Búsqueda Local Intensificación

Daniela Niño

Descripción de Algoritmos

Descripción de algoritmos

Para este trabajo se hizo búsqueda local usando dos criterios de selección de soluciones: first improvement y best improvement. Ambos criterios de selección se implementaron considerando tres tipos de vecindarios: inserción hacia adelante, inserción hacia atrás e intercambio hacia adelante.

First Improvement

1. Toma una solución inicial

El algoritmo parte de una solución inicial. En la primera iteración usa como solución inicial la del algoritmo constructivo.

2. Halla soluciones vecinas a la solución inicial

El algoritmo empieza a iterar sobre las máquinas y sobre los trabajos para poder hallar soluciones vecinas. Para realizar las soluciones vecinas se pueden usar los métodos insertion forward, insertion backward e interchange forward.

First Improvement

3. Verifica si es factible la solución vecina

Cuando se crea una solución vecina se verifica si esta es factible. En caso de no ser factible se continúa en la búsqueda de una nueva solución vecina.

4. Haya la función objetivo de la solución vecina

En caso de ser factible la solución, a esta se le calcula el valor de la función objetivo.

First Improvement

5. Se comparan los valores de la función objetivo de la solución inicial y de la solución vecina

En caso de que la función objetivo de una solución vecina sea mayor o igual a la de la solución inicial, se continúa con la búsqueda de nuevos vecinos.

En caso contrario, si es menor, se toma esta solución como la nueva solución inicial y empieza otra iteración (se vuelve al paso 1).

First Improvement

6. Se retorna una nueva solución

Cuando se toma una solución inicial y a partir de esta no se hallan vecinos que sean mejores, el algoritmo retorna la última solución encontrada.

Best Improvement

1. Toma una solución inicial y temporal

El algoritmo parte de una solución inicial y una temporal. En la primera iteración usa como solución inicial y temporal la del algoritmo constructivo.

2. Halla soluciones vecinas a la solución temporal

El algoritmo empieza a iterar sobre las máquinas y sobre los trabajos para poder hallar soluciones vecinas. Para realizar las soluciones vecinas se pueden usar los métodos insertion forward, insertion backward e interchange forward.

Best Improvement

3. Verifica si es factible la solución vecina

Cuando se crea una solución vecina se verifica si esta es factible. En caso de no ser factible se continúa en la búsqueda de una nueva solución vecina.

4. Haya la función objetivo de la solución vecina

En caso de ser factible la solución, a esta se le calcula el valor de la función objetivo.

Best Improvement

5. Se comparan los valores de la función objetivo de la solución temporal y de la solución vecina

En caso de que la función objetivo de una solución vecina sea menor a la de la solución temporal, se toma esta solución vecina como la nueva solución temporal.

Luego el algoritmo sigue evaluando el resto de soluciones vecinas hasta terminar de evaluar todo el vecindario. Es decir que aunque haya encontrado o no una mejor solución, en ambos casos el algoritmo va a continuar evaluando el vecindario.

Best Improvement

6. Se compara la solución temporal con la solución inicial

Cuando el algoritmo termina de evaluar todo el vecindario, la solución temporal es la correspondiente a la mejor solución de todo el vecindario.

La solución temporal se compara con la solución inicial. En caso de ser mejor la solución temporal, esta se asigna como la solución inicial y se vuelve a repetir el proceso (se vuelve al paso 1).

Best Improvement

7. Se retorna una nueva solución

Cuando se toma una solución temporal y a partir de esta no se hallan vecinos que sean mejores, el algoritmo retorna la última solución encontrada.

Insertion forward

Toma un elemento en una posición (j) y la inserta en una posición (i).

Suponiendo que tenemos una matriz resultado del trabajo anterior, si tomamos la fila de una máquina m que se ve así:

```
[1 2 3 4 5 6 7 8 9]
```

Y tomamos j = 4 e i = 7 Este método inserta el valor 5 donde está el valor 8 y desplaza el resto de números a la izquierda. Generando esta nueva fila:

```
[1 2 3 4 6 7 8 5 9]
```

Finalmente, el método retorna una copia de la matriz con el cambio que se le hizo a la fila.

Insertion backward

Toma un elemento en una posición (j) y la inserta en una posición (i).

Suponiendo que tenemos una matriz resultado del trabajo anterior, si tomamos la fila de una máquina m que se ve así:

```
[1 2 3 4 5 6 7 8 9]
```

Y tomamos j = 4 e i = 1. Este método inserta el valor 5 donde está el valor 2 y desplaza el resto de números a la derecha. Generando esta nueva fila:

```
[1 5 2 3 4 6 7 8 9]
```

Finalmente, el método retorna una copia de la matriz con el cambio que se le hizo a la fila.

Interchange forward

Toma un elemento en una posición (j) y la intercambia con el elemento en otra posición (i).

Suponiendo que tenemos una matriz resultado del trabajo anterior, si tomamos la fila de una máquina m que se ve así:

```
[1 2 3 4 5 6 7 8 9]
```

Y tomamos j = 4 e i = 7 Este método intercambia el valor 5 con el valor 8. Generando esta nueva fila:

```
[1 2 3 4 8 6 7 5 9]
```

Finalmente, el método retorna una copia de la matriz con el cambio que se le hizo a la fila.

Comparación entre Algoritmos

Comparación entre algoritmos

Abreviaciones de las siguientes tablas:

ILBI	Insertion backward, best improvement
ILFI	Insertion backward, first improvement
IRBI	Insertion forward, best improvement
IRFI	Insertion forward, first improvement
IBI	Interchange, best improvement
IFI	Interchange, first improvement

Gap

	JSSP1	JSSP2	JSSP3	JSSP4	JSSP5	JSSP6	JSSP7	JSSP8
Const.	47,5%	46,8%	58,5%	36,9%	77,8%	54,7%	28,4%	32,3%
ILBI	41,3%	42,5%	56,8%	26,8%	76,1%	54,5%	26,2%	30,1%
ILFI	46,9%	46,4%	57,9%	36,8%	77,7%	54,5%	28,4%	32,2%
IRBI	39,9%	44,7%	52,6%	27,0%	70,5%	54,5%	26,0%	30,1%
IRFI	44,8%	46,4%	57,9%	36,8%	77,6%	54,5%	28,4%	32,2%
IBI	39,9%	39,3%	55,7%	26,8%	72,0%	54,5%	26,2%	30,1%
IFI	44,8%	46,4%	57,9%	36,8%	77,7%	54,5%	28,4%	32,2%
Avg	**43,6%**	**44,6%**	**56,8%**	**32,6%**	**75,6%**	**54,5%**	**27,4%**	**31,3%**

Gap

	JSSP9	JSSP10	JSSP11	JSSP12	JSSP13	JSSP14	JSSP15	JSSP16
Const.	33,2%	51,9%	36,5%	8,8%	23,6%	23,4%	12,8%	5,2%
ILBI	28,3%	46,1%	24,1%	8,1%	21,6%	17,4%	11,7%	5,1%
ILFI	31,3%	51,1%	33,1%	8,5%	23,5%	23,3%	12,7%	5,1%
IRBI	28,2%	46,1%	23,9%	6,4%	21,6%	16,5%	11,8%	5,1%
IRFI	32,4%	51,1%	36,2%	8,5%	23,5%	22,6%	12,8%	5,2%
IBI	28,2%	46,1%	24,1%	8,1%	21,6%	17,4%	11,0%	5,1%
IFI	31,3%	51,1%	36,3%	8,5%	23,5%	22,4%	12,7%	5,1%
Avg	**30,4%**	**49,1%**	**30,6%**	**8,1%**	**22,7%**	**20,4%**	**12,2%**	**5,1%**

Gap

	Avg
Constructivo	36,2%
ILBI	32,3%
ILFI	35,6%
IRBI	31,5%
IRFI	35,7%
IBI	31,6%
IFI	35,6%
Avg	**34,1%**

	Avg
Constructivo	36,2%
Insertion backward	34,0%
Insertion forward	33,6%
Interchange forward	33,6%

Función objetivo

	JSSP1	JSSP2	JSSP3	JSSP4	JSSP5	JSSP6	JSSP7	JSSP8
Const.	1462	1446	1865	1667	2175	1965	2335	2432
ILBI	1400	1404	1845	1545	2154	1962	2294	2392
ILFI	1456	1442	1859	1666	2173	1962	2334	2430
IRBI	1386	1425	1796	1547	2085	1962	2291	2392
IRFI	1435	1442	1859	1666	2172	1962	2334	2430
IBI	1386	1372	1833	1545	2103	1962	2294	2392
IFI	1435	1442	1859	1666	2173	1962	2334	2430
Avg	**1423**	**1425**	**1845**	**1615**	**2148**	**1962**	**2317**	**2414**

Función objetivo

	JSSP9	JSSP10	JSSP11	JSSP12	JSSP13	JSSP14	JSSP15	JSSP16
Const.	2499	2710	3856	3266	3606	3639	6232	5973
ILBI	2406	2606	3507	3244	3547	3463	6172	5963
ILFI	2464	2696	3760	3255	3602	3638	6227	5967
IRBI	2405	2606	3500	3192	3547	3438	6175	5963
IRFI	2483	2696	3849	3256	3602	3618	6230	5971
IBI	2405	2606	3507	3244	3547	3463	6135	5963
IFI	2464	2696	3850	3255	3602	3610	6227	5967
Avg	**2447**	**2659**	**3690**	**3245**	**3579**	**3553**	**6200**	**5967**

Función objetivo

	Avg
Constructivo	2946
ILBI	2869
ILFI	2933
IRBI	2857
IRFI	2938
IBI	2860
IFI	2936
Avg	**2905**

Comparación con Algoritmos Anteriores

Comparación con algoritmos anteriores

	Const.	Noise	GRASP	ILBI	ILFI	IRBI	IRFI	IBI	IFI
Avg	36,2%	34,6%	37,4%	32,3%	35,6%	31,5%	35,7%	31,6%	35,6%

* 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
Const.	0,001	0,001	0,001	0,001	0,002	0,002	0,001	0,001
ILBI	0,136	0,141	0,152	0,458	0,398	0,136	1,012	0,558
ILFI	0,007	0,014	0,004	0,002	0,028	0,049	0,010	0,041
IRBI	0,179	0,101	0,493	0,503	0,677	0,138	1,135	0,579
IRFI	0,007	0,011	0,004	0,002	0,006	0,050	0,004	0,042
IBI	0,204	0,200	0,289	0,535	0,823	0,169	1,247	0,693
IFI	0,008	0,012	0,005	0,003	0,023	0,062	0,014	0,050
Avg	**0,077**	**0,069**	**0,136**	**0,215**	**0,279**	**0,086**	**0,489**	**0,281**

Tiempo de cómputo

Unidad de tiempo: minutos

	JSSP9	JSSP10	JSSP11	JSSP12	JSSP13	JSSP14	JSSP15	JSSP16
Const.	0,003	0,003	0,002	0,002	0,004	0,004	0,009	0,009
ILBI	1,402	1,417	3,554	1,881	2,604	5,755	21,459	10,573
ILFI	0,019	0,122	0,366	0,059	0,125	0,128	1,760	3,618
IRBI	1,185	1,446	4,486	5,212	2,668	9,893	18,812	11,313
IRFI	0,019	0,126	0,373	0,053	0,135	0,128	1,383	1,124
IBI	1,528	1,854	3,016	1,941	3,556	8,114	68,457	15,889
IFI	0,025	0,160	0,451	0,065	0,179	0,183	2,625	5,424
Avg	**0,597**	**0,733**	**1,750**	**1,316**	**1,324**	**3,458**	**16,358**	**6,850**

Tiempo de cómputo

Unidad de tiempo: minutos

	Avg
Constructivo	0,003
ILBI	3,227
ILFI	0,397
IRBI	3,676
IRFI	0,217
IBI	6,782
IFI	0,581
Avg	**2,126**

Conclusiones

Conclusiones

- Los algoritmos de best improvement demoran más en tiempo de ejecución que los de first improvement debido a que deben evaluar todas las soluciones vecinas por cada iteración.
- Los algoritmos de best improvement son muy ineficientes para instancias grandes, pues su tiempo de ejecución puede sobrepasar los 5 minutos.
- En general son mejores las soluciones de best improvement que las de first improvement debido a que recorren todo el vecindario.
- En general, para las instancias más grandes, los algoritmos encuentran soluciones más cercanas a la cota inferior.

Conclusiones

- En general, para las instancias más grandes, los algoritmos encuentran soluciones más cercanas a la solución del algoritmo constructivo. Esto se debe a que el gap de la solución constructiva es muy pequeño entonces para el algoritmo es más difícil encontrar soluciones que minimicen más la función objetivo. Contrario a lo que sucede con las instancias más pequeñas, pues estas tienen un gap más grande.
- Los seis algoritmos desarrollados son mejores que GRASP pero tres de los seis algoritmos desarrollados son mejores que Noise.

Conclusiones

- En general los mejores algoritmos fueron los de interchange forward y los de insertion forward. Esto puede ser debido a que como ambos hacen cambios hacia adelante, pueden hallar la misma solución.

  Si se ve en el gap, los algoritmos de interchange - best improvement e insertion forward - best improvement, encontraron la misma función objetivo para siete instancias.

Referencias Bibliográficas

Referencias

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