Búsqueda Local Metaheurísticos

Daniela Niño

Descripción de Algoritmos

VND

El algoritmo VND funciona así:

1. Toma una solución inicial como la mejor solución

En este caso, utiliza la del método constructivo

2. Se definen el número de vecindarios que van a ser tomados en cuenta para el algoritmo

En este caso los vecindarios son: insertion forward first improvement, insertion backward first improvement y interchange forward first improvement

VND

3. Se realiza búsqueda local a la solución con un vecindario.

Para la primera iteración se inicia con el primer vecindario definido, es decir, insertion forward first improvement.

4. Si la solución generada por el vecindario es mejor:

Se define esa solución como la mejor solución y para la próxima iteración se vuelve a realizar la búsqueda local con el primer vecindario definido.

5. Si no se encuentra una mejor solución:

Para la próxima iteración se realiza búsqueda local con el siguiente vecindario definido.

VND

6. Cuando para una solución no se encuentra una mejor con ninguno de los tres vecindarios definidos, se detiene el algoritmo.

7. El algoritmo también puede detenerse cuando se cumple el tiempo límite definido para cada instancia.

VND

```
procedure VND()
    s ← initial_solution();
    j = 1;
    while j ≤ number_of_neighborhoods
        Find s' ∈ N_j(s)
        if f(s') < f(s) do
                j = 1;
                s ← s';
        else
                j = j + 1;
        end
    end
    return s
end VND
```

También puede acabar cuando se agota el tiempo límite

Combinado

El algoritmo combinado está compuesto con:

- Múltiples soluciones iniciales
- Recocido simulado
- ELS

En general la combinación es así:

Se hace el algoritmo recocido simulado con múltiples soluciones iniciales las cuales se sacan del algoritmo de ruido del trabajo 1. Se selecciona la mejor solución y esta se usa como solución inicial del algoritmo ELS.

Combinado - Múltiples soluciones iniciales

Se generan nsol soluciones inciales (en este caso, nsol = 3), las cuales se generan a partir del algoritmo de ruido del trabajo 1. A cada una de estas soluciones se le hace búsqueda local con el vecindario insertion forward first improvement y luego con el algoritmo de recocido simulado.

Posteriormente, se elige la mejor de las tres soluciones y a esta se le hace búsqueda local con el algoritmo ELS.

Combinado - Recocido simulado

El algoritmo recocido simulado empieza con una solución inicial (s), en este caso, la solución es una del algoritmo de ruido del trabajo 1.

La solución s* es la correspondiente a la mejor solución encontrada. Al principio del algoritmo esta es la misma que la solución inicial.

```
procedure Simulated_Annealing()
    s ←initial_solution()
    s* ← s;
    while stop criteria=false
        T = T0;
        while T > TF do
            l=0;
            while l < L do
                l = l + 1;
                Find s' ∈ N(s);
                d = f(s')− f(s);
                if d <0 do
                        s ← s';
                        if f(s) < f(s*) do
                                s* ← s;
                        end;
                else
                        if random < e^(−d/T) do
                                s ← s';
                        end;
                end;
            end;
            T =rT ;
        end;
    end;
    return s*;
end Simulated_Annealing
```

Combinado - Recocido simulado

Se hacen tres ciclos while:

1. El primer ciclo tiene como criterio de parada el tiempo limite definido, es decir que este ciclo se detiene cuando el algoritmo ya excedió el tiempo límite de ejecución.

2. El segundo ciclo se detiene cuando la temperatura es mayor que la temperatura final (posteriormente se explica cómo se calcula la temperatura) y también tiene como criterio de

```
procedure Simulated_Annealing()
    s ←initial_solution()
    s* ← s;
    while stop criteria=false
        T = T0;
        while T > TF do
            l=0;
            while l < L do
                l = l + 1;
                Find s' ∈ N(s);
                d = f(s')− f(s);
                if d <0 do
                        s ←s';
                        if f(s) < f(s*) do
                                s* ← s;
                        end;
                else
                        if random < e^(−d/T) do
                                s ←s';
                        end;
                end;
            end;
            T =rT ;
        end;
    end;
    return s*;
end Simulated_Annealing
```

Combinado - Recocido simulado

parada el tiempo límite (esto no lo indica el pseudocódigo pero en el código sí está implementado).

3. El tercer ciclo itera la cantidad de veces en que la temperatura permanece constante (L veces) y también tiene como criterio de parada el tiempo límite (esto no lo indica el pseudocódigo pero en el código sí está implementado).

```
procedure Simulated_Annealing()
    s ←initial_solution()
    s* ← s;
    while stop criteria=false
        T = T0;
        while T > TF do
            l=0;
            while l < L do
                l = l + 1;
                Find s' ∈ N(s);
                d = f(s')− f(s);
                if d <0 do
                    s ← s';
                    if f(s) < f(s*) do
                        s* ← s;
                    end;
                else
                    if random < e^(−d/T) do
                        s ← s';
                    end;
                end;
            end;
            T = rT ;
        end;
    end;
    return s*;
end Simulated_Annealing
```

Combinado - Recocido simulado

Para cada iteración se encuentra una nueva solución (s') haciéndole búsqueda local a la solución inicial (s) con el vecindario insertion forward first improvement.

Luego pueden suceder dos caso:

1. Si la nueva solución s' es mejor que la solución s: A s se le asigna el valor de s' y también si la solución s es mejor que la solución s*, a s* se le asigna el valor de s.

```
procedure Simulated_Annealing()
    s ←initial_solution()
    s* ← s;
    while stop criteria=false
        T = T0;
        while T > TF do
            l=0;
            while l < L do
                l = l + 1;
                Find s' ∈ N(s);
                d = f(s')− f(s);
                if d <0 do
                        s ←s';
                        if f(s) < f(s*) do
                                s* ← s;
                        end;
                else
                        if random < e^(−d/T) do
                                s ←s';
                        end;
                end;
            end;
            T =rT ;
        end;
    end;
    return s*;
end Simulated_Annealing
```

Combinado - Recocido simulado

2. Si la nueva solución s' es mejor que la solución s: se genera un número aleatorio entre 0 y 1 y se calcula $e^{-(f(s')-f(s))/T}$. Si entre estos dos números, el aleatorio es el menor, a la solución s se le va a asignar el valor s'. En otras palabras lo que se hace es que se acepta una solución peor usando una probabilidad que es función del empeoramiento.

```
procedure Simulated_Annealing()
    s ←initial_solution()
    s* ← s;
    while stop criteria=false
        T = T0;
        while T > TF do
            l=0;
            while l < L do
                l = l + 1;
                Find s' ∈ N(s);
                d = f(s')− f(s);
                if d <0 do
                        s ←s';
                        if f(s) < f(s*) do
                                s* ← s;
                        end;
                else
                        if random < e^(-d/T) do
                                s ←s';
                        end;
                end;
            end;
            T =rT ;
        end;
    end;
    return s*;
end Simulated_Annealing
```

Combinado - Recocido simulado

Por último se modifica la temperatura multiplicandola por una constante (en este caso, es de 0.7), entonces la temperatura va a empezar en el valor inicial (en este caso es 10) hasta llegar al valor final (en este caso es 1).

Es decir que para este caso, la temperatura va a decrecer así: 10, 7, 4.9, 3,43, 2.401, 1.6807, 1.17649, 0.823543.

Finalmente se retorna la mejor solución encontrada (s*).

```
procedure Simulated_Annealing()
    s ←initial_solution()
    s* ← s;
    while stop criteria=false
        T = T0;
        while T > TF do
            l=0;
            while l < L do
                l = l + 1;
                Find s' ∈ N(s);
                d = f(s')− f(s);
                if d <0 do
                        s ← s';
                        if f(s) < f(s*) do
                                s* ← s;
                        end;
                else
                        if random < e^(−d/T) do
                                s ← s';
                        end;
                end;
            end;
            T = rT;
        end;
    end;
    return s*;
end Simulated_Annealing
```

Combinado - ELS

Se empieza el algoritmo con la solución S*, la cual es la mejor de las soluciones encontradas al aplicar recocido simulado a tres soluciones iniciales.

A esta solución se le hace búsqueda local con el vecindario insertion backward first improvement.

```
1:  initialize random number generator
2:  H(S*)
3:  LS(S*)
4:  for j := 1 to ni do
5:      f̄ := +∞
6:      for k := 1 to nc do
7:          S := S*
8:          Mutate(S)
9:          LS(S)
10:         if f(S) < f̄ then
11:             f̄ := f(S)
12:             S̄ := S
13:         end if
14:     end for
15:     if f̄ < f(S*) then
16:         S* := S̄
17:     end if
18: end for
```

Combinado - ELS

Se itera con dos ciclos, uno que itera ni veces (en esta caso, ni = 10) y otro que itera nc veces (en este caso, nc = 10). Aunque los ciclos pueden terminar antes de que se itere ni/nc veces si se acaba el tiempo límite definido (esto no lo indica el pseudocódigo pero en el código sí está implementado).

También se define una solución S_ en la que su función objetivo (f_) es infinito.

```
1:  initialize random number generator
2:  H(S*)
3:  LS(S*)
4:  for j := 1 to ni do
5:      f̄ := +∞
6:      for k := 1 to nc do
7:          S := S*
8:          Mutate(S)
9:          LS(S)
10:         if f(S) < f̄ then
11:             f̄ := f(S)
12:             S̄ := S
13:         end if
14:     end for
15:     if f̄ < f(S*) then
16:         S* := S̄
17:     end if
18: end for
```

Combinado - ELS

Se define una nueva solución S a la cual se le da el valor de S*, a esta se le hace una perturbación (es decir que se hace un cambio factible a la solución aleatoriamente) y se le hace búsqueda local con el vecindario insertion backward first improvement.

```
1:  initialize random number generator
2:  H(S*)
3:  LS(S*)
4:  for j := 1 to ni do
5:      f̄ := +∞
6:      for k := 1 to nc do
7:          S := S*
8:          Mutate(S)
9:          LS(S)
10:         if f(S) < f then
11:             f̄ := f(S)
12:             S̄ := S
13:         end if
14:     end for
15:     if f̄ < f(S*) then
16:         S* := S̄
17:     end if
18: end for
```

Combinado - ELS

Si la solución S es mejor que la solución S_, se asigna el valor de S a la variable S_.

```
1:  initialize random number generator
2:  H(S*)
3:  LS(S*)
4:  for j := 1 to ni do
5:      f̄ := +∞
6:      for k := 1 to nc do
7:          S := S*
8:          Mutate(S)
9:          LS(S)
10:         if f(S) < f then
11:             f̄ := f(S)
12:             S̄ := S
13:         end if
14:     end for
15:     if f̄ < f(S*) then
16:         S* := S̄
17:     end if
18: end for
```

Combinado - ELS

Para cada iteración del primer ciclo, si la solución S_ es mejor que la solución S*, se le asigna el valor S_ a la variable S*.

Por último se retorna S*.

1: initialize random number generator
2: $H(S^*)$
3: $LS(S^*)$
4: **for** $j := 1$ **to** ni **do**
5: $\quad \bar{f} := +\infty$
6: $\quad$ **for** $k := 1$ **to** nc **do**
7: $\quad\quad S := S^*$
8: $\quad\quad Mutate(S)$
9: $\quad\quad LS(S)$
10: $\quad\quad$ **if** $f(S) < \bar{f}$ **then**
11: $\quad\quad\quad \bar{f} := f(S)$
12: $\quad\quad\quad \overline{S} := S$
13: $\quad\quad$ **end if**
14: $\quad$ **end for**
15: $\quad$ **if** $\bar{f} < f(S^*)$ **then**
16: $\quad\quad S^* := \overline{S}$
17: $\quad$ **end if**
18: **end for**

Comparaciones con Diferentes Valores de Parámetros

Comparaciones con diferentes valores de parámetros

	Avg f(s)	Avg gap	Parámetros
1	1639	46.3%	nsol = 3; T0 = 10; Tf = 1; L = 10; p = 0.7; nit = 10; nc = 3; r = 5
2	1658	48.0%	nsol = 3; T0 = 10; Tf = 1; L = 10; p = 0.8; nit = 10; nc = 3; r = 5
3	1648	46.8%	nsol = 3; T0 = 10; Tf = 1; L = 10; p = 0.9; nit = 10; nc = 3; r = 5
4	1662	48.5%	nsol = 3; T0 = 30; Tf = 1; L = 30; p = 0.7; nit = 30; nc = 3; r = 5
5	1663	48.2%	nsol = 3; T0 = 30; Tf = 1; L = 30; p = 0.8; nit = 30; nc = 3; r = 5
6	1660	48.3%	nsol = 3; T0 = 30; Tf = 1; L = 30; p = 0.9; nit = 30; nc = 3; r = 5

* Debido a que los algoritmos se demoran mucho tiempo en correr, solo se probaron los diferentes parámetros con unas instancias, no con todas.

Comparación entre Algoritmos

Gap

	JSSP1	JSSP2	JSSP3	JSSP4	JSSP5	JSSP6	JSSP7	JSSP8
VND	40.9%	44.7%	55.7%	36.0%	67.8%	54.5%	26.5%	26.2%
Mixed	41.5%	44.7%	47.7%	32.8%	64.6%	50.2%	23.2%	26.6%
Avg	41.2%	44.7%	51.7%	34.4%	66.2%	52.4%	24.8%	26.4%

	JSSP9	JSSP10	JSSP11	JSSP12	JSSP13	JSSP14	JSSP15	JSSP16
VND	26.1%	46.7%	33.3%	6.4%	20.7%	17.5%	11.6%	4.6%
Mixed	29.2%	39.0%	26.9%	6.9%	19.1%	25.2%	12.5%	3.1%
Avg	27.6%	42.9%	30.1%	6.6%	19.9%	21.4%	12.1%	3.8%

Gap

	Avg
VND	32.4%
Mixed	30.8%
Avg	31.6%

Función objetivo

	JSSP1	JSSP2	JSSP3	JSSP4	JSSP5	JSSP6	JSSP7	JSSP8
VND	1396	1425	1833	1656	2052	1962	2299	2320
Mixed	1402	1425	1739	1618	2013	1908	2240	2326
Avg	1399	1425	1786	1637	2033	1935	2270	2323

	JSSP9	JSSP10	JSSP11	JSSP12	JSSP13	JSSP14	JSSP15	JSSP16
VND	2365	2618	3765	3192	3520	3466	6166	5937
Mixed	2424	2480	3586	3207	3474	3694	6217	5852
Avg	2395	2549	3676	3200	3497	3580	6192	5895

Función objetivo

	Avg
VND	2873
Mixed	2850
Avg	2862

Comparación con Algoritmos Anteriores

Comparación con algoritmos anteriores

Abreviaciones de la siguiente tabla:

IBBI	Insertion backward, best improvement
IBFI	Insertion backward, first improvement
IFBI	Insertion forward, best improvement
IFFI	Insertion forward, first improvement
IBI	Interchange forward, best improvement
IFI	Interchange forward, first improvement

Comparación con algoritmos anteriores

	Const.	Noise	GRASP
Avg	36,2%	34,6%	37,4%

	IBBI	IBFI	IFBI	IFFI	IBI	IFI
Avg	32,3%	35,6%	31,5%	35,7%	31,6%	35,6%

	VND	Mixed
Avg	32.4%	30.8%

* El método Noise con r = 5 y GRASP con k = 3.

Tiempo de Cómputo

Tiempo de cómputo

Unidad de tiempo: minutos

	JSSP1	JSSP2	JSSP3	JSSP4	JSSP5	JSSP6	JSSP7	JSSP8
VND	0.037	0.049	0.036	0.065	0.907	0.269	0.759	1.188
Mixed	0.055	0.086	0.116	0.238	1.390	1.459	1.409	1.441
Avg	0.046	0.067	0.076	0.151	1.149	0.864	1.084	1.315

	JSSP9	JSSP10	JSSP11	JSSP12	JSSP13	JSSP14	JSSP15	JSSP16
VND	1.364	1.274	2.638	2.786	5.467	5.202	31.760	28.356
Mixed	1.754	1.549	3.317	3.135	6.886	7.006	38.770	39.314
Avg	1.559	1.412	2.978	2.960	6.177	6.104	35.265	33.835

Tiempo de cómputo

Unidad de tiempo: minutos

	Avg
VND	5.135
Mixed	6.745
Avg	5.940

Tiempo de cómputo

Unidad de tiempo: segundos

	JSSP1	JSSP2	JSSP3	JSSP4	JSSP5	JSSP6	JSSP7	JSSP8
VND	2.235	2.914	2.141	3.893	54.434	16.136	45.551	71.307
Mixed	3.313	5.155	6.949	14.274	83.416	87.518	84.512	86.476
Avg	2.774	4.034	4.545	9.083	68.925	51.827	65.032	78.891

	JSSP9	JSSP10	JSSP11	JSSP12	JSSP13	JSSP14	JSSP15	JSSP16
VND	81.849	76.443	158.292	167.151	328.013	312.094	1905.599	1701.377
Mixed	105.266	92.942	199.045	188.080	413.185	420.353	2326.201	2358.832
Avg	93.558	84.692	178.669	177.615	370.599	366.223	2115.900	2030.105

Tiempo de cómputo

Unidad de tiempo: segundos

	Avg
VND	308.089
Mixed	404.720
Avg	356.405

Conclusiones

Conclusiones

- El algoritmo ELS no logra hacer muchas iteraciones ya que se demora mucho tiempo encontrando una perturbación factible. Es por esto que los algoritmos exceden por muchos segundo el tiempo límite (a veces se demora hasta el doble del tiempo límite), pues el algoritmo se queda mucho tiempo en la función de perturbación.
- Se intentó usar las múltiples soluciones usando el algoritmo ELS pero no fue posible hacerlo de esta manera porque se quedaba mucho tiempo en la función de perturbación, entonces se demoraba mucho tiempo en ejecutar el algoritmo y sobrepasaba por mucho el tiempo límite. Es por esto que se usó múltiples soluciones con el algoritmo recocido simulado para reducir el tiempo de ejecución.

Conclusiones

- El algoritmo combinado logró encontrar la mejor solución (30.8% de gap) comparado con los algoritmos de los otros trabajos.
- El algoritmo combinado (usando los vecindarios de first improvement) fue mejor que los algoritmos de best improvement. El algoritmo combinado logró reducir el gap promedio a 30.8% mientras que el mejor algoritmo de best improvement lo tuvo un gap promedio de 31.5%. Aunque por otra parte, hay que tener en cuenta que no es una diferencia significativa (0.7% de diferencia) teniendo en cuenta que el algoritmo combinado encontró las soluciones en un total de 108 minutos mientras que el mejor de first improvement demoró 59 minutos.

Conclusiones

- El algoritmo VND superó a 6 de los 9 algoritmos de los trabajos anteriores.
- El algoritmo VND (usando los vecindarios de first improvement) no fue mejor que ninguno de los algoritmos de best improvement del trabajo anterior y demoró más que dos de ellos. Estos son los tiempos de ejecución:
  - VND: 108 min
  - Insertion backward - best improvement: 51 min (VND demoró 2.1 veces más)
  - Insertion forward - best improvement: 59 min (VND demoró 1.8 veces más)
  - Interchange forward - best improvement: 109 min (VND demoró un min menos)

Referencias Bibliográficas

Referencias

Rivera Agudelo, J. C. (2023). MS-ELS [Diapositivas]. Escuela de Ciencias Aplicadas e Ingeniería, Universidad EAFIT. https://interactivavirtual.eafit.edu.co/

Rivera Agudelo, J. C. (2023). Recocido Simulado [Diapositivas]. Escuela de Ciencias Aplicadas e Ingeniería, Universidad EAFIT. https://interactivavirtual.eafit.edu.co/

Rivera Agudelo, J. C. (2023). VNS [Diapositivas]. Escuela de Ciencias Aplicadas e Ingeniería, Universidad EAFIT. https://interactivavirtual.eafit.edu.co/