How to Use Submodular Function Oracles

onigiri

September 9, 2014

Contents

1 Introductions

This is the document for Submodular Functions. We explain how to use and create submodular function oracles. C++ and C# are available.

2 How to Use Submodular Function Oracle

We explain about a class ”SubmodularOracle” and classes which derived from the class.

```cpp
UndirectedCut* oracle0 = new UndirectedCut(path0);
SetCover* oracle1 =new SetCover(path1);
FacilityLocation* oracle2 = new FacilityLocation(n,modular,matrix);
```

Figure 1: Example of oracle by C++

```cs
var oracle0 = new UndirectedCut(path0);
var oracle1 = new SetCover(path1);
var oracle2 = new FacilityLocation(n, modular, matrix);
```

Figure 2: Example of oracle by C#

2.1 Submodular Function Oracle: Base Class

We first explain a base class "SubmodularOracle". This is an abstract class, therefore, you cannot create a instance of this class. All oracles inherit this class.

This class support methods and properties as follows.

C++

- int N();
- double CalcValue(const int* order, int cardinality);
- void CalcBase(const int* order,double* base);
- bool IsSubmodular();
- static void GetOrder(int n, int mask,int &usedBit,int* order);
- static void GetOrder(int n, string &minimizer,int &usedBit,int* order);

C#

- int N;
- double CalcValue(int[] order, int cardinality);
- double[] CalcBase(int[] order);
- bool IsSubmodular();
- static int[] GetOrder(int n,int mask,out int usedBit);
- static int[] GetOrder(string mask,out int usedBit)

details

- N() gets the cardinality of the ground set.
- CalcValue() calculates the value $f(X)$, where $X := \cup_{0 \leq i < \mathrm{cardinality}} \{\mathrm{order}[i]\}$.
- CalcBase() calculates the base according to order, i.e., this finds the base whose $\mathrm{order}[i]$th element is $f(X_i) - f(X_{i-1})$, where $X_: = \cup_{0 \leq i <= i} \{\mathrm{order}[i]\}$. In C++, instead of returning an array, we update the second argument base.
- IsSubmodular() checks whether this oracle is submodular or not. Note that it takes $O(N^2 * N^2)$ time.

- GetOrder() is for converting a set (bitmask) to an order since our submodular function oracle need an order instead of a set. If bitmask represents a set X and we get order and usedBit by this function, then in order to know $f(X)$, we only need to call CalcValue(order, usedBit). We cannot decide this order uniquely. Note that if we use integer mask, then positions is read from right to left, but if we use string mask, then positions read from left to right. When we use string, all characters which are not '0' are seemed as '1'. n is the size of the number of positions of mask. Here, usedBit is the number of positions which is not 0. In C++, instead of returning an array, the argument order is updated.

examples

Let oracle f is a modular function on $\{0, 1, 2\}$ which satisfies
$f(\{0\}) = 10$,
$f(\{1\}) = 7$,
$f(\{2\}) = 3$.
Then, N() returns 3.
CalcValue($(2, 0, 1), 0$) returns $f(\{\}) = 0$,
CalcValue($(2, 0, 1), 1$) returns $f(\{2\}) = 3$,
CalcValue($(2, 0, 1), 2$) returns $f(\{2, 0\}) = 13$,
CalcValue($(2, 0, 1), 3$) returns $f(\{2, 0, 1\}) = 20$.
Of course CalcBase($(2, 0, 1)$) returns $(13 - 3, 20 - 13, 3 - 0) = (10, 7, 3)$. Let $n = 5$ and mask is a integer 6. Then, GetOrder($n = 5$, int mask, usedBit) returns order $= (1, 2, 4, 3, 0)$ and usedBit $= 3$, since $6 = 00110$ if we use binary expression. Here, 4 is the left most position and 0 is the right most position. It is important that $1, 2$ comes before than $0, 3, 4$, and orders of $1, 2$ or $0, 3, 4$ do not matter. If we use a string mask "00110", then GetOrder($n = 5$, string mask, usedBit) returns order $= (2, 3, 4, 1, 0)$ and usedBit $= 3$. Here, 4 is the right most position and 0 is the left most position.

2.2 Submodular Function Oracles: Derived Class

We next explain about derived classes. All these function normally does not have their own functions, we only explain about the constructors.

There exist two type of constructors for each submodular function oracle. One constructor needs a path of data which is created by DataCreator. The other constructor needs variables directly.

C++

- NameOfDerivedClass(string path);
- UndirectedCut(int n, double* modular, double** weight);
- DirectedCut(int n, double* modular, double** weight);
- ConnectedDetachment(int n, double* modular, map¡int,bool¿* edges);
- FacilityLocation(int n, double* modular, double** matrix);
- GraphicMatroid(int n, int V, double* modular, int* heads, int*tails);
- BinaryMatrixRank(int n, int row, double* modular, bool** columnVectors);
- NegativeSymmetricMatrixSummation(int n, double* modular, double** matrix);
- SetCover(int n, int m, double* modular, double* weight, int* length, int** edges);

C#

- NameOfDeriveClass(string path);
- UndirectedCut(int n, double[]modular, double[][] weight);
- DirectedCut(int n, double[]modular, double[][] weight);
- ConnectedDetachment(int n, double[] modular, HashSet¡int¿[] edges);
- FacilityLocation(int n, double[] modular, double[][] matrix);
- GraphicMatroid(int n, int V, double[] modular, int[]heads, int[]tails);
- BinaryMatrixRank(int n, int row, double[] modular, bool[][] columnVectors);
- NegativeSymmetricMatrixSummation(int n, double[] modular, double[][] matrix);
- SetCover(int n, int m,double[] modular, double[]weight, int[][] edges);

details All the arguments n above means the cardinality of the ground set and modular means a modular function. We explain about the other arguments.

- NameDeriveClass() is a standard constructor. We need to change the name according to what we want to use. For example, if we want to use undirected cut, we need to use UndirectedCut(). For the argument path, we need to set a path where data exists, like $\mathrm{C:\backslash Submodular\backslash UndirectedCut\backslash 10\_8}$. The data should be created by the DataCreator application.
- UndirectedCut() needs a matrix "weight" which express' weight of edges.
- DirectedCut() needs a matrix "weight" which express' weight of edges.
- ConnectedDetachment() needs incidence list "edges" of the graph of map in C++ or of HashSet C#.
- FacilityLocation() needs a matrix.
- GraphicMatroid() needs the number of cardinality "V" of a vertex set. It also needs two array, "heads" and "tails", which express edges of the graph. The ith edge is represented by $\{$heads[i], tails[i]$\}$.
- BinaryMatrixRank() needs "row" which express the number of rows of the matrix. The matrix is expressed by column vectors "columnVectors".
- NegativeSymmetricMatrixSummation() need a non-positive symmetric matrix.
- SetCover() needs the cardinality m of the other set of bipartite graph, which has weight by "weight". It also needs incidence list "edges" of the graph. In C++, we need "length", too. Here, length[i] represents the cardinality of edges[i].

3 How to Use Submodular Function Minimization Algorithms

We explain about a class ”SubmodularFunctionMinimization” and classes which derived from the class.

```
Hybrid* algo0 = new Hybrid();
BruteForce* algo1 = new BruteForce();
SFMResult sfmResult0 = algo0->Minimization(oracle);
SFMResult sfmResult1 = algo1->Minimization(oracle);
```

Figure 3: Example of minimization by C++

```
var algo0 = new BruteForce();
var sfmResult0 = algo0.Minimization(oracle0);
```

Figure 4: Example of minimization by C#

3.1 Submodular Function Minimization

SubmodularFunctionMinimization class and classes derived from it has a method SFMResult Minimization(SubmodularOracle* oracle) in C++, and SFMResult Minimization(SubmodularOracle oracle) in C#. Here, SFMResult which we discuss later is a class which includes some results of execution.

If we are given a submodular function oracle, all we have to do are to make a instance of algorithm and give it a oracle, like Figure. 3 or Figure. 3.

3.2 SFMResult Class

We next explain about ”SFMResult” class which has some information about execution results.

methods

- int N();
- long Iteration();
- long ExecutionTime();
- long OracleTime();
- long ReductionTime();
- long OracleCall();
- long BaseCall();
- long ReductionCall();
- double MinimumValue();
- string Minimizer();
- double DualValue();
- void X(double* &x);
- void Output(string path, bool withX = true);

Note that if we use C#, these functions are properties. Hence, we do not need (). Moreover, in C#, X returns an array double[], and the argument x is not required.

details

- int N() returns the cardinality of the ground set of the submodular function.
- long Iteration() returns the number of iterations of the algorithm.
- long ExecutionTime() return the total time which algorithm needed to find a minimizer.
- long OracleTime() returns the total time which algorithm needed to calculate oracle values and bases.
- long ReductionTime() return the total time which algorithm needed to execute Reduction().
- long OracleCall() returns the number of oracle call in the algorithm.

- long BaseCall() returns the number of base created in the algorithm.
- long ReductionCall() returns the number of reduction call.
- double MinimumValue() returns an oracle value by using above minimizer.
- string Minimizer() returns a bitmask expression of a minimizer. For example, if a submodular function whose cardinality of ground set is 6 has a minimizer $\{0, 3, 4\}$, then the returned value is "100110".
- double DualValue() returns the objective value of dual problem with above X. If the algorithm does not use dual formulation, then we set this value 0.
- void X(double* &x) returns a feasible solution at the end of the algorithm of the dual problem. It is not necessary to be a optimal maximizer and if the algorithm does not use dual formulation, then the returned value is NULL. If we use CaluBase() instead of CalcValue(), then this call is not counted. If we use ClacValue() instead of CalcBase() in order to create a base, then this is not counted.
- void Output(string path) writes above data to a file at path. If we set withX = false, then x is not written to the file. The out put is like Figure 3.2.

```
N                : 4
Iteration        : 22
Execurtion Time  : 0
Oracle Time      : 0
Reduction Time   : 0
Oracle Call      : 9
Base Call        : 0
Reduce Call      : 0
Minimum Value    : -139133
Minimizer        : 0110
Dual Value       : -139133
x                :
119382
-15837
-123296
934904
```

Figure 5: Output file by Output()

4 How to develop Submodular Function Oracle

We explain what to do to make submodular function oracle. When we create a new submodular function's oracle, please inherit a class "SubmodularOracle". We need to implement constructor, CalcValue(), CalcBase() (optional), and destructor (in C++).

```
class Modular: public SubmodularOracle
{
private:
   double* modular;
public:
   Modular(int n,double*modular);
   Modular(string path);
   void SetVariables(int n,double* modular);
   virtual ~Modular();
   virtual double CalcValue(const int* order,int cardinality) final override;
   virtual void CalcBase(const int* order,double* base) final override;
};
```

Figure 6: Example of a header file by C++

4.1 Constructor

We need to initialize N and fOfEmpty. N is the cardinality of the ground set and fOFEmpty is the value $f(\emptyset)$, where f is the submodular function. It is possible to initialize fOfEmpty by calling CalcValue() (in the case of Figure 4.1, we do not call CalcValue()).

4.2 CalcValue()

We need to implement double CalcValue(const int* order, int cardinality) in C++ and double CalcValue(int[]order, int cardinality) in C#. This function calculates the oracle value. Let $f(X)$ is a value which we want to calculate. Then X is expressed by "order" and "cardinality", i.e., X is defined by $\cup_{0 \leq i < \mathrm{cardinality}} \mathrm{order}[i]$.

For example, a implementation for modular function is as Figure 4.2.

4.3 CalcBase()

This function is defined by void CalcBase(const int* order, double* base) in C++ and double[] CalcBase(int[] order) in C#. This is an optional function, therefore you do not

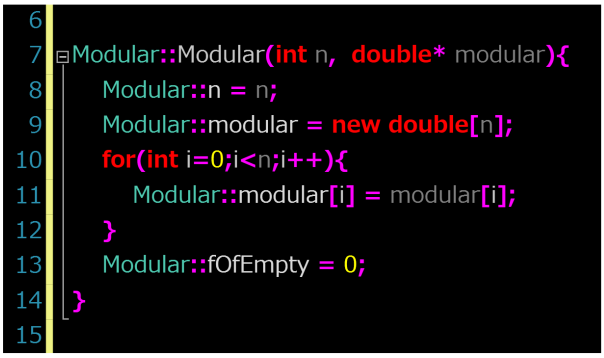

Figure 7: The implementation for constructor of a modular function

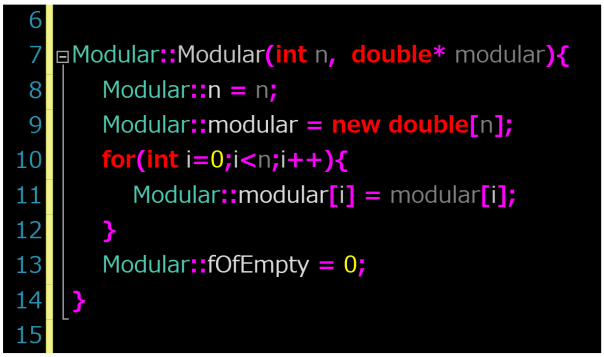

Figure 8: The implementation for GetValue() of a modular function

need to implement this function. In the case that we do not implement this function, we need to call CalcValue() for n times, where n is the cardinality of the ground set. If there exists a way to calculate a base fast, we recommend to implement this function. For example, a base of modular function is defined as Figure 4.3 in C++ and as Figure 4.3 in C#.

```
60  void Modular::CalcBase(const int* order,double* base)
61  {
62      for (int i = 0; i < n; i++)
63      {
64          base[i] = modular[i];
65      }//for i
66  }
```

Figure 9: The implementation for GetBase() of a modular function by C++

```
public virtual double[] CalcBase(int[] order)
{
    double sum = fOfEmpty;
    var res = new double[N];
    for (int i = 0; i < N; i++)
    {
        double curValue = CalcValue(order, i+1);
        res[order[i]] = curValue - sum;
        sum = curValue;
    }//for i
    return res;
}
```

Figure 10: The implementation for GetBase() of a modular function by C#

4.4 destructor

We need to implement destructor in C++. Please delete what you allocate memory.

5 How to develop Submodular Function Minimization Algorithms

We explain what to do to make submodular function minimization algorithm. When we create a new submodular function's oracle, please inherit a class "Minimization".

5.1 Minimization()

Any class which inherit "Minimization" class must implement SFMResult Minimization(SubmodularOracle* oracle) in C++ and SFMResult Minimization(SubmodularOracle oracle) in C#. Please implement call SetOracle() at the beginning of Minimization() and call SetResult() at the end of it. SetOracle needs an argument and SetResult needs three or four arguments. This function is implemented as 5.1 and 5.1.

```
public override SFMResult Minimization(SubmodularOracle oracle)
{
    SetOracle(oracle);
    double minValue = double.MaxValue;
    int minimizer = -1;
    int iteration = 0;
    for (int mask = 0; mask < (1 << N); mask++)
    {
        iteration++;
        int usedBit ;
        var order = SubmodularOracle.GetOrder(N, mask, out usedBit);
        double cur = CalcValue(order, usedBit);
        if (cur<minValue)
        {
            minValue = cur;
            minimizer = mask;
        }//if
    }//for mask
    var min = ConvertToString(minimizer,N);
    SetResult(null,min,iteration);
    return Result;
}
```

Figure 11: The implementation for Minimization() by a brute force algorithm by C++

```
    public override SFMResult Minimization(SubmodularOracle oracle)
    {
        SetOracle(oracle);
        double minValue = double.MaxValue;
        int minimizer = -1;
        int iteration = 0;
        for (int mask = 0; mask < (1 << N); mask++)
        {
            iteration++;
            int usedBit ;
            var order = SubmodularOracle.GetOrder(N, mask, out usedBit);
            double cur = CalcValue(order, usedBit);
            if (cur<minValue)
            {
                minValue = cur;
                minimizer = mask;
            }//if
        }//for mask
        var min = ConvertToString(minimizer,N);
        SetResult(null,min,iteration);
        return Result;
    }
```

Figure 12: The implementation for Minimization() of a brute force algorithm by C#