Escuela Técnica Superior de Ingeniería Informática
Universidad de La Laguna
Grado en Ingeniería Informática

Diseño y Análisis de Algoritmos.

Práctica 4.
Algoritmos constructivos y búsquedas por entornos

Rubén Labrador Páez
alu0100309553@ull.edu.es

Índice

Descripción de los algoritmos.

Algoritmo Constructivo 1:

El algoritmo voraz diseñado es el propuesto en el guión de la práctica.

1: Seleccionar la arista (i, j) con mayor afinidad;
2: S = {i, j};
3: repeat
4: S*= S;
5: Obtener el vértice k que maximiza md(S ∪ {k});
6: if md(S ∪ {k}) ≥ md(S) then
7: S = S ∪ {k};
8: until (S* = S)
9: Devolver S* ;

La implementación cuenta con los siguientes métodos:

- Constructor: Se encarga de inicializar el problema y el objeto Eval empleado para analizar las soluciones, además ejecuta el algoritmo.
- Método init(): se encarga de inicializar el problema, buscando los dos nodos con mayor afinidad.
- Método mejornodo(): devuelve el mejor nodo a añadir a la solución de entre los que no pertenecen a la misma.
- Método toString(): Devuelve un string con la solución.
- Método fObj(): Devuelve el valor de la función objetivo.

Las estructuras de datos empleadas han sido las siguientes:

- Clase Solución: Clase empleada para almacenar soluciones y que tiene métodos para modificar las mismas, las soluciones se almacenan como un array booleano.
- Clase Eval: Esta clase se emplea para evaluar soluciones, devuelve el valor de la función objetivo dada una solución, tiene métodos para calcular la función objetivo sólo teniendo en cuenta una solución, así como añadiendo o quitando el nodo indicado.

Algoritmo Constructivo 2:

El algoritmo voraz diseñado parte de una solución que incluye todos los nodos y en cada se busca el nodo que eliminandolo maximiza la función objetivo.

1: Añadir todos los nodos a la solución;
2: S = {1, 2, … ,n};
3: repeat
4: S * = S;
5: Obtener el vértice k que maximiza md(S - {k});
6: if md(S - {k}) ≥ md(S) then
7: S = S ∪ {k};
8: until (S * = S)
9: Devolver S * ;

La implementación cuenta con los siguientes métodos:

- Constructor: Se encarga de inicializar el problema y el objeto Eval empleado para analizar las soluciones, además ejecuta el algoritmo.
- Método init(): Añade todos los nodos a la solución de partida.
- Método peornodo(): devuelve el nodo a sustraer de la solución actual.
- Método toString(): Devuelve un string con la solución.
- Método fObj(): Devuelve el valor de la función objetivo.

Las estructuras de datos empleadas han sido las siguientes:

- Clase Solución: Clase empleada para almacenar soluciones y que tiene métodos para modificar las mismas, las soluciones se almacenan como un array booleano.
- Clase Eval: Esta clase se emplea para evaluar soluciones, devuelve el valor de la función objetivo dada una solución, tiene métodos para calcular la función objetivo sólo teniendo en cuenta una solución, así como añadiendo o quitando el nodo indicado.

Algoritmo GRASP:

El algoritmo GRASP propuesto para esta solución consiste en: partiendo de una solución Greedy Aleatoria, realizar una búsqueda local sobre la misma, el algoritmo se ejecuta mientras no haya habido 10 ciclos sin encontrar mejora.

```
1: Generar una solución Greedy aleatoria sobre una lista de candidatos.
2: S = randomGreedy().
3: Repeat:
4:      S*  = randomGreedy().;
5:      S* =  busquedaLocal (S*);
6:      if md(S*) ≥ md(S) then
7:              S = S*;
8:              contador = 0;
9:      else
10:             contador++;
11: until (contador < 10);
12: Devolver S* ;
```

La implementación cuenta con los siguientes métodos:

- Constructor: Se encarga de inicializar el problema y el objeto Eval empleado para analizar las soluciones, además ejecuta el algoritmo.
- Método randomGreedy(): Genera una solución Greedy aleatoria, eligiendo el mejor nodo de entre una lista de candidatos, hace uso de randominit() y randomMejornodo().
- Método busquedaLocal(): Realiza una búsqueda en el entorno de la solución que se le pasa como parámetro, devuelve el mejor de los vecinos, hace uso del método generarVecinos().
- Método toString(): Devuelve un string con la solución.
- Método fObj(): Devuelve el valor de la función objetivo.

Las estructuras de datos empleadas han sido las siguientes:

- Clase Solución: Clase empleada para almacenar soluciones y que tiene métodos para modificar las mismas, las soluciones se almacenan como un array booleano.
- Clase Eval: Esta clase se emplea para evaluar soluciones, devuelve el valor de la función objetivo dada una solución, tiene métodos para calcular la función objetivo sólo teniendo en cuenta una solución, así como añadiendo o quitando el nodo indicado.
- Clase Nodo: Clase que almacena un nodo asociado a un valor de función objetivo, esta clase implementa el método compareTo, que junto a un arraylist, nos permite almacenar posibles nodos a añadir a la solución para posteriormente ordenarlos y poder elegir una de las mejores soluciones para el método randomGreedy().
- Clase Distancia: Clase que almacena una arista asociada a dos nodos, al igual que en la clase nodo, se implementa el método compareTo, que junto a un arraylist, nos permite ordenar las arista de mayor a menor para poder seleccionar, de manera aleatoria, una de las mejores, esta estructura es utilizada por el método randomGreedy().

Algoritmo Multi-Arranque:

El Algoritmo Multi-Arranque implementado consiste en generar una solución aleatoria y sobre esta realizar una búsqueda local, este proceso se repite tantas veces como nodos tenga el problema, siempre se almacena la mejor solución encontrada hasta el momento para poder compararla con las sucesivas que se van explorando.

```
1: Generar una solución aleatoria al problema.
2: S = randomSolución().
3: for ( i = 0 ;i < nº de nodos; i++)
4:        S*  = randomSolucion().;
5:        S* =  busquedaLocal (S*);
6:        if md(S*) ≥ md(S) then
7:                  S = S*;
8:                  contador = 0;
9:
10: rof ;
11: Devolver S* ;
```

La implementación cuenta con los siguientes métodos:

- Constructor: Se encarga de inicializar el problema y el objeto Eval empleado para analizar las soluciones, además ejecuta el algoritmo.
- Método randomSolucion(): Genera una solución aleatoria al problema, añadiendo o quitando nodos de manera aleatoria.
- Método busquedaLocal(): Realiza una búsqueda en el entorno de la solución que se le pasa como parámetro, devuelve el mejor de los vecinos, hace uso del método generarVecinos().
- Método toString(): Devuelve un string con la solución.
- Método fObj(): Devuelve el valor de la función objetivo.

Las estructuras de datos empleadas han sido las siguientes:

- Clase Solución: Clase empleada para almacenar soluciones y que tiene métodos para modificar las mismas, las soluciones se almacenan como un array booleano.
- Clase Eval: Esta clase se emplea para evaluar soluciones, devuelve el valor de la función objetivo dada una solución, tiene métodos para calcular la función objetivo sólo teniendo en cuenta una solución, así como añadiendo o quitando el nodo indicado.

Algoritmo VNS:

El algoritmo VNS implementado consiste en: partiendo de una solución, realizar búsqueda local sobre la misma para intentar mejorarla, cuando mediante búsqueda local ya no se consigue mejorar dicha solución, se cambia de entorno un número k de veces previamente establecido, una vez se ha encontrado la mejor solución para todos los entornos, se vuelve a ejecutar el algoritmo sobre la nueva solución, esto se realiza mientras la solución mejore.

Para no explorar toda la lista de vecinos, se emplea el método agitación, que devuelve, de manera aleatoria, uno de los vecinos en el entorno indicado a la solución propuesta.

```
1: Generar una solución aleatoria al problema.
2: S = randomSolución().
3: S* = S;
4: Repeat;
5:      S  = S*;
6:      k = 1;
7:      Repeat;
8:           S** =  agitacion( k ,S*);
9:           S** =  busquedaLocal(S**);
10:          if md(S**) ≥ md(S*) then
11:               S *= S**;
12:          else
13:               k++;
9:      until (k <= kmax)
10: until (exista mejora);
11: Devolver S* ;
```

La implementación cuenta con los siguientes métodos:

- Constructor: Se encarga de inicializar el problema y el objeto Eval empleado para analizar las soluciones, además ejecuta el algoritmo.
- Método randomSolucion(): Genera una solución aleatoria al problema, añadiendo o quitando nodos de manera aleatoria.
- Método busquedaLocal(): Realiza una búsqueda en el entorno de la solución que se le pasa como parámetro, devuelve el mejor de los vecinos, hace uso del método generarVecinos().

- Método agitacion(): devuelve de manera aleatoria, y en función del valor k que se le pasa como parámetro un vecino a la solución que se le pasa como parámetro, el valor de k define el entorno en el que se realiza la búsqueda.
- Método toString(): Devuelve un string con la solución.
- Método fObj(): Devuelve el valor de la función objetivo.

Las estructuras de datos empleadas han sido las siguientes:

- Clase Solución: Clase empleada para almacenar soluciones y que tiene métodos para modificar las mismas, las soluciones se almacenan como un array booleano.
- Clase Eval: Esta clase se emplea para evaluar soluciones, devuelve el valor de la función objetivo dada una solución, tiene métodos para calcular la función objetivo sólo teniendo en cuenta una solución, así como añadiendo o quitando el nodo indicado.

Análisis de algoritmos:

Algoritmo Constructivo 1:

Problema	N	Ejecución	Valor	Tiempo CPU (ms)	Tiempo CPU medio (ms)	Valor/ Mejor Valor	Min Valor/Mejor Valor
1	10	1	24,74	8		0,5641961231	
1	10	2	24,74	12		0,5641961231	
1	10	3	24,74	9		0,5641961231	
1	10	4	24,74	8		0,5641961231	
1	10	5	24,74	9	9,2	0,5641961231	0,5641961231
2	20	1	63,03	13		0,9364136087	
2	20	2	63,03	14		0,9364136087	
2	20	3	63,03	11		0,9364136087	
2	20	4	63,03	11		0,9364136087	
2	20	5	63,03	13	12,4	0,9364136087	0,9364136087
3	30	1	80,72	23		0,7978649797	
3	30	2	80,72	29		0,7978649797	
3	30	3	80,72	27		0,7978649797	
3	30	4	80,72	24		0,7978649797	
3	30	5	80,72	24	25,4	0,7978649797	0,7978649797
4	40	1	94,9	43		0,8161334709	
4	40	2	94,9	33		0,8161334709	
4	40	3	94,9	34		0,8161334709	
4	40	4	94,9	32		0,8161334709	
4	40	5	94,9	38	36	0,8161334709	0,8161334709
5	50	1	103,76	55		0,9188806235	
5	50	2	103,76	47		0,9188806235	
5	50	3	103,76	53		0,9188806235	
5	50	4	103,76	53		0,9188806235	
5	50	5	103,76	52	52	0,9188806235	0,9188806235
6	100	1	175,14	122		0,8846794969	
6	100	2	175,14	100		0,8846794969	
6	100	3	175,14	128		0,8846794969	
6	100	4	175,14	111		0,8846794969	
6	100	5	175,14	144	121	0,8846794969	0,8846794969

Algoritmo Constructivo 2:

Problema	N	Ejecución	Valor	Tiempo CPU (ms)	Tiempo CPU medio (ms)	Valor/ Mejor Valor	Min Valor/Mejor Valor
1	10	1	43,85	8		1	
1	10	2	43,85	8		1	

1	10	3	43,85	9		1	
1	10	4	43,85	8		1	
1	10	5	43,85	9	8,4	1	1
2	20	1	65,17	14		0,9682068043	
2	20	2	65,17	16		0,9682068043	
2	20	3	65,17	13		0,9682068043	
2	20	4	65,17	12		0,9682068043	
2	20	5	65,17	13	13,6	0,9682068043	0,9682068043
3	30	1	100,7	31		0,9953543541	
3	30	2	100,7	27		0,9953543541	
3	30	3	100,7	31		0,9953543541	
3	30	4	100,7	30		0,9953543541	
3	30	5	100,7	29	29,6	0,9953543541	0,9953543541
4	40	1	112,13	48		0,9643102855	
4	40	2	112,13	51		0,9643102855	
4	40	3	112,13	47		0,9643102855	
4	40	4	112,13	50		0,9643102855	
4	40	5	112,13	49	49	0,9643102855	0,9643102855
5	50	1	105,74	76		0,9364151612	
5	50	2	105,74	71		0,9364151612	
5	50	3	105,74	66		0,9364151612	
5	50	4	105,74	76		0,9364151612	
5	50	5	105,74	74	72,6	0,9364151612	0,9364151612
6	100	1	195,11	155		0,9855533667	
6	100	2	195,11	139		0,9855533667	
6	100	3	195,11	171		0,9855533667	
6	100	4	195,11	143		0,9855533667	
6	100	5	195,11	156	152,8	0,9855533667	0,9855533667

GRASP:

Problema	N	LRC	Ejecución	Valor	Tiempo CPU (ms)	Tiempo CPU medio (ms)	Valor/ Mejor Valor	Min Valor/Mejor Valor
1	10	2	1	43,85	17		1	
1	10	2	2	43,85	14		1	
1	10	2	3	43,85	18		1	
1	10	2	4	43,85	13		1	
1	10	2	5	43,85	15	15,4	1	1
1	10	3	1	43,85	14		1	
1	10	3	2	43,85	15		1	
1	10	3	3	43,85	14		1	
1	10	3	4	43,85	13		1	
1	10	3	5	43,85	16	14,4	1	1
2	20	2	1	65,27	31		0,9696924677	
2	20	2	2	65,27	26		0,9696924677	
2	20	2	3	65,27	28		0,9696924677	
2	20	2	4	65,27	34		0,9696924677	
2	20	2	5	65,27	33	30,4	0,9696924677	0,9696924677
2	20	3	1	67,31	30		1	
2	20	3	2	65,27	33		0,9696924677	
2	20	3	3	65,27	29		0,9696924677	
2	20	3	4	65,27	35		0,9696924677	
2	20	3	5	65,27	33	32	0,9696924677	0,9696924677
3	30	2	1	94,95	44		0,9385193239	
3	30	2	2	94,95	47		0,9385193239	
3	30	2	3	94,95	57		0,9385193239	

3	30	2	4	101,17	56		1	
3	30	2	5	94,95	58	52,4	0,9385193239	0,9385193239
3	30	3	1	101,17	74		1	
3	30	3	2	101,17	52		1	
3	30	3	3	101,17	66		1	
3	30	3	4	101,17	51		1	
3	30	3	5	101,17	50	58,6	1	1
4	40	2	1	116,28	104		1	
4	40	2	2	116,28	71		1	
4	40	2	3	116,28	80		1	
4	40	2	4	116,28	87		1	
4	40	2	5	116,28	85	85,4	1	1
4	40	3	1	116,28	78		1	
4	40	3	2	116,28	83		1	
4	40	3	3	116,28	93		1	
4	40	3	4	116,28	102		1	
4	40	3	5	116,28	72	85,6	1	1
5	50	2	1	112,92	130		1	
5	50	2	2	112,92	106		1	
5	50	2	3	112,92	104		1	
5	50	2	4	112,92	101		1	
5	50	2	5	112,92	109	110	1	1
5	50	3	1	112,92	115		1	
5	50	3	2	112,92	138		1	
5	50	3	3	112,92	132		1	
5	50	3	4	112,92	116		1	
5	50	3	5	112,92	114	123	1	1
6	100	2	1	197,97	408		1	
6	100	2	2	193,13	387		0,9755518513	
6	100	2	3	193,13	476		0,9755518513	
6	100	2	4	193,13	376		0,9755518513	
6	100	2	5	193,13	395	408,4	0,9755518513	0,9755518513
6	100	3	1	193,13	374		0,9755518513	
6	100	3	2	193,13	435		0,9755518513	
6	100	3	3	193,13	447		0,9755518513	
6	100	3	4	193,13	315		0,9755518513	
6	100	3	5	193,13	395	393,2	0,9755518513	0,9755518513

Multi-Arranque:

Problema	N	Ejecución	Valor	Tiempo CPU (ms)	Tiempo CPU medio (ms)	Valor/ Mejor Valor	Min Valor/Mejor Valor
1	10	1	43,85	16		1	
1	10	2	43,85	12		1	
1	10	3	43,85	22		1	
1	10	4	43,85	11		1	
1	10	5	43,85	12	14,6	1	1
2	20	1	67,31	29		1	
2	20	2	67,31	29		1	
2	20	3	67,31	31		1	
2	20	4	67,31	28		1	
2	20	5	67,31	35	30,4	1	1
3	30	1	101,17	68		1	
3	30	2	101,17	59		1	
3	30	3	101,17	60		1	
3	30	4	101,17	64		1	
3	30	5	101,17	67	63,6	1	1
4	40	1	116,28	119		1	
4	40	2	116,28	127		1	
4	40	3	116,28	119		1	
4	40	4	116,28	107		1	

4	40	5	116,28	134	121,2	1	1
5	50	1	112,92	212		1	
5	50	2	112,92	210		1	
5	50	3	112,92	193		1	
5	50	4	112,92	205		1	
5	50	5	112,92	216	207,2	1	1
6	100	1	197,97	5527		1	
6	100	2	197,97	5205		1	
6	100	3	197,97	5271		1	
6	100	4	197,97	5154		1	
6	100	5	197,97	5120	5255,4	1	1

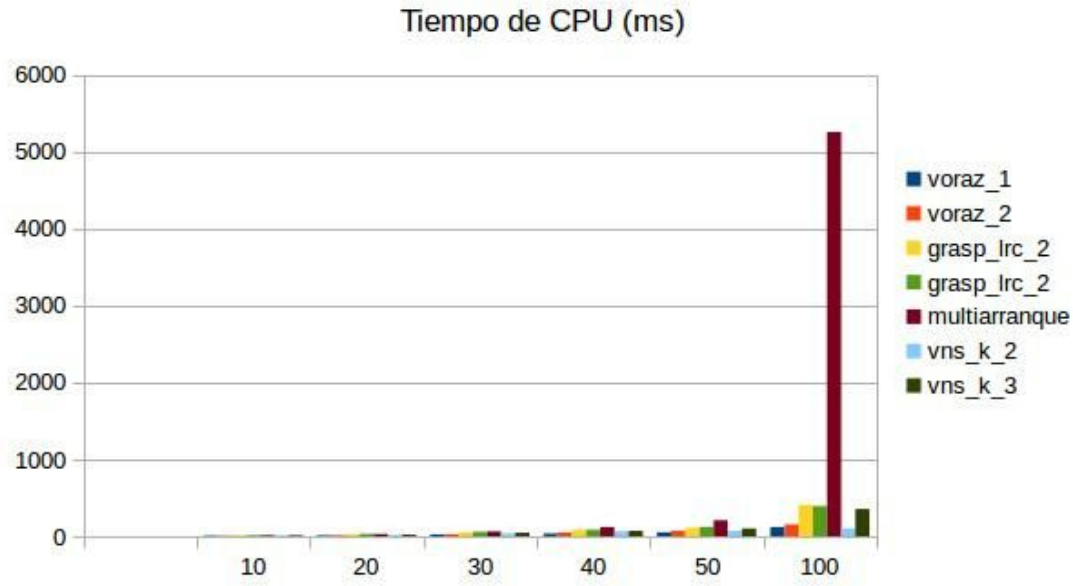
VNS:

Problema	N	Kmax	Ejecución	Valor	Tiempo CPU (ms)	Tiempo CPU medio (ms)	Valor/ Mejor Valor	Min Valor/Mejor Valor
1	10	2	1	43,85	14		1	
1	10	2	2	43,85	14		1	
1	10	2	3	43,85	12		1	
1	10	2	4	43,85	13		1	
1	10	2	5	43,85	15	13,6	1	1
1	10	3	1	43,85	13		1	
1	10	3	2	43,85	14		1	
1	10	3	3	43,85	16		1	
1	10	3	4	43,85	12		1	
1	10	3	5	43,85	14	13,8	1	1
2	20	2	1	67,31	17		1	
2	20	2	2	66,01	18		0,9806863765	
2	20	2	3	66,01	19		0,9806863765	
2	20	2	4	65,9	20		0,9790521468	
2	20	2	5	67,31	24	19,6	1	0,9790521468
2	20	3	1	65,9	19		0,9790521468	
2	20	3	2	65,9	26		0,9790521468	
2	20	3	3	66,01	18		0,9806863765	
2	20	3	4	67,31	28		1	
2	20	3	5	66,01	19	22	0,9806863765	0,9790521468
3	30	2	1	100,7	37		0,9953543541	
3	30	2	2	100,7	30		0,9953543541	
3	30	2	3	100,7	42		0,9953543541	
3	30	2	4	99,25	34		0,9810220421	
3	30	2	5	101,17	49	38,4	1	0,9810220421
3	30	3	1	101,17	53		1	
3	30	3	2	101,17	52		1	
3	30	3	3	100,7	45		0,9953543541	
3	30	3	4	100,7	39		0,9953543541	
3	30	3	5	101,17	48	47,4	1	0,9953543541
4	40	2	1	116,28	68		1	
4	40	2	2	116,28	61		1	
4	40	2	3	116,28	64		1	
4	40	2	4	112,13	60		0,9643102855	
4	40	2	5	116,28	68	64,2	1	0,9643102855
4	40	3	1	116,28	78		1	
4	40	3	2	116,28	65		1	
4	40	3	3	116,28	67		1	
4	40	3	4	116,28	80		1	
4	40	3	5	112,13	63	70,6	0,9643102855	0,9643102855
5	50	2	1	105,27	81		0,9322529224	
5	50	2	2	111,47	100		0,9871590507	
5	50	2	3	104	84		0,921006022	
5	50	2	4	112,92	84		1	
5	50	2	5	109,3	96	89	0,9679419058	0,921006022
5	50	3	1	111,21	96		0,9848565356	

5	50	3	2	112,74	103		0,9984059511	
5	50	3	3	112,92	102		1	
5	50	3	4	106,39	100		0,9421714488	
5	50	3	5	105,27	109	102	0,9322529224	0,9322529224
6	100	2	1	195,36	310		0,9868161843	
6	100	2	2	195,11	246		0,9855533667	
6	100	2	3	194,73	240		0,9836338839	
6	100	2	4	195,48	260		0,9874223367	
6	100	2	5	190,29	262	263,6	0,9612062434	0,9612062434
6	100	3	1	196,65	350		0,9933323231	
6	100	3	2	195,36	419		0,9868161843	
6	100	3	3	180,9	325		0,9137748144	
6	100	3	4	194,35	354		0,9817144012	
6	100	3	5	197,97	331	355,8	1	0,9137748144

Gráficas:

Tiempo de ejecución medio para cada algoritmo (ms) / (número de nodos)



Aproximación al mejor valor encontrado del problema para cada algoritmo.