the cost matrix of the old problem to create the new problem (shallow copy).

Parameters:

fun [in] the cost matrix function

old_problem [in] the problem to apply the function to

delta [in] delta parameter

new_problem [out] the resulting new problem

Definition at line 364 of file btsp_fun.c.

References ARROW_FAILURE, arrow_print_error, ARROW_SUCCESS, basic_edgelen(), basic_data::dat, arrow_problem::data, and basic_data::delta.

Referenced by arrow_btsp_fun_basic().

7.25.2.15 int constrained_deep_apply (arrow_btsp_fun * *fun*, arrow_problem * *old_problem*, int *delta*, arrow_problem * *new_problem*)

Applies constrained BTSP function to the cost matrix of the old problem to create the new problem (deep copy).

Parameters:

fun [in] the cost matrix function

old_problem [in] the problem to apply the function to

delta [in] delta parameter

new_problem [out] the resulting new problem

Definition at line 500 of file btsp_fun.c.

References ARROW_SUCCESS, arrow_problem::data, arrow_btsp_fun::data, arrow_problem::get_cost, and arrow_problem::size.

Referenced by arrow_btsp_fun_constrained().

7.25.2.16 void constrained_destruct (arrow_btsp_fun * *fun*)

Destructs constrained BTSP function structure.

Parameters:

fun [out] the function structure to destruct

Definition at line 518 of file btsp_fun.c.

References arrow_btsp_fun::data.

Referenced by arrow_btsp_fun_constrained().

7.25.2.17 static int constrained_edgelen (int *i*, int *j*, struct CCdatagroup * *dat*) [static]

Concorde edge length function for the constrained cost matrix function. Returns the cost C[i,j].

Parameters:

i [in] node i

j [in] node j

dat [in] Concorde data structure.

Definition at line 528 of file btsp_fun.c.

References constrained_data::dat, and constrained_data::infinity.

Referenced by constrained_shallow_apply().

7.25.2.18 int constrained_shake_deep_apply (arrow_btsp_fun * *fun*, arrow_problem * *old_problem*, int *delta*, arrow_problem * *new_problem*)

Applies constrained shake BTSP function to the cost matrix of the old problem to create the new problem (deep copy).

Parameters:

fun [in] the cost matrix function

old_problem [in] the problem to apply the function to

delta [in] delta parameter

new_problem [out] the resulting new problem

Definition at line 536 of file btsp_fun.c.

References arrow_bintree_destruct(), arrow_bintree_init(), arrow_bintree_insert(), arrow_bintree_to_-array(), ARROW_FAILURE, arrow_print_error, ARROW_SUCCESS, arrow_util_binary_search(), arrow_util_random_between(), arrow_problem_info::cost_list, arrow_problem_info::cost_list_length, arrow_problem::data, arrow_btsp_fun::data, arrow_problem::get_cost, constrained_shake_data::infinity, constrained_shake_data::info, constrained_shake_data::rand_max, constrained_shake_data::rand_min, arrow_problem::size, and arrow_bintree::size.

Referenced by arrow_btsp_fun_constrained_shake().

7.25.2.19 void constrained_shake_destruct (arrow_btsp_fun * *fun*)

Destructs constrained shake BTSP function structure.

Parameters:

fun [out] the function structure to destruct

Definition at line 597 of file btsp_fun.c.

References arrow_btsp_fun::data.

Referenced by arrow_btsp_fun_constrained_shake().

7.25.2.20 int constrained_shake_feasible (arrow_btsp_fun * *fun*, arrow_problem * *problem*, int *delta*, double *tour_length*, int * *tour*)

Determines if the given tour is feasible or not.

Parameters:

fun [in] function structure

problem [in] the problem to check against

delta [in] delta parameter

tour_length [in] the length of the given tour

tour [in] the tour in node-node format

Returns:

ARROW_TRUE if the tour is feasible, ARROW_FALSE if not

Definition at line 607 of file btsp_fun.c.

References ARROW_FALSE, ARROW_TRUE, arrow_btsp_fun::data, arrow_btsp_fun::feasible_length, arrow_problem::get_cost, and arrow_problem::size.

Referenced by arrow_btsp_fun_constrained_shake().

7.25.2.21 int constrained_shallow_apply (arrow_btsp_fun * *fun*, arrow_problem * *old_problem*, int *delta*, arrow_problem * *new_problem*)

Applies constrained BTSP function to the cost matrix of the old problem to create the new problem (shallow copy).

Parameters:

fun [in] the cost matrix function

old_problem [in] the problem to apply the function to

delta [in] delta parameter

new_problem [out] the resulting new problem

Definition at line 476 of file btsp_fun.c.

References ARROW_FAILURE, arrow_print_error, ARROW_SUCCESS, constrained_edgelen(), constrained_data::dat, arrow_btsp_fun::data, arrow_problem::data, constrained_data::delta, and constrained_data::infinity.

Referenced by arrow_btsp_fun_constrained().

7.26 lib/options.c File Reference

Helper for parsing program options.

```
#include "arrow.h"
```

Functions

- void print_usage (char *program_name, char *usage)
- void print_version (char *program_name)
- void print_help (int num_opts, arrow_option options[ ], char *description)
- int arrow_options_parse (int num_opts, arrow_option options[ ], char *description, char *usage, int argc, char *argv[ ], int *opt_ind)

7.26.1 Detailed Description

Helper for parsing program options.

For helping parse all those pesky program options!

Author:

John LaRusic

Definition in file options.c.

7.26.2 Function Documentation

7.26.2.1 int arrow_options_parse (int *num_opts*, arrow_option *options*[ ], char * *description*, char * *usage*, int *argc*, char * *argv*[ ], int * *opt_ind*)

Definition at line 24 of file options.c.

References ARROW_FAILURE, ARROW_FALSE, ARROW_OPTION_DOUBLE, ARROW_OPTION_INT, ARROW_OPTION_STRING, arrow_print_error, ARROW_SUCCESS, ARROW_TRUE, arrow_option::long_option, print_help(), print_usage(), print_version(), and arrow_option::short_option.

Referenced by main().

7.26.2.2 void print_help (int *num_opts*, arrow_option *options*[ ], char * *description*)

Definition at line 207 of file options.c.

7.26.2.3 void print_usage (char * *program_name*, char * *usage*)

Definition at line 186 of file options.c.

7.26.2.4 void print_version (char * *program_name*)

Definition at line 192 of file options.c.

References ARROW_VERSION.

7.27 lib/problem.c File Reference

Functions for working with problem data.

```
#include "arrow.h"
```

Functions

- int is_symmetric (char *file_name)

  Determines if the given file is a symmetric TSPLIB file.

- int is_asymmetric (char *file_name)

  Determines if the given file is an asymmetric TSPLIB file.

- int read_atsp (char *file_name, arrow_problem *problem)

  Reads an asymmetric TSPLIB file (.atsp).*

- static int fullmatrix_edgelen (int i, int j, CCdatagroup *dat)

  Edge length function for full matrix data.

- int arrow_problem_read (char *file_name, arrow_problem *problem)

  Reads a problem from a TSPLIB file.

- void arrow_problem_destruct (arrow_problem *problem)

  Deallocates problem data structure.

- int arrow_problem_info_get (arrow_problem *problem, arrow_problem_info *info)

  Builds ordered cost list and finds min/max cost in a problem.

- void arrow_problem_info_destruct (arrow_problem_info *info)

  Deallocates problem info data structure.

- void arrow_problem_print (arrow_problem *problem)

  Prints out information about a problem.

- int arrow_problem_get_cost (arrow_problem *problem, int i, int j)

  Retrieves cost between nodes i and j.

- int arrow_problem_read_tour (char *file_name, int size, int *tour)

  Reads a TSPLIB tour file.

- int arrow_problem_abtsp_to_sbtsp (arrow_problem *old_problem, int infinity, arrow_problem *new_problem)

  Transforms an asymmetric BTSP problem of n nodes into a symmetric BTSP problem with 2n nodes.

7.27.1 Detailed Description

Functions for working with problem data.

Function implemenations for working with problem data, generally manipulating the arrow_problem data structure.

Author:

John LaRusic

Definition in file problem.c.

7.27.2 Function Documentation

7.27.2.1 int arrow_problem_abtsp_to_sbtsp (arrow_problem * *old_problem*, int *infinity*, arrow_problem * *new_problem*)

Transforms an asymmetric BTSP problem of n nodes into a symmetric BTSP problem with 2n nodes.

Parameters:

old_problem [in] the asymmetric problem

infinity [in] value to use as "infinity"

new_problem [out] the new symmetric problem

Definition at line 228 of file problem.c.

References ARROW_FAILURE, ARROW_FALSE, arrow_print_error, arrow_problem_get_cost(), ARROW_SUCCESS, ARROW_TRUE, arrow_util_CCdatagroup_init_matrix(), arrow_problem::data, arrow_problem::get_cost, arrow_problem::name, arrow_problem::shallow, arrow_problem::size, and arrow_problem::symmetric.

Referenced by main().

7.27.2.2 void arrow_problem_destruct (arrow_problem * *problem*)

Deallocates problem data structure.

Parameters:

problem [out] problem data structure

Definition at line 98 of file problem.c.

References arrow_problem::data, and arrow_problem::shallow.

Referenced by feasible(), and main().

7.27.2.3 int arrow_problem_get_cost (arrow_problem * *problem*, int *i*, int *j*) [inline]

Retrieves cost between nodes i and j.

Parameters:

problem [in] pointer to arrow_problem structure

i [in] id of starting node

j [in] id of ending node

Returns:

cost between node i and node j

Definition at line 213 of file problem.c.

References arrow_problem::data.

Referenced by arrow_problem_abtsp_to_sbtsp(), and arrow_problem_read().

7.27.2.4 void arrow_problem_info_destruct (arrow_problem_info * *info*)

Deallocates problem info data structure.

Parameters:

info [in] problem info data structure

Definition at line 158 of file problem.c.

References arrow_problem_info::cost_list.

Referenced by main().

7.27.2.5 int arrow_problem_info_get (arrow_problem * *problem*, arrow_problem_info * *info*)

Builds ordered cost list and finds min/max cost in a problem.

Parameters:

problem [in] problem data structure

info [out] problem info data structure

Definition at line 115 of file problem.c.

References arrow_bintree_destruct(), arrow_bintree_init(), arrow_bintree_insert(), arrow_bintree_-to_array(), ARROW_SUCCESS, ARROW_TRUE, arrow_problem_info::cost_list, arrow_problem_-info::cost_list_length, arrow_problem::get_cost, arrow_problem_info::max_cost, arrow_problem_-info::min_cost, arrow_bintree::size, arrow_problem::size, and arrow_problem::symmetric.

Referenced by main().

7.27.2.6 void arrow_problem_print (arrow_problem * *problem*)

Prints out information about a problem.

Parameters:

problem [in] problem data structure

Definition at line 165 of file problem.c.

References arrow_debug, arrow_problem::get_cost, and arrow_problem::size.

7.27.2.7 int arrow_problem_read (char * *file_name*, arrow_problem * *problem*)

Reads a problem from a TSPLIB file.

Parameters:

file_name [in] path to TSPLIB file

problem [out] problem data structure

Definition at line 54 of file problem.c.

References arrow_debug, ARROW_FAILURE, ARROW_FALSE, arrow_print_error, arrow_problem_-get_cost(), ARROW_SUCCESS, ARROW_TRUE, arrow_problem::data, arrow_problem::get_cost, is_-asymmetric(), is_symmetric(), arrow_problem::name, read_atsp(), arrow_problem::shallow, arrow_-problem::size, and arrow_problem::symmetric.

Referenced by main().

7.27.2.8 int arrow_problem_read_tour (char * *file_name*, int *size*, int * *tour*)

Reads a TSPLIB tour file.

Parameters:

file_name [in] the TSPLIB tour file to read

size [in] the number of cities in the tour

tour [out] an array to hold the tour in node-node format

Definition at line 219 of file problem.c.

References ARROW_ERROR_FATAL, ARROW_SUCCESS, and CONCORDE_SUCCESS.

Referenced by main().

7.27.2.9 static int fullmatrix_edgelen (int *i*, int *j*, CCdatagroup * *dat*) [static]

Edge length function for full matrix data.

Parameters:

i [in] node i

j [in] node j

dat [in] Concorde problem data structure

Returns:

The cost C[i,j] between nodes i and j

Definition at line 443 of file problem.c.

Referenced by read_atsp().

7.27.2.10 int is_asymmetric (char * *file_name*)

Determines if the given file is an asymmetric TSPLIB file.

Parameters:

file_name [in] the path to the TSPLIB file

Returns:

ARROW_TRUE if the file is for asymmetric data, ARROW_FALSE if not

Definition at line 281 of file problem.c.

References arrow_util_regex_match().

Referenced by arrow_problem_read().

7.27.2.11 int is_symmetric (char * *file_name*)

Determines if the given file is a symmetric TSPLIB file.

Parameters:

file_name [in] the path to the TSPLIB file

Returns:

ARROW_TRUE if the file is for symmetric data, ARROW_FALSE if not

Definition at line 275 of file problem.c.

References arrow_util_regex_match().

Referenced by arrow_problem_read().

7.27.2.12 int read_atsp (char * *file_name*, arrow_problem * *problem*)

Reads an asymmetric TSPLIB file (*.atsp).

Parameters:

file_name [in] the path to the TSPLIB file

problem [out] problem structure

Definition at line 287 of file problem.c.

References arrow_debug, ARROW_ERROR_FATAL, arrow_print_error, ARROW_SUCCESS, arrow_problem::data, fullmatrix_edgelen(), and arrow_problem::size.

Referenced by arrow_problem_read().

7.28 lib/util.c File Reference

Useful utility functions.

```
#include "arrow.h"
```

Functions

- int arrow_util_create_int_array (int size, int **array)
  Creates an integer array.
- int arrow_util_create_int_matrix (int rows, int cols, int ***matrix, int **space)
  Creates a full integer matrix.
- void arrow_util_print_error (const char *file_name, int line_num, const char *message)
  Prints an error message to stderr with consistent formatting.
- double arrow_util_zeit ()
  Used to measure timings.
- void arrow_util_redirect_stdout_to_file (const char *filename, int *old_stream)
  Redirects STDOUT stream to a file (can be used to completely surpress output by directing to /dev/null).
- void arrow_util_restore_stdout (int old_stream)
  Restores STDOUT stream that's been redirected.
- void arrow_util_CCdatagroup_shallow_copy (CCdatagroup *from, CCdatagroup *to)
  Makes a shallow copy of the Concorde CCdatagroup structure.
- int arrow_util_CCdatagroup_init_matrix (int size, CCdatagroup *dat)
  Initializes an upper-diagonal matrix norm structure for Concorde that is ready to be filled in with values.
- int arrow_util_binary_search (int *array, int size, int element, int *pos)
  Performs a binary search to find the wanted element in a sorted integer array.
- int arrow_util_regex_match (char *string, char *pattern)
  Determines if the given string turns up a match for the given regular expression pattern.
- void arrow_util_print_program_args (int argc, char *argv[ ], FILE *out)
  Prints out the given program arguments to the specified file.
- void arrow_util_random_seed (int seed)
  Seeds the random number generator. Pass a value of 0 to seed with the current time.
- int arrow_util_random ()
  Returns a random number between 0 and RAND_MAX (normally, RAND_MAX = INT_MAX).
- int arrow_util_random_between (int min, int max)
  Returns a random number between min and max.
- void arrow_util_write_tour (arrow_problem *problem, char *comment, int *tour, FILE *out)
- void arrow_util_sbtsp_to_abstp_tour (arrow_problem *problem, int *old_tour, int *new_tour)

7.28.1 Detailed Description

Useful utility functions.

Useful utility functions that have general purpose throughout the library.

Author:

John LaRusic

Definition in file util.c.

7.28.2 Function Documentation

7.28.2.1 int arrow_util_binary_search (int * *array*, int *size*, int *element*, int * *pos*)

Performs a binary search to find the wanted element in a sorted integer array.

Parameters:

array [in] the array to search (note: must be sorted in non-increasing order)
size [in] size of the array
element [in] the element to find in the array
pos [out] the index where the element can be found in the array

Definition at line 145 of file util.c.

References ARROW_FAILURE, and ARROW_SUCCESS.

Referenced by arrow_btsp_solve(), and constrained_shake_deep_apply().

7.28.2.2 int arrow_util_CCdatagroup_init_matrix (int *size*, CCdatagroup * *dat*)

Initializes an upper-diagonal matrix norm structure for Concorde that is ready to be filled in with values.

Parameters:

size [in] the number of cities/vertices
dat [out] the CCdatagroup structure to create

Definition at line 115 of file util.c.

References ARROW_FAILURE, arrow_print_error, and ARROW_SUCCESS.

Referenced by arrow_btsp_fun_apply(), and arrow_problem_abtsp_to_sbtsp().

7.28.2.3 void arrow_util_CCdatagroup_shallow_copy (CCdatagroup * *from*, CCdatagroup * *to*)

Makes a shallow copy of the Concorde CCdatagroup structure.

Parameters:

from [in] the CCdatagroup structure to copy from
to [out] the CCdatagroup structure to copy to

Definition at line 89 of file util.c.

7.28.2.4 int arrow_util_create_int_array (int *size*, int ** *array*) [inline]

Creates an integer array.

Parameters:

size [in] size of array

array [out] pointer to array that will be created

Definition at line 15 of file util.c.

References ARROW_FAILURE, arrow_print_error, and ARROW_SUCCESS.

Referenced by arrow_bap_solve(), arrow_bbssp_biconnected(), arrow_bintree_to_array(), arrow_btsp_result_init(), arrow_tsp_result_init(), arrow_util_create_int_matrix(), main(), and strongly_connected().

7.28.2.5 int arrow_util_create_int_matrix (int *rows*, int *cols*, int *** *matrix*, int ** *space*) [inline]

Creates a full integer matrix.

Parameters:

rows [in] number of rows

cols [in] number of columns

matrix [out] pointer to matrix that will be created

space [out] pointer to matrix space that will be created

Definition at line 27 of file util.c.

References ARROW_FAILURE, arrow_print_error, ARROW_SUCCESS, and arrow_util_create_int_array().

Referenced by arrow_bap_solve(), and arrow_dcbpb_solve().

7.28.2.6 void arrow_util_print_error (const char * *file_name*, int *line_num*, const char * *message*) [inline]

Prints an error message to stderr with consistent formatting.

Parameters:

file_name [in] file error occured in

line_num [in] line number error occured at

message [in] error message to write

Definition at line 56 of file util.c.

7.28.2.7 void arrow_util_print_program_args (int *argc*, char * *argv*[ ], FILE * *out*)

Prints out the given program arguments to the specified file.

Parameters:

argc [in] the number of arguments

argv [in] the program arugment array

out [in] the file handle to print out to

Definition at line 195 of file util.c.

Referenced by main().

7.28.2.8 int arrow_util_random () [inline]

Returns a random number between 0 and RAND_MAX (normally, RAND_MAX = INT_MAX).

Returns:

a random integer.

Definition at line 215 of file util.c.

7.28.2.9 int arrow_util_random_between (int *min*, int *max*) [inline]

Returns a random number between min and max.

Parameters:

min [in] the minimum random number to return

max [in] the maximum random number to return

Returns:

a random integer in the range [min, max]

Definition at line 221 of file util.c.

Referenced by constrained_shake_deep_apply().

7.28.2.10 void arrow_util_random_seed (int *seed*)

Seeds the random number generator. Pass a value of 0 to seed with the current time.

Parameters:

seed [in] the random number seed.

Definition at line 206 of file util.c.

Referenced by main().

7.28.2.11 void arrow_util_redirect_stdout_to_file (const char * *filename*, int * *old_stream*)

Redirects STDOUT stream to a file (can be used to completely surpress output by directing to /dev/null).

Parameters:

filename [in] name of file to direct STDOUT to

old_stream [out] existing file handle for STDOUT stream (necessary for restoring stream afterwards)

Definition at line 69 of file util.c.

Referenced by main().

7.28.2.12 int arrow_util_regex_match (char * *string*, char * *pattern*)

Determines if the given string turns up a match for the given regular expression pattern.

Parameters:

string [in] the string to match against

pattern [in] the regular expression pattern to match

Returns:

ARROW_TRUE if a match is found, ARROW_FALSE if not.

Definition at line 177 of file util.c.

References ARROW_FALSE, and ARROW_TRUE.

Referenced by is_asymmetric(), and is_symmetric().

7.28.2.13 void arrow_util_restore_stdout (int *old_stream*)

Restores STDOUT stream that's been redirected.

Parameters:

old_stream [in] existing file handle for STDOUT stream

Definition at line 78 of file util.c.

Referenced by main().

7.28.2.14 void arrow_util_sbtsp_to_abstp_tour (arrow_problem * *problem*, int * *old_tour*, int * *new_tour*)

Definition at line 246 of file util.c.

References arrow_problem::get_cost, and arrow_problem::size.

Referenced by main().

7.28.2.15 void arrow_util_write_tour (arrow_problem * *problem*, char * *comment*, int * *tour*, FILE * *out*)

Definition at line 227 of file util.c.

References arrow_problem::name, and arrow_problem::size.

Referenced by main().

7.28.2.16 double arrow_util_zeit () [inline]

Used to measure timings.

Returns:

a value representing the CPU time in seconds

Definition at line 63 of file util.c.

Referenced by arrow_2mb_solve(), arrow_bap_solve(), arrow_bbssp_solve(), arrow_bscssp_solve(), arrow_btsp_solve(), arrow_dcbpb_solve(), arrow_tsp_exact_solve(), arrow_tsp_lk_solve(), and main().