ACO_in_AMP_sample documentation

Purpose

The sample's purpose is to illustrate how a tentative solution to the challenge posed by Beyond3D's Travelling in Style contest might look. It is in no way intended to be illustrative of an optimal solution (it definitely is not) or of production quality code (it does not qualify as such). What it does show is how code that adheres (to a sufficient extent) to the coding style guidelines should look and how the input to the solver is provided. Note that the menu component included in the sample is of no relevance to competitor's solvers – you are not expected to implement the menu as well. You are also not expected to implement a TSP file loading class / function, albeit one is included in the sample. Also note that an Abstract Base Class named **Solver** is provided, and it can be freely used by competitor's solvers – indeed, inheriting from it would make it easier for the referees to plug the solvers into the performance testing scaffolding they use.

Notes about the menu

Navigation in the TSP loading interface is done through the keys W (up) and S (down), whilst a selection is made by pressing the Enter key. The loader is capable of reading all TSPLIB formatted files, bar the XRAY problems for which the implementation is momentarily lacking.

Notes about the solver

The solver itself is neither performance optimal nor correctness optimal. We do not in any way suggest that you use the algorithm employed by the solver. Note that you will have to disable Timeout Detection and Recovery (TDR) if you want to attempt using the solver with large problems, on the GPU that does Windows rendering. If you're running the solve on a secondary GPU that does not own the display and using Windows 8, merely selecting it through the menu ensures that TDR is disabled.

Notes about the packaged problems

Alongside the solver we provide a few TSPs and ATSPs that you can use to check how your own programs fare. We place emphasis on the fact that these problems are not part of the set of problems that will be used in the scoring process. Optimal tour lengths are presented in the following table:

Problem type	Problem name	Optimal tour length
TSP	a280	2579
	burma14	3323
	dsj1000	18659688
ATSP	br17	39
	ft70	38673
	rgb443	2720

Some referee commentaries

The following are observations about the code's style coming from one of the referees. Hopefully they will help in outlining how optimal scoring in the coding style aspect is possible:

"*I think your use of such heavy abbreviations for namespaces in places is slightly offputting. It would make sense if you weren't using "using" statements so heavily, but as you lean towards using using statements it seems that clearer namespace names would be preferable. I tend to follow a policy of saying that if I give two people a word and there is any significant chance they won't abbreviate it the same way, I wouldn't abbreviate it.*

```
using namespace aba; // er…?
using namespace ant; // something about the solver I'd guess as it's ant related
using namespace hlp; // help, that one's easy but pointless
using namespace pbl; // problem? pbm seems more likely
using namespace slv; // solver
using namespace std; // ok… that's pretty standard J
```

"You use a function here:

```
for (auto i = 0u; i != cities_left; ++i) {

    queen.find_best_cand(workers);
```

That contains a barrier (and hence may not be used in divergent control flow). It is a limitation of AMP's design that it has pinched from CUDA, OpenCL etc that functions containing barriers don't compose well. I wonder if functions like that should be named to make it clear that they synchronize. Another option would be a coding style that banned use of barriers from within functions unless the barrier itself was passed as a parameter."

"While I don't like the 80 character wrap that many people go for, some of your lines are crazy in terms of the need to scroll on my macbook! "