Mutable Priority Queue Container

Release 0.01

Arnaud Gelas, Alexandre Gouaillard
and Sean Megason

October 20, 2008

Department of System Biology, Harvard Medical School,
Boston, MA 02115, USA

Abstract

When dealing with functional minimization, or maximization, it can sometimes be solved by a greedy algorithm. To implement greedy algorithm one needs priority queue container, i.e. get for a very low cost the lowest or highest element present in a sorted container. Whenever the priority of one element present in the queue needs to be modified, standard implementations, like `std::priority_queue`, can not be applied directly. VTK has is own implementation `vtkPriorityQueue` which is not templated and can only be applied for `vtkIdType` and for the minimizing the functional. We propose here an implementation of a mutable priority queue container where element, priority, and objective (minimization or maximization) are given by template arguments. Our implementation allows to minimize or maximize a given functional, and any element can be modified, deleted at any time, and with a low cost.

Contents

1 Implementation

1.1 Priority Queue

The class `itk::PriorityQueueContainer` inherits from `itk::VectorContainer` and is templated over for elements

```
template<
  typename TElementWrapper,           // the type of wrapper (direct or indirect)
  typename TElementWrapperInterface,  // the type of queue (min or max)
  typename TElementPriority = double, // the type of the priority, cost, error...
  typename TElementIdentifier = int   //  the identifier of the element
  >
class PriorityQueueContainer
```

1.2 Wrapper Interface

Direct

If you want the element to be stored in the priority, which will be in charge of their memory (the allocation is thus done in the heap), you can use a direct wrapper.

```
template<
  typename TElement,          // the element itself
  typename TElementIdentifier //  the identifier of the element
  >
class ElementWrapperInterface
```

Indirect

If you prefer to keep the element elsewhere you can use the indirect wrapper:

```
template<
  typename ElementWrapperPointerType, // pointer to the element
  typename TElementIdentifier = int   // the identifier of the element
  >
class ElementWrapperPointerInterface
```

1.3 Minimization or Maximization

You can switch from a minimum priority queue to a maximum priority queue by using the right wrapper that internally define the compare function.

Minimization

```
template<
  typename TElement,
  typename TElementPriority = double,
  typename TElementIdentifier = int
 >
class MinPriorityQueueElementWrapper
```

Maximization

```
template<
  typename TElement,
  typename TElementPriority = double,
  typename TElementIdentifier = int
 >
class MaxPriorityQueueElementWrapper
```

2 Usage

Let's consider the problem of maximizing the edge length in a surface mesh, and assume that this can be solved by a greedy algorithm where the shortest edge is collapsed. All edges should be first pushed in the container, and sorted according to their length, from the shortest to the longest. Then after each edge collapse operation, few edges length need to be updated in the container.

2.1 types definition

```
 // First define the edge length Type
typedef MeasureType PriorityType;

// then define a wrapper for the edge pointer and the endge length for a min ordering
typedef itk::MinPriorityQueueElementWrapper<
   OutputQEType*, // the Edge Type
   PriorityType   // the edge length type
   >  PriorityQueueItemType;

// now we are ready to define the priority Queue container
typedef itk::PriorityQueueContainer< PriorityQueueItemType*,
  ElementWrapperPointerInterface< PriorityQueueItemType* >,
  PriorityType
  > PriorityQueueType;

// now define the pointer type to create the object later
typedef PriorityQueueType::Pointer PriorityQueuePointer;
```

2.2 Useful methods

```
// initialize the container to no item. Do not release memory
void Clear( )

// Check if empty
bool Empty( )

 // insert element in the queue
void Push( Element element )

// look at the first element, equivalent to top()
Element Peek( )

// pop the first element from the queue
void Pop( )

// NEW - update an element already in the Container.
void Update( Element element )
```

```
// NEW - delete an element anywhere in the Container.
void Delete( Element element )
```

3 Examples

3.1 Simple Example

```
typedef MinPriorityQueueElementWrapper< int, double, int > PQElementType;
typedef PriorityQueueContainer< PQElementType, PQElementType, double, int >
  PQType;
PQType::Pointer priority_queue = PQType::New( );

vnl_random random( 12 );
int i( 0 ), element( 0 );
double value = random.drand32( -1000., 1000. );

PQElementType to_be_erased( i, value );
priority_queue->Push( to_be_erased );

for( i = 1; i < 100; i++ )
{
  value = random.drand32( -1000., 1000. );
  std::cout <<"{" <<i <<", " <<value <<"}" <<std::endl;
  priority_queue->Push( PQElementType( i, value ) );
}

i = 0;

while( !priority_queue->Empty() )
{
  element = priority_queue->Peek( ).m_Element;
  value = priority_queue->Peek( ).m_Priority;
  std::cout <<i++ <<" ** element: " <<element <<" priority: " <<value;
  std::cout <<" ** size: " <<priority_queue->Size( )<<std::endl;
  priority_queue->Pop( );
}
```

3.2 Practical Example

Will be given in subsequent papers on Decimation and Delaunay conforming.

4 Acknowledgment

This work was funded by a grant from the NHGRI (P50HG004071-02) to found the Center for in toto genomic analysis of vertebrate development.