PRÁCTICA 2.B : Búsquedas Multiarranque para el Problema de la Selección de Característica

4º CURSO DEL GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS

MARINA ESTÉVEZ ALMENZAR
75571215 – T
marinaestal@correo.ugr.es

GRUPO DE PRÁCTICAS: JUEVES
PROFESOR: ÓSCAR CORDÓN

Índice

1 BREVE DESCRIPCIÓN DEL PROBLEMA

Se desarrollará el problema de **selección de características**, en el que se proporciona un conjunto inicial de características relativas a un conjunto de elementos, y cuyo objetivo es extraer un subconjunto de las mismas que maximice el acierto de clasificación de dichos elementos. La clasificación hace uso del clasificador considerado para esta práctica, el **3-NN**.

El clasificador 3-NN recibirá un conjunto de características seleccionadas de un elemento en concreto, y determinará, según dicha selección, la clase del elemento (que posteriormente será comparada con la clase que corresponde y se contará o no como un acierto).

Así, para combinar el trabajo de la selección de características y el clasificador, se han usado los 3 conjuntos de datos especificados en el guión de prácticas (**WDBC**, **Movement_Libras** y **Arrhythmia**), y el método de validación **5x2-cross**.

Los algoritmos de selección de características implementados son **BMB**, **GRASP** e **ILS**. También se ha implementado, con el fin de comparar los algoritmos nombrados, el greedy *Sequenctial Forward Selection* (**SFS**).

2 BREVE DESCRIPCIÓN DE LA APLICACIÓN DE LOS ALGORITMOS EMPLEADOS AL PROBLEMA

Los tres algoritmos implementados (**BMB, GRASP** e **ILS**) mantienen algunas similitudes en cuanto a código que cabe destacar.

Lectura de archivos .arff

Puesto que los datos nos vienen dados en ficheros con formato .arff, se ha implementado un método específico para leer este tipo de archivos. Dicho método recibe como argumento el nombre del archivo y devuelve una matriz NxN en la que cada fila corresponde a un elemento, y contiene N-1 características del mismo. En la posición N se encuentra la clase a la que pertenece dicho elemento. Este método de lectura de archivos .arff será común en la ejecución de cada algoritmo. Sin embargo, para el caso del archivo **WDBC** ha sido necesario implementar **un método de lectura distinto** que el de los archivos de Movement_Libras y Arrhythmia, ya que la disposición de los datos de wdbc.arff es distinta a las otras dos (la clase de cada elemento, en este caso B o M, aparece al principio de cada línea, y no al final).

5x2 Cross – Validation

Para las 10 ejecuciones de cada algoritmo, los conjuntos considerados (conjunto de prueba y conjunto de entrenamiento) son los obtenidos en el método de validación 5x2. Aunque dicho método es el mismo para todos los algoritmos, su implementación varía dependiendo de la estructura de nuestros datos; se diferencia principalmente en el número de clases que se estén considerando. A continuación se muestra el pseudocódigo de este método aplicado al WDBC, que contiene únicamente dos tipos de clases:

```
// 5 x 2 Cross - Validation para el problema WDBC

    datos // es nuestra matriz con los datos leídos de wdbc.arff
    vector claseM; // contendrá los índices de las filas cuya clase es M
    vector claseB; // contendrá los índices de las filas cuya clase es B
    vector posiciones // contendrá los índices de las filas que ya han sido asignadas
                      // al conjunto de prueba o al conjunto de entrenamiento
    int posicion;

    for (i = 0; i < datos.size(); i++) {
        if (datos[i][posicion_clase] == 'M') claseM < --i;
        else if (datos[i][posicion_clase] == 'B') claseB < --i;
    }
    // Se escogen aleatoriamente los datos que contendrá el conjunto de prueba
    for (i = 0; i < claseM.size() / 2; i++) {
        while (posicion ya esté en posiciones) {
            posicion = Rand(0, claseM.size() - 1);
        }
        prueba <-- datos[ claseM[posicion] ];
        posiciones <-- posicion;
    }

    // Se cogen los datos restantes para el conjunto de entrenamiento
    for (i = 0; i < claseM.size(); i++) {
        if (i no está en posiciones) entrenamiento <-- datos[ claseM[i] ];
    }

    // Se repite este proceso para los datos contenidos en claseB
```

Esquema de representación de soluciones y generación de solución inicial

En todos los algoritmos se considera como posible solución un vector binario de tamaño N (número de características) que representa qué características son tomadas para la clasificación. Las soluciones iniciales son aleatorias en un principio en cada uno de los algoritmos:

- En **BMB**: Se generan 25 soluciones iniciales aleatorias.
- En **GRASP**: Se generan 25 soluciones inciales aleatorias.
- En **ILS**: Se generan 25 soluciones iniciales, de las cuales la primera es aleatoria y las 24 restantes serán soluciones mutadas.

Pseudocódigo de la función objetivo y de la generación de vecino flip(s, i)

```
FuncionObjetivo {
        for (i = 0; i < N; i++) {
            clase = Clas3NN(prueba[i],matriz);
            if (clase == prueba[i][posicion_clase]) aciertos++;
        }
        tasa = 100 * (aciertos / prueba.size());
        return tasa;
}


Clas3NN(matriz conjunto, matriz conjunto_comparacion, vector s) {
        vector distancias; // quedarán almacenadas las 3 distancias más cortas
        distancias[0] = 9999.99;
        distancias[1] = 9999.99;
        distancias[2] = 9999.99;
        vector posiciones;

        if(conjunto == conjunto_comparacion){
            cs = Multiplica(conjunto, s); // se multiplica cada fila de la matriz
    conjunto por el vector s para obtener el conjunto con las características
    seleccionadas
            for(i=0; i<cs.size(); i++){
                for(k=0; k<cs.size(); k++){
                    if(i != k){
                        distancia = CalculaDistancias(cs[i], cs[k]);
                        if(distancia < distancias[2]) {
                             if(distancia < distancias[1]){
                                if(distancia < distancias[0]){
                                    distancias[2] = distancias[1];
                                    posiciones[2] = posiciones[1];
                                    distancias[1] = distancias[0];
                                    posiciones[1] = posiciones[0];
                                    distancias[0] = distancia;
                                    posiciones[0] = k;
                                }
                                else{
                                    distancias[2] = distancias[1];
                                    posiciones[2] = posiciones[1];
                                    distancias[1] = distancia;
```

```
                            posiciones[1] = k;
                        }
                    else{
                        distancias[2] = distancia;
                        posiciones[2] = k;
                    }
                }
            }
        } // fin del segundo bucle for

        // en distancias quedan almacenadas las 3 distancias más cortas
        // en posiciones quedan almacenadas las posiciones a esas 3
distancias
        POS_CLASE = cs[posiciones[0]].size()-1;
        if(cs[posiciones[0]][POS_CLASE] == cs[posiciones[1]][POS_CLASE] ||
cs[posiciones[0][POS_CLASE] == cs[posiciones[2]][POS_CLASE])
            clase = cs[posiciones[0]][POS_CLASE];

        else if(cs[posiciones[1]][POS_CLASE] == cs[posiciones[2]]
[POS_CLASE])
            clase = cs[posiciones[1]][POS_CLASE];
        else
            clase = cs[posiciones[0]][POS_CLASE];


        if(clase == cs[i][POS_CLASE]) aciertos++;

    } // fin del primer bucle for

    else{ // conjunto != conjunto_comparacion

        cs = Multiplica(conjunto, s); // se multiplica cada fila de la matriz
conjunto por el vector s para obtener el conjunto con las características
seleccionadas
        csc = Multiplica(conjunto_comparacion, s); // se multiplica cada fila
de la matriz conjunto_comparacion por el vector s para obtener el conjunto con
las características seleccionadas
        for(i=0; i<cs.size(); i++){
            for(k=0; k<csc.size(); k++){
                    distancia = CalculaDistancias(cs[i], csc[k]);
                    if(distancia < distancias[2]) {
                         if(distancia < distancias[1]){
                            if(distancia < distancias[0]){
                                distancias[2] = distancias[1];
                                posiciones[2] = posiciones[1];
                                distancias[1] = distancias[0];
                                posiciones[1] = posiciones[0];
                                distancias[0] = distancia;
                                posiciones[0] = k;
                            }
                            else{
                                distancias[2] = distancias[1];
                                posiciones[2] = posiciones[1];
                                distancias[1] = distancia;
                                posiciones[1] = k;
                            }
                        else{
                            distancias[2] = distancia;
```

```
                                posiciones[2] = k;
                            }
                        }
                    } // fin del segundo bucle for

                    // en distancias quedan almacenadas las 3 distancias más cortas
                    // en posiciones quedan almacenadas las posiciones a esas 3
        distancias
                    POS_CLASE = csc[posiciones[0]].size()-1;
                    if(csc[posiciones[0]][POS_CLASE] == csc[posiciones[1]][POS_CLASE]
        || csc[posiciones[0][POS_CLASE] == csc[posiciones[2]][POS_CLASE])
                        clase = csc[posiciones[0]][POS_CLASE];

                    else if(csc[posiciones[1]][POS_CLASE] == csc[posiciones[2]]
        [POS_CLASE])
                        clase = csc[posiciones[1]][POS_CLASE];
                    else
                        clase = csc[posiciones[0]][POS_CLASE];


                    if(clase == cs[i][POS_CLASE]) aciertos++;

                } // fin del primer bucle for

                 return aciertos;
            }
```

// Generación de vecino

```
    N = s.size() - 1;
    i → {0,1,2,...,N}
    matriz prueba; // matriz de datos
    vecino = flip(s, i);
    for ( j = 0; j < prueba.size(); j++) {
        for (int k = 0; k < vecino.size(); k++) {
            v.push_back(prueba[j][k] * vecino[k]);
        }
            v.push_back(prueba[j][vecino.size()]); // donde se encuentra la clase a la que
                                                   pertenece
        prueba_vecino.push_back(v);    // prueba_vecino es la prueba con las
                                           caracteristicas elegidas en vecino = flip(s,i)
        v.clear();
    }
```

Pseudocódigo del proceso de generación de soluciones aleatorias de BMB e ILS

```
// BMB
    s_aleatorias = 0;
    vector s; // Vector de selección
    matriz prueba; // Matriz de datos
    TAM = prueba[0].size();

    while(s_aleatorias < 25){
        s.clear();
        // Genero solución inicial
        for(i = 0; i < TAM; i++){
            r = Randint(0,1);
            s.push_back(r);
        }
        // Calculo del coste de la solución inicial
        ...
        // Aplicación del algoritmo de BL a la solución inicial
        ...
        // Comparación de la solución obtenida en la BL con la mejor hasta el momento
        ...
        s_aleatorias++;
    }


// ILS
    vector s; // Vector de selección
    matriz prueba; // Matriz de datos
    BL = 0;

    // Genero una solución inicial aleatoria. El resto de soluciones iniciales serán
sucesivas mutaciones
    for(i = 0; i < TAM; i++){
            r = Randint(0,1);
            s.push_back(r);
    }
    // Aplico BL sobre s
    ...
    BL++;

    while(BL < 25){

        // Muto s y lo guardo en s'
        ...
        // Aplico BL a s'
        ...
        BL++;
        // Comparo s y s' y me quedo con la mejor ( mejor{s,s'} → s )
        ...
        // Actualizo la mejor solución encontrada hasta el momento
        ...
    }
```

3 PSEUDOCÓDIGOS Y DESCRIPCIONES

3.1 PSEUDOCÓDIGO DEL MÉTODO DE BÚSQUEDA LOCAL

```
// BÚSQUEDA LOCAL
    datos = readfile();
    5x2cross-validation --> prueba y conjunto;

    s = s_inicial // solucion inicial generada de forma aleatoria
    mejor_tasa = CalculaCoste(s); // clas3nn(prueba, prueba, s)

    while (i < s.size()) {
        vecino = CalculaVecino(s);
        tasa_vecino = CalculaCoste(vecino); // clas3nn(prueba, prueba, vecino)

        if (tasa_vecino > mejor_tasa) {
            mejor_tasa = tasa_vecino;
            s = vecino;
            i = 0;
        }
        else i++;
    }
```

3.2 PSEUDOCÓDIGO DE BMB

```
// BMB
    datos = readfile();
    5x2cross-validation(datos) --> matrices prueba y conjunto;

    s_aleatorias = 0;
    vector s; // Vector de selección
    TAM = prueba[0].size();
    M = 0 // Mejor solución global

    while(s_aleatorias < 25){
        s.clear();
        // Genero solución inicial
        for(i = 0; i < TAM; i++){
            r = Randint(0,1);
            s.push_back(r);
        }
        // Calculo del coste de la solución inicial
        mejor_solucion = clas3nn(prueba, prueba, s);

        // Aplicación del algoritmo de BL a la solución inicial
        while (i < s.size()) {
            vecino = CalculaVecino(s); // flip(s,i)
            solucion = clas3nn(prueba, prueba, vecino);

            if (solucion > mejor_solucion){
```

```
            mejor_solucion = solucion;
            s = vecino;
            i = 0;
        }
        else i++;
    }
    // Comparación de la solución obtenida en la BL con la mejor hasta el momento
    if(mejor_solucion > M){
        M = mejor_solucion;
        mejor_vecino = s;
    }

    s_aleatorias++;
}
```

3.3 PSEUDOCÓDIGO DE GRASP

```
// GRASP
    datos = readfile();
    5x2cross-validation(datos) --> matrices prueba conjunto;
    M = 0 // Mejor solución global
    s_aleatorias = 0;
    vector s; // Vector de selección

    while(s_aleatorias < 25){
        // Generamos solución inicial mediante SFS aleatorizado (pseudocódigo de SFS
    aleatorizado desarrollado más abajo) (*)

        s ← SFS_aleatorizado;
        mejor_solucion = clas3nn(s);

        // Aplico BL sobre s
        while (i < s.size()) {
            vecino = CalculaVecino(s); // flip(s,i)
            solucion = clas3nn(prueba, prueba, vecino);

            if (solucion > mejor_solucion){
                mejor_solucion = solucion;
                s = vecino;
                i = 0;
            }
            else i++;
        }

        // Comparación de la solución obtenida en la BL con la mejor hasta el momento
        if(mejor_solucion > M){
            M = mejor_solucion;
            mejor_vecino = s;
        }

        s_aleatorias++;
    }
```

```
//(*) MECANISMO DE GENERACIÓN DE SOLUCIONES GREEDY PROBABILÍSTICAS
    hay_mejora = true;
    vector LC; // Lista de candidatos
    vector LRC; // Lista de candidatos restringida
    alpha = 0.3;

    while(hay_mejora){
        LC.clear();
        LRC.clear();
        cmejor = -99999;
        cpeor = 99999;

        // Construimos LC
        for(i = 0; i < s.size(); i++){
             if(s[i] == 0){
                s[i] = 1;
                aciertos = clas3nn(prueba, prueba, s);
                LC ← aciertos;
                if(aciertos > cmejor) cmejor = aciertos;
                else if(aciertos < cpeor) cpeor = aciertos;
                s[i] = 0;
            }
            else LC ← -1;
        }

        // Construimos LRC
        mu = cmejor – alpha * (cmejor – cpeor);
        for(i = 0; i < LC.size(); i++){
             if(LC[i] > mu && LC[i] != -1) LRC ← i;
        }

        // En LC quedan almacenados los aciertos que supone hacer s[i] = 1
            En caso de que en i ya haya un 1, se hace LC[i] = -1
        // En LRC quedan almacenadas las posiciones i de LC que hacen que LC[i] > mu

        r = Randint(0, LRC.size()-1);
        if(M > LC[LRC[rand]]) hay_mejora = false;
        else{
            M = LC[LRC[rand]];
            s[LRC[rand]] = 1;
        }
    }
```

3.4 Pseudocódigo de ILS

```
// ILS
    datos = readfile();
    5x2cross-validation(datos) --> matrices prueba conjunto;
    M = 0 // Mejor solución global
    s_aleatorias = 0;
    s // Vector de selección

    // Genero una solución inicial aleatoria. El resto de soluciones iniciales serán
sucesivas mutaciones
    for(i = 0; i < TAM; i++){
            r = Randint(0,1);
            s.push_back(r);
    }
    // Aplico BL sobre s
    while (i < s.size()) {
        vecino = CalculaVecino(s); // flip(s,i)
        solucion = clas3nn(prueba, prueba, vecino);

        if (solucion > mejor_solucion){
            mejor_solucion = solucion;
            s = vecino;
            i = 0;
        }
        else i++;
    }
    BL++;

    while(BL < 25){

        // Muto s y lo guardo en s'
        mutacion(s) → s'; //(*)Pseudocódigo del operador de mutación desarrollado más
    abajo
        // Aplico BL a s'
        while (i < s'.size()) {
            vecino = CalculaVecino(s'); // flip(s',i)
            solucion = clas3nn(prueba, prueba, vecino);

            if (solucion > mejor_solucion){
                mejor_solucion = solucion;
                s' = vecino;
                i = 0;
            }
            else i++;
        }
        BL++;

        // Comparo s y s' y me quedo con la mejor ( mejor{s,s'} → s )
        clasS0 = clas3nn(prueba, prueba, s);
        clasS1 = clas3nn(prueba, prueba, s');

        if(clasS0 > clasS1){
            mejor_s = s;
            mejor_solucion = clasS0;
        }

        else{
```

```
            mejor_s = s';
            mejor_solucion = clasS1;
        }

        // Comparo y actualizo la mejor solución encontrada hasta el momento
        if(mejor_solucion > M){
            M = mejor_solucion;
            mejor_vecino = mejor_s;
        }
        s = s_mejor;
        s'.clear();
    }

// (*) OPERADOR DE MUTACIÓN
    vector aleatorios;
    t = 0.1 * s.size();
    mutaciones = 0;

    aleatorios.clear();
    while (mutaciones < t) {
        while (r esté contenido en aleatorios) {
            r = Randint(0, s.size()-1);
        } // Comprobamos que la posición escogida aleatoriamente para realizar la
mutación no esté repetida
        aleatorios ← r;
        s' = flip(s,r);
        mutaciones++;
    }
```

4 DESCRIPCIÓN DEL ALGORITMO DE COMPARACIÓN (SFS)

A continuación se muestra el pseudocódigo del algoritmo de comparación:

```
// SFS
    hay_mejora = true;
    tasa_anterior = 0;
    mejor_tasa = 0;

    while (hay_mejora) {
        for (i = 0; i < s.size(); i++) {
            if (s[i] != 1) {
                s[i] = 1;
                tasa = CalculaCoste(s);

                if (tasa > mejor_tasa) {
                    mejor_tasa = tasa;
                    mejor_s = s;
                }
                s[i] = 0;
            }
        }
        s = mejor_s;
        if (tasa_anterior >= mejor_tasa) hay_mejora = false;
        else tasa_anterior = mejor_tasa;
    }
```

5 PROCEDIMIENTO CONSIDERADO PARA DESARROLLAR LA PRÁCTICA

La implementación del código ha sido iniciada desde cero (no se ha usado el código proporcionado en la plataforma de la asignatura ni ningún otro). Tampoco se ha hecho uso de ningún *framework*. Tan solo se ha hecho uso del generador aleatorio proporcionado en la web de la asignatura.

6 EXPERIMENTOS Y ANÁLISIS DE RESULTADOS

6.1 CASOS DEL PROBLEMA Y VALORES DE PARÁMETROS CONSIDERADOS

- **Semillas consideradas:**

Para la primera partición del conjunto de datos se considera una semilla con valor **47**.
Para la segunda partición del conjunto de datos se considera una semilla con valor **18**.
Para la tercera partición del conjunto de datos se considera una semilla con valor **95**.
Para la cuarta partición del conjunto de datos se considera una semilla con valor **6**.
Para la quinta partición del conjunto de datos se considera una semilla con valor **33**.

Estas semillas son las mismas para las distintas ejecuciones de todos los algoritmos.

6.2 RESULTADOS OBTENIDOS: TABLAS

Tabla de resultados obtenidos por el algoritmo SFS para el problema de SC

	WDBC			*MOVEMENT_LIBRAS*			*ARRHYTHMIA*		
	%CLAS	**%RED**	**T (seg)**	**%CLAS**	**%RED**	**T(seg)**	**%CLAS**	**%RED**	**T(seg)**
1 – 1	95'7895	83'3333	2'849	71'6667	91'1111	6'832	75'7732	95'6835	19'604
1 – 2	94'3662	93'3333	1'248	72'2222	90'0000	7'251	68'2292	97'8417	12'501
2 – 1	94'386	86'6667	1'89	72'7778	92'2222	6'137	77'3169	97'1223	18'965
2 – 2	93'4958	90'6667	2'075	66'6667	92'2222	4'903	69'7917	97'482	14'747
3 – 1	94'3667	83'3333	2'901	65'6667	93'3334	5'0871	75'8993	95'6667	20'102
3 – 2	92'7778	94'4443	1'907	71'6667	90'0000	7'005	69'7726	98'1223	15'2245
4 – 1	95'0333	83'3333	2'778	74'2222	87'7778	4'9728	77'2849	97'201	16'0961
4 – 2	93'7778	86'6667	1'08	63'3333	91'1111	10'221	66'998	98'8411	10'334
5 – 1	92'9825	93'3333	1'065	74'4444	87'7778	10'767	69'0722	98'2014	10'674
5 – 2	94'7183	83'3333	2'248	63'3333	87'7778	9'339	73'9583	97'482	15'452
MEDIA	**94'5218**	**87'3333**	**1'701**	**69'6667**	**90'3334**	**7'1289**	**72'3726**	**97'4667**	**15'5927**

Tabla de resultados obtenidos por el algoritmo del Clasificador 3-NN para el problema de SC

	WDBC			*MOVEMENT_LIBRAS*			*ARRHYTHMIA*		
	%CLAS	**%RED**	**T (seg)**	**%CLAS**	**%RED**	**T(seg)**	**%CLAS**	**%RED**	**T(seg)**
1 – 1	95'7895	X	0'078	76'1111	X	0'071	67'5258	X	0'268
1 – 2	96'1268	X	0'041	73'8889	X	0'085	61'9792	X	0'255
2 – 1	95'0877	X	0'128	76'1111	X	0'076	65'4639	X	0'337
2 – 2	96'831	X	0'181	67'7778	X	0'079	65'625	X	0'235
3 – 1	97'386	X	0'102	77'1111	X	0'091	61'9792	X	0'301
3 – 2	96'1267	X	0'069	73'7778	X	0'082	65'625	X	0'276
4 – 1	94'0877	X	0'061	67'8889	X	0'07	64'3882	X	0'227
4 – 2	96'7895	X	0'083	76'1111	X	0'071	66'2441	X	0'316
5 – 1	94'386	X	0'078	73'8889	X	0'083	64'433	X	0'321
5 – 2	97'8873	X	0'093	73'8889	X	0'071	65'625	X	0'237
MEDIA	**96'3887**	**X**	**0'091**	**73'7778**	**X**	**0'0779**	**66'2258**	**X**	**0'2773**

Tabla de resultados obtenidos en BMB para el problema de SC

	WDBC			*MOVEMENT_LIBRAS*			*ARRHYTHMIA*		
	%CLAS	**%RED**	**T (seg)**	**%CLAS**	**%RED**	**T(seg)**	**%CLAS**	**%RED**	**T(seg)**
1 – 1	96.8310	45.1613	64'293	76'1111	43'956	319'142	66'1458	48'3871	1656'64
1 – 2	93.3333	61.2903	55'831	70'0000	43'956	147'376	64'3221	48'8992	1644'6
2 – 1	96.1268	41.9355	51'714	75'0000	46'1538	209'883	65'625	49'8208	1677'63
2 – 2	94.7368	48.3871	48'218	70'0000	52'7473	231'704	64'101	48'8891	1665'44
3 – 1	97.5352	64.5161	61'08	76'1111	50'5494	192'761	66'1458	48'3871	1569'5
3 – 2	92.9825	51.6129	56'487	76'1111	43'956	137'872	65'9331	47'773	1591'23
4 – 1	97.1831	51.6129	57'554	75'5556	52'7473	169'689	64'0625	46'2366	1387'96
4 – 2	95.0877	48.3871	50'9110	68'8889	47'2527	201'276	62'1955	46'0133	1357'7
5 – 1	94.7183	51.6129	52'5680	75'5556	57'1429	198'083	65'625	47'3118	1675'23
5 – 2	97.1930	45.1613	59'4230	75'5556	49'4506	186'136	64'9485	46'9543	1655'86
MEDIA	**95'5728**	**50'9677**	**55'8079**	**73'8889**	**48`7912**	**199'392**	**64'9104**	**47'8672**	**1588'17**

Tabla de resultados obtenidos en GRASP para el problema de SC

	WDBC			*MOVEMENT_LIBRAS*			*ARRHYTHMIA*		
	%CLAS	**%RED**	**T (seg)**	**%CLAS**	**%RED**	**T(seg)**	**%CLAS**	**%RED**	**T(seg)**
1 – 1	94'7183	66'6667	90'685	72'7778	81'1111	149'385			
1 – 2	94'7368	50'0000	98'517	68'3333	88'8889	173'154			
2 – 1	94'7183	73'3333	71'969	72'2222	85'5556	149'557			
2 – 2	95'7895	70'0000	81'129	75'0000	85'5556	138'467			
3 – 1	96'4789	56'6667	91'796	68'3333	85'5556	203'694			
3 – 2	95'4386	23'3333	78'698	72'7778	88'8889	146'828			
4 – 1	96'4789	70'0000	95'556	71'1111	86'6667	140'345			
4 – 2	94'386	50'0000	144'198	73'8889	85'5556	168'823			
5 – 1	95'0704	60'0000	99'881	75'0000	87'7778	197'652			
5 – 2	96'4912	56'6667	56'568	76'1111	85'5556	134'949	72'1649	93'8849	1331.41
MEDIA	**95'4307**	**57'6667**	**98'556**	**72'779**	**85'556**	**165'863**			

Tabla de resultados obtenidos en ILS para el problema de SC

	WDBC			*MOVEMENT_LIBRAS*			*ARRHYTHMIA*		
	%CLAS	**%RED**	**T (seg)**	**%CLAS**	**%RED**	**T(seg)**	**%CLAS**	**%RED**	**T(seg)**
1 – 1	97'5352	53'3333	39'902	67'8891	51'6667	145'332			
1 – 2	95'4386	43'3333	44'639	66'6667	47'3333	133'785			
2 – 1	95'4225	46'6667	37'712	76'1111	47'7778	137'381			
2 – 2	96'8421	46'6667	45'468	66'6667	45'5556	143'364			
3 – 1	97'1831	33'3333	44'255	73'1111	50'0000	146'667			
3 – 2	93'3333	33'3333	44'978	75'3343	44'3333	133'789			
4 – 1	97'8873	46'6667	38'988	76'6667	51'6667	141'374			
4 – 2	94'7368	50'0000	40'423	74'4225	47'7778	138'53			
5 – 1	95'4225	33'3333	45'949	75'6678	50'6667	147'837			
5 – 2	97'8947	46'6667	38'206	76'1111	45'3333	130'778			
MEDIA	**96'6524**	**43'3333**	**39'403**	**71'3343**	**48'7778**	**138'667**			

6.3 ANÁLISIS DE RESULTADOS

A la vista de los resultados obtenidos en las ejecuciones, en cuanto al **Método de Multiarranque Básico (BMB)**, queda en evidencia una de las desventajas que conocemos de dicho algoritmo: es **ineficiente**. Esto se debe a que inicialmente, las soluciones iniciales aleatorias de las que partimos están relativamente "cerca" unas de otras y, al aplicar la Búsqueda Local, se obtiene varias veces un mismo óptimo local.

En cuanto a las **tasas de clasificación,** son un poco mejores que las obtenidas con **GRASP** pero peores que las obtenidas con **ILS.** Sin embargo cabe observar que las **tasas de reducción** del BMB son muy bajas, y observamos que, en concreto, la diferencia entre las tasas de clasificación de BMB y GRASP no es tan significativa como la diferencia entre las tasas de reducción de estos mismos algoritmos: podemos concluir que de forma general **la capacidad del algoritmo GRASP para obtener soluciones de calidad es mejor que la de BMB**, en virtud de las altas tasas de reducción de GRASP.

Hemos de tener en cuenta que **el algoritmo BMB converge al óptimo global cuando el número de puntos generados tiende a infinito**, y nosotros tan solo hemos generado 25 puntos. Quizás, de haber generado más puntos, los resultados obtenidos con BMB habrían sido más satisfactorios (aunque eso supusiese una carga extra de tiempo de ejecución).

Otra comparación relevante que podemos hacer es la siguiente: dado que el algoritmo **GRASP** mejora sus posibilidades de encontrar un buen óptimo a la hora de realizar Búsqueda Local mediante la generación de diversidad en su primera etapa (**SFS aleatorizado**), conviene hacer hincapié entre los resultados obtenidos por el algoritmo **SFS y GRASP**. Como cabe esperar, GRASP resulta mejor que SFS en todos los ámbitos ya que, aunque SFS produce soluciones que pueden parecer razonables, no llegan a ser todo lo buenas que podrían ser principalmente por la **falta de diversidad**. El uso de SFS aleatorizado en GRASP asegura una rica diversidad que se ve reflejada en la mejora de resultados.

La diversidad es una ventaja que no solo aporta el GRASP, sino que también se ve reflejada en el método **ILS**. Si comparamos GRAPS e ILS podemos observar que las **tasas de clasificación** son mejores en ILS que en GRASP. Esto puede deberse a que, aunque ambos métodos introducen formas de proporcionar diversidad a la búsqueda de una solución, dichas formas son distintas: GRASP introduce aleatoridad escogiendo una solución inicial **aleatoria** (a la que posteriormente aplicará Búsqueda Local) **de entre un conjunto de mejores soluciones**. Sin embargo, podríamos decir que la forma de diversificar de ILS está más "controlada"; lo que hace ILS es **mutar un % de una solución**, pero no de cualquier solución, sino de las sucesivas soluciones que se van obteniendo; es decir, conforme avance la ejecución, dicha solución será **cada vez mejor**, y por tanto de cada diversificación se irán obteniendo soluciones mejores. Sin embargo, en contra de esto, las soluciones de ILS serán más "complejas": como podemos observar en las tablas, la tasa de reducción de ILS, por lo general, es notablemente más baja que la de GRASP.

Podemos concluir sospechando que, en una supuesta situación, realmente no tiene sentido decir que uno de estos algoritmos es mejor que otro de forma generalizada: conviene identificar y estudiar el problema que se nos pueda plantear y, en función de eso, escoger el método que más se adapte a las necesidades requeridas.