|  |
| --- |
| Práctica 1 |
| Algoritmos basados en Entornos y Trayectorias |
| Modelos bioinspirados y heurísticas de búsqueda |

|  |
| --- |
| Victor Manuel Rodriguez Navarro |



Contenido

[Objetivo 2](#_Toc70460335)

[Algoritmo Greedy 2](#_Toc70460336)

[Búsqueda aleatoria 3](#_Toc70460337)

[Búsqueda local del mejor vecino 4](#_Toc70460338)

[Búsqueda local del primer mejor vecino 5](#_Toc70460339)

[Enfriamiento Simulado 6](#_Toc70460340)

[Búsqueda Tabú 7](#_Toc70460341)

[Comparativa y conclusiones 8](#_Toc70460342)

# Objetivo

El propósito de este documento es reunir y analizar las tablas de resultados de los diferentes algoritmos basados en entornos y trayectorias para resolver el Problema del Viajante de Comercio (TSP). Se mostrará un desglose por algoritmo y finalmente, varias tablas comparativas entre todos ellos.

# Algoritmo Greedy

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Algoritmo Greedy** | | | | | | | | | | | | |
|  | **st70** | | **ch130** | | **a280** | | **pa654** | | **vm1084** | | **vm1748** | |
|  | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** |
| **Ejecución1** | 830,00 | 1,00 | 7.579,00 | 1,00 | 3.157,00 | 1,00 | 43.457,00 | 1,00 | 301.476,00 | 1,00 | 408.101,00 | 1,00 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| algoritmoGreedy | |  | algoritmoGreedy | |  | algoritmoGreedy |
| Tiempo de ejecución: 0.745999 | | | Tiempo de ejecución: 0.704498 | | | Tiempo de ejecución: 0.717003 |
| Coste: 830 |  |  | Coste: 7579 |  |  | Coste: 3157 |
|  |  |  |  |  |  |  |
| algoritmoGreedy | |  | algoritmoGreedy | |  | algoritmoGreedy |
| Tiempo de ejecución: 0.747000 | | | Tiempo de ejecución: 0.784001 | | | Tiempo de ejecución: 0.879998 |
| Coste: 43457 | |  | Coste: 301476 | |  | Coste: 408101 |

Tras la ejecución del algoritmo Greedy sobre cada dataset, se han obtenido los estos resultados.

Cabe destacar que solo es necesario ejecutarlo una vez, ya que es un algoritmo totalmente determinista, y dará siempre el mismo resultado al empezar desde un nodo (en nuestro caso el primer nodo).

Por otro lado, es un algoritmo muy simple (se limita a seleccionar el nodo a menor distancia del actual) y eficiente, ya que es capaz de procesar todos los dataset en menos de un segundo cada uno, usando la librería que permite la ejecución multihilo “multiprocessing”.

# Búsqueda aleatoria

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Búsqueda aleatoria** | | | | | | | | | | | | |
|  | **st70** | | **ch130** | | **a280** | | **pa654** | | **vm1084** | | **vm1748** | |
|  | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** |
| **Ejecución1** | 2900 | 112000 | 38704 | 208000 | 29733 | 448000 | 1819347 | 1046400 | 7981407 | 1734400 | 14071939 | 2796800 |
| **Ejecución2** | 2912 | 112000 | 38402 | 208000 | 29959 | 448000 | 1796285 | 1046400 | 7794062 | 1734400 | 14190898 | 2796800 |
| **Ejecución3** | 2811 | 112000 | 38793 | 208000 | 29541 | 448000 | 1798309 | 1046400 | 7946794 | 1734400 | 14142942 | 2796800 |
| **Ejecución4** | 2824 | 112000 | 38588 | 208000 | 29428 | 448000 | 1822048 | 1046400 | 7984070 | 1734400 | 14135601 | 2796800 |
| **Ejecución5** | 2776 | 112000 | 38857 | 208000 | 28949 | 448000 | 1795091 | 1046400 | 7961523 | 1734400 | 14131011 | 2796800 |
| **Ejecución6** | 2836 | 112000 | 38496 | 208000 | 29621 | 448000 | 1786004 | 1046400 | 8010081 | 1734400 | 14148999 | 2796800 |
| **Ejecución7** | 2793 | 112000 | 38520 | 208000 | 29179 | 448000 | 1830159 | 1046400 | 7997939 | 1734400 | 14040688 | 2796800 |
| **Ejecución8** | 2690 | 112000 | 38746 | 208000 | 29520 | 448000 | 1814518 | 1046400 | 7849193 | 1734400 | 14195096 | 2796800 |
| **Ejecución9** | 2872 | 112000 | 38430 | 208000 | 28712 | 448000 | 1815083 | 1046400 | 7966476 | 1734400 | 14195751 | 2796800 |
| **Ejecución10** | 2714 | 112000 | 38846 | 208000 | 29751 | 448000 | 1799080 | 1046400 | 7945923 | 1734400 | 14181537 | 2796800 |
| **Mejor** | 2.690,00 | 112.000,00 | 38.402,00 | 208.000,00 | 28.712,00 | 448.000,00 | 1.786.004,00 | 1.046.400,00 | 7.794.062,00 | 1.734.400,00 | 14.040.688,00 | 2.796.800,00 |
| **Media** | 2.812,80 | 112.000,00 | 38.638,20 | 208.000,00 | 29.439,30 | 448.000,00 | 1.807.592,40 | 1.046.400,00 | 7.943.746,80 | 1.734.400,00 | 14.143.446,20 | 2.796.800,00 |
| **Des. Tip. (s)** | 73,18 | 0,00 | 172,18 | 0,00 | 385,95 | 0,00 | 14.411,81 | 0,00 | 68.778,12 | 0,00 | 52.731,89 | 0,00 |



Tras la ejecución del algoritmo Búsqueda aleatoria sobre 10 semillas aleatorias en cada dataset, se han obtenido los estos resultados.

Este algoritmo es muy simple, pero para nada eficiente ni eficaz. Su coste depende del número de iteraciones que pasemos por parámetro, y estas serán el número de caminos aleatorios que se generarán. Por otra parte, no es eficaz ya que no emplea ninguna heurística de búsqueda ni interacción con el entorno (el entorno es todo el espacio de búsqueda), y sus resultados dependen de la aleatoriedad de obtener una buena solución en ese número fijo de iteraciones.

# Búsqueda local del mejor vecino

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Búsqueda local mejor vecino** | | | | | | | | | | | | |
|  | **st70** | | **ch130** | | **a280** | | **pa654** | | **vm1084** | | **vm1748** | |
|  | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** |
| **Ejecución1** | 1173 | 113505 | 22094 | 209625 | 26412 | 468720 | 1913102 | 1067655 | 8454398 | 1760958 | 15133126 | 3053756 |
| **Ejecución2** | 1224 | 113505 | 22914 | 209625 | 24998 | 468720 | 1874183 | 1067655 | 8441813 | 1760958 | 15016535 | 3053756 |
| **Ejecución3** | 1190 | 113505 | 24253 | 209625 | 26666 | 468720 | 1954410 | 1067655 | 8366794 | 1760958 | 14738116 | 3053756 |
| **Ejecución4** | 1130 | 113505 | 24196 | 209625 | 27215 | 468720 | 2030531 | 1067655 | 8402660 | 1760958 | 14816225 | 3053756 |
| **Ejecución5** | 1194 | 113505 | 22502 | 209625 | 26931 | 468720 | 1973411 | 1067655 | 8376210 | 1760958 | 15018962 | 3053756 |
| **Ejecución6** | 1156 | 113505 | 24078 | 209625 | 26520 | 468720 | 1913706 | 1067655 | 8466410 | 1760958 | 14976830 | 3053756 |
| **Ejecución7** | 1259 | 113505 | 23038 | 209625 | 25446 | 468720 | 2011963 | 1067655 | 8366279 | 1760958 | 14828772 | 3053756 |
| **Ejecución8** | 1157 | 113505 | 25236 | 209625 | 26954 | 468720 | 1938008 | 1067655 | 8208834 | 1760958 | 14872157 | 3053756 |
| **Ejecución9** | 1056 | 113505 | 23045 | 209625 | 25677 | 468720 | 1990716 | 1067655 | 8148831 | 1760958 | 14795905 | 3053756 |
| **Ejecución10** | 1276 | 113505 | 23378 | 209625 | 27366 | 468720 | 1863563 | 1067655 | 8367801 | 1760958 | 14670935 | 3053756 |
| **Mejor** | 1.056,00 | 113.505,00 | 22.094,00 | 209.625,00 | 24.998,00 | 468.720,00 | 1.863.563,00 | 1.067.655,00 | 8.148.831,00 | 1.760.958,00 | 14.670.935,00 | 3.053.756,00 |
| **Media** | 1.181,50 | 113.505,00 | 23.473,40 | 209.625,00 | 26.418,50 | 468.720,00 | 1.946.359,30 | 1.067.655,00 | 8.360.003,00 | 1.760.958,00 | 14.886.756,30 | 3.053.756,00 |
| **Des. Tip. (s)** | 63,85 | 0,00 | 951,24 | 0,00 | 793,49 | 0,00 | 56.233,17 | 0,00 | 103.601,81 | 0,00 | 144.763,74 | 0,00 |

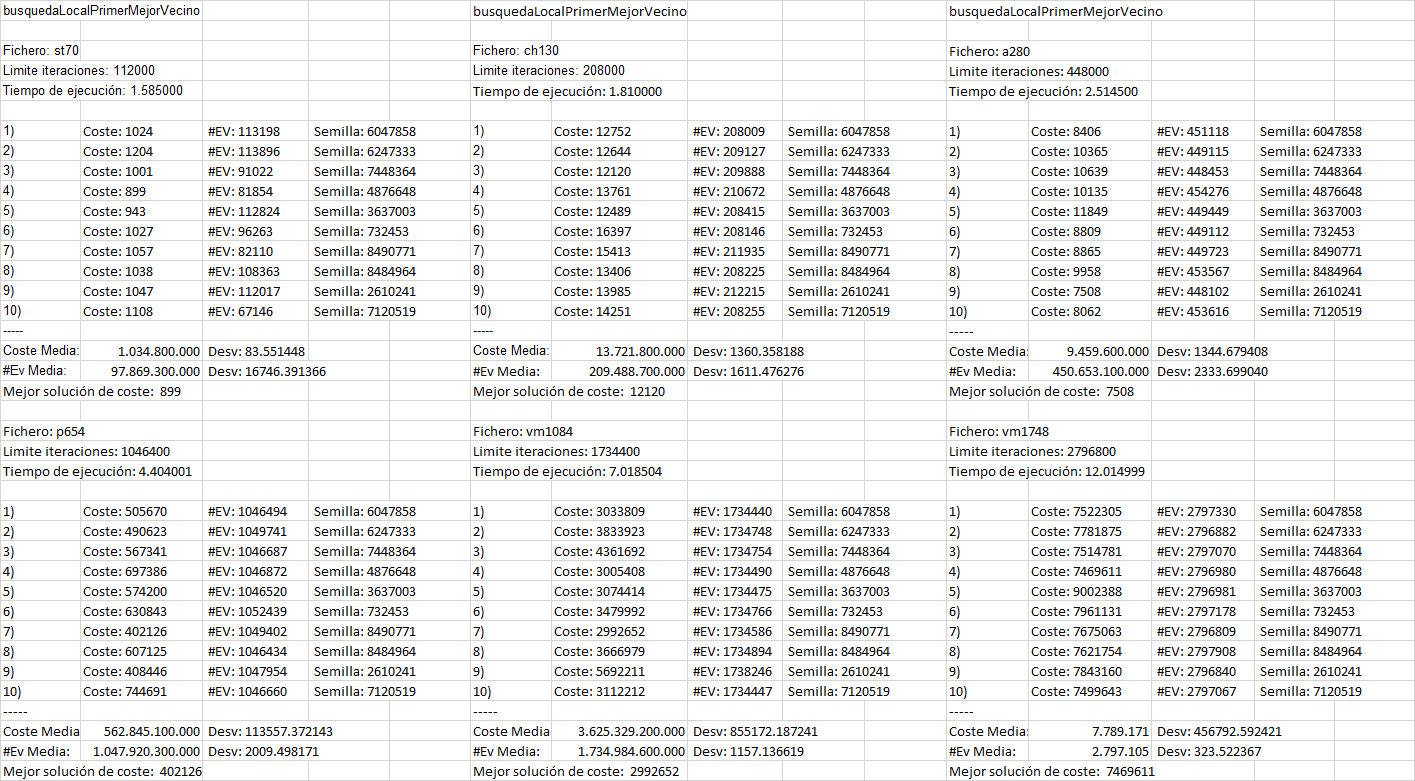


El algoritmo de búsqueda local supone un avance ya utiliza estructura de entorno, en este caso la vecindad dada por el operador opt-2 (intercambiar dos posiciones de un camino para generar un camino solución vecino).

Partimos de una solución inicial aleatoria y vamos explorando todos los vecinos para seleccionar el mejor. Si no encontramos un vecino que mejore la solución actual, se detiene el algoritmo. Igualmente se detendrá al llegar a un número fijo de iteraciones (1600 \* numNodos), que se contabilizan al llamar a la función de coste.

# Búsqueda local del primer mejor vecino

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Búsqueda local primer mejor vecino** | | | | | | | | | | | | |
|  | **st70** | | **ch130** | | **a280** | | **pa654** | | **vm1084** | | **vm1748** | |
|  | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** |
| **Ejecución1** | 1024 | 113198 | 12752 | 208009 | 8406 | 451118 | 505670 | 1046494 | 3033809 | 1734440 | 7522305 | 2797330 |
| **Ejecución2** | 1204 | 113896 | 12644 | 209127 | 10365 | 449115 | 490623 | 1049741 | 3833923 | 1734748 | 7781875 | 2796882 |
| **Ejecución3** | 1001 | 91022 | 12120 | 209888 | 10639 | 448453 | 567341 | 1046687 | 4361692 | 1734754 | 7514781 | 2797070 |
| **Ejecución4** | 899 | 81854 | 13761 | 210672 | 10135 | 454276 | 697386 | 1046872 | 3005408 | 1734490 | 7469611 | 2796980 |
| **Ejecución5** | 943 | 112824 | 12489 | 208415 | 11849 | 449449 | 574200 | 1046520 | 3074414 | 1734475 | 9002388 | 2796981 |
| **Ejecución6** | 1027 | 96263 | 16397 | 208146 | 8809 | 449112 | 630843 | 1052439 | 3479992 | 1734766 | 7961131 | 2797178 |
| **Ejecución7** | 1057 | 82110 | 15413 | 211935 | 8865 | 449723 | 402126 | 1049402 | 2992652 | 1734586 | 7675063 | 2796809 |
| **Ejecución8** | 1038 | 108363 | 13406 | 208225 | 9958 | 453567 | 607125 | 1046434 | 3666979 | 1734894 | 7621754 | 2797908 |
| **Ejecución9** | 1047 | 112017 | 13985 | 212215 | 7508 | 448102 | 408446 | 1047954 | 5692211 | 1738246 | 7843160 | 2796840 |
| **Ejecución10** | 1108 | 67146 | 14251 | 208255 | 8062 | 453616 | 744691 | 1046660 | 3112212 | 1734447 | 7499643 | 2797067 |
| **Mejor** | 899,00 | 67.146,00 | 12.120,00 | 208.009,00 | 7.508,00 | 448.102,00 | 402.126,00 | 1.046.434,00 | 2.992.652,00 | 1.734.440,00 | 7.469.611,00 | 2.796.809,00 |
| **Media** | 1.034,80 | 97.869,30 | 13.721,80 | 209.488,70 | 9.459,60 | 450.653,10 | 562.845,10 | 1.047.920,30 | 3.625.329,20 | 1.734.984,60 | 7.789.171,10 | 2.797.104,50 |
| **Des. Tip. (s)** | 83,55 | 16.746,39 | 1.360,36 | 1.611,48 | 1.344,68 | 2.333,70 | 113.557,37 | 2.009,50 | 855.172,19 | 1.157,14 | 456.792,59 | 323,52 |



Este algoritmo es similar al anterior, solo que al encontrar un vecino que haga disminuir el coste de la solución actual, se detiene la exploración de vecinos.

Esto reduce la capacidad de exploración, pero al usar menos veces la función de coste en exploración de vecinos, consigue aprovechar mejor la limitación de iteraciones y por tanto, obtiene mejores resultados.

Por otra parte, ambos algoritmos de búsqueda local son eficientes y su tiempo de ejecución es considerablemente menor al de búsqueda aleatoria, tardando 12s en el dataset más grande frente a los 8000s del de búsqueda aleatoria.

# Enfriamiento Simulado

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Enfriamiento Simulado** | | | | | | | | | | | | |
|  | **st70** | | **ch130** | | **a280** | | **pa654** | | **vm1084** | | **vm1748** | |
|  | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** |
| **Ejecución1** | 950 | 112001 | 10560 | 208001 | 6048 | 448001 | 318852 | 1046401 | 1300005 | 1734401 | 2215078 | 2796801 |
| **Ejecución2** | 1126 | 112001 | 11910 | 208001 | 6314 | 448001 | 342624 | 1046401 | 1398311 | 1734401 | 2398102 | 2796801 |
| **Ejecución3** | 1054 | 112001 | 11424 | 208001 | 6781 | 448001 | 320283 | 1046401 | 1322923 | 1734401 | 2356964 | 2796801 |
| **Ejecución4** | 900 | 112001 | 9981 | 208001 | 6315 | 448001 | 307500 | 1046401 | 1233144 | 1734401 | 2245650 | 2796801 |
| **Ejecución5** | 1094 | 112001 | 11193 | 208001 | 6218 | 448001 | 374388 | 1046401 | 1292924 | 1734401 | 2308416 | 2796801 |
| **Ejecución6** | 910 | 112001 | 11160 | 208001 | 6418 | 448001 | 322763 | 1046401 | 1306309 | 1734401 | 2260814 | 2796801 |
| **Ejecución7** | 955 | 112001 | 11559 | 208001 | 5521 | 448001 | 355828 | 1046401 | 1286527 | 1734401 | 2279721 | 2796801 |
| **Ejecución8** | 976 | 112001 | 10737 | 208001 | 5856 | 448001 | 349229 | 1046401 | 1237837 | 1734401 | 2209846 | 2796801 |
| **Ejecución9** | 1121 | 112001 | 10357 | 208001 | 5236 | 448001 | 367920 | 1046401 | 1277652 | 1734401 | 2203989 | 2796801 |
| **Ejecución10** | 916 | 112001 | 10434 | 208001 | 5974 | 448001 | 389706 | 1046401 | 1287425 | 1734401 | 2272878 | 2796801 |
| **Mejor** | 900,00 | 112.001,00 | 9.981,00 | 208.001,00 | 5.236,00 | 448.001,00 | 307.500,00 | 1.046.401,00 | 1.233.144,00 | 1.734.401,00 | 2.203.989,00 | 2.796.801,00 |
| **Media** | 1.000,20 | 112.001,00 | 10.931,50 | 208.001,00 | 6.068,10 | 448.001,00 | 344.909,30 | 1.046.401,00 | 1.294.305,70 | 1.734.401,00 | 2.275.145,80 | 2.796.801,00 |
| **Des. Tip. (s)** | 89,77 | 0,00 | 612,03 | 0,00 | 449,75 | 0,00 | 27.332,69 | 0,00 | 46.102,77 | 0,00 | 64.080,09 | 0,00 |



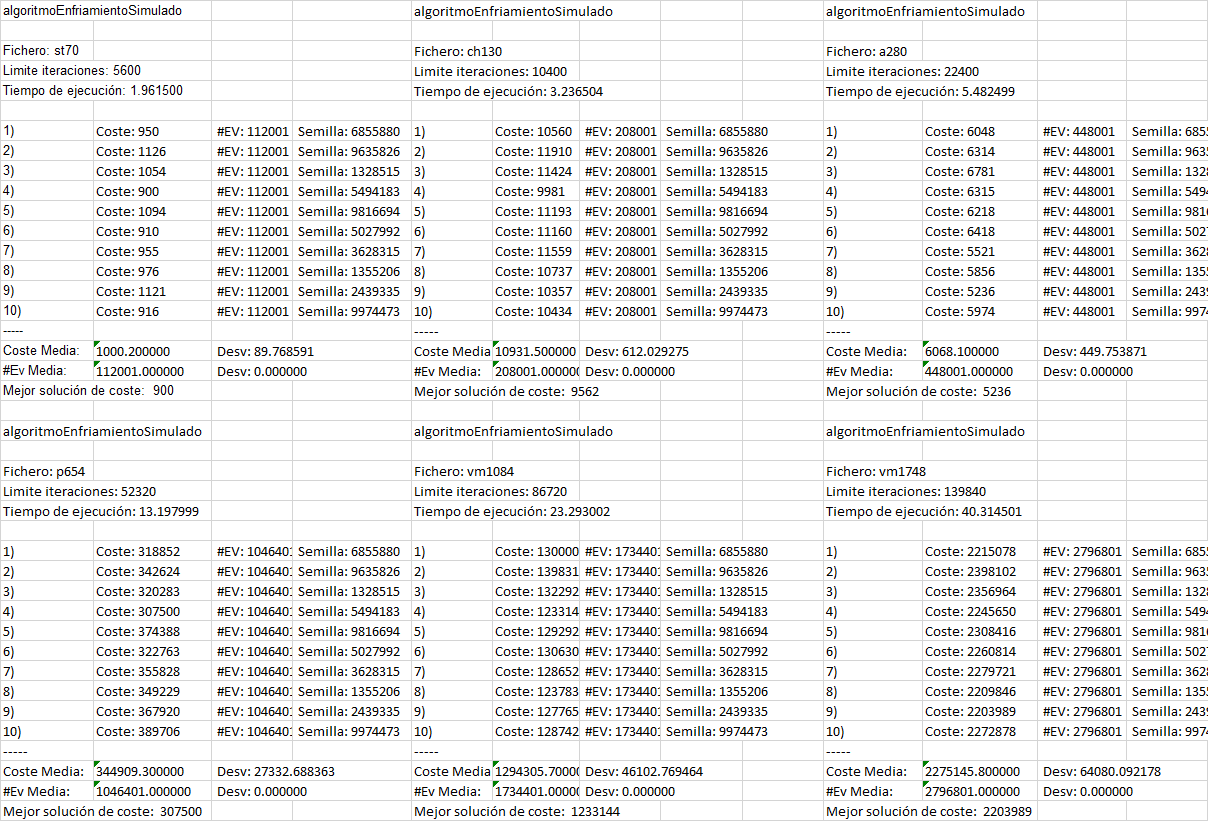
Este algoritmo es un método de búsqueda global, que permite movimientos hacia soluciones peores para poder escapar de mínimos locales. Mediante el parámetro de temperatura se consigue mayor probabilidad de aceptar soluciones peores al principio (exploración) y menor probabilidad al final (explotación), haciendo que la búsqueda se diversifique al principio y se intensifique al final.

Nuestra implementación usa el esquema de enfriamiento de Cauchy y un número fijo de soluciones por cada temperatura (20). La condición de parada es de un número de iteraciones (80\*n).

Este algoritmo consigue resultados notablemente mejores que la búsqueda aleatoria y local, aunque requiere un poco más de tiempo de procesamiento, no obstante, asumible.

# Búsqueda Tabú

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Búsqueda Tabú** | | | | | | | | | | | | |
|  | **st70** | | **ch130** | | **a280** | | **pa654** | | **vm1084** | | **vm1748** | |
|  | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** | **Coste** | **#EV** |
| **Ejecución1** | 980 | 112003 | 10250 | 208004 | 7122 | 448004 | 98462 | 1046404 | 1493573 | 1734405 | 2570327 | 2796804 |
| **Ejecución2** | 1073 | 112004 | 11195 | 208004 | 2996 | 448004 | 329114 | 1046403 | 1469902 | 1734404 | 2409871 | 2796802 |
| **Ejecución3** | 992 | 112005 | 11310 | 208004 | 2879 | 448004 | 99219 | 1046404 | 1395645 | 1734403 | 2696201 | 2796804 |
| **Ejecución4** | 1022 | 112004 | 10306 | 208003 | 3245 | 448004 | 100353 | 1046404 | 1416843 | 1734404 | 2303177 | 2796804 |
| **Ejecución5** | 1023 | 112005 | 9992 | 208005 | 2868 | 448003 | 328567 | 1046401 | 1367140 | 1734403 | 2641385 | 2796804 |
| **Ejecución6** | 1065 | 112004 | 11285 | 208005 | 3474 | 448005 | 103497 | 1046404 | 1469118 | 1734404 | 2604359 | 2796805 |
| **Ejecución7** | 989 | 112004 | 10330 | 208005 | 3034 | 448004 | 348415 | 1046404 | 1404381 | 1734404 | 2660937 | 2796805 |
| **Ejecución8** | 1013 | 112003 | 10816 | 208004 | 3126 | 448003 | 102587 | 1046405 | 1503934 | 1734404 | 2436246 | 2796804 |
| **Ejecución9** | 1076 | 112005 | 10856 | 208003 | 3141 | 448005 | 339662 | 1046402 | 1566976 | 1734405 | 2613595 | 2796804 |
| **Ejecución10** | 976 | 112004 | 10890 | 208004 | 3466 | 448003 | 100851 | 1046403 | 1470293 | 1734404 | 2338296 | 2796804 |
| **Mejor** | 976,00 | 112.003,00 | 9.992,00 | 208.003,00 | 2.868,00 | 448.003,00 | 98.462,00 | 1.046.401,00 | 1.367.140,00 | 1.734.403,00 | 2.303.177,00 | 2.796.802,00 |
| **Media** | 1.020,90 | 112.004,10 | 10.723,00 | 208.004,10 | 3.535,10 | 448.003,90 | 195.072,70 | 1.046.403,40 | 1.455.780,50 | 1.734.404,00 | 2.527.439,40 | 2.796.804,00 |
| **Des. Tip. (s)** | 38,45 | 0,74 | 474,21 | 0,74 | 1.277,95 | 0,74 | 121.800,63 | 1,17 | 59.935,93 | 0,67 | 142.467,10 | 0,82 |



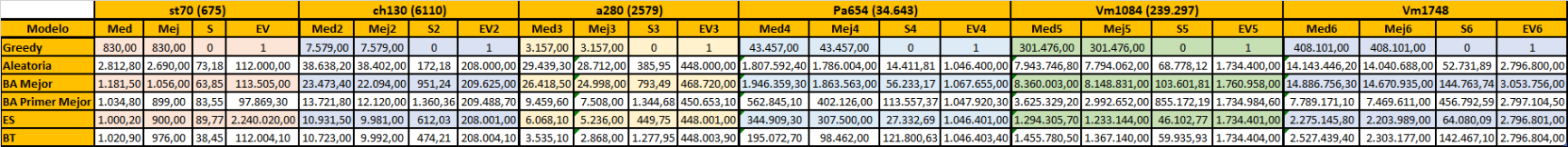
Este algoritmo presenta soluciones ante el problema de quedarse atrapado en óptimos locales. Es capaz de aceptar movimientos de empeoramiento, modificar la estructura de entorno y, a diferencia del anterior, permite volver a comenzar la búsqueda desde otra solución inicial.

Para ello usa dos estructuras de memoria adaptativa: memoria corto plazo o lista tabú y memoria a largo plazo (que permite intensificar o diversificar la búsqueda) para reinicializar la búsqueda a partir de otra solución.

El tiempo de procesamiento es más elevado que el algoritmo anterior, pero obtiene mejores resultados en los datasets a280 y pa654 y en el resto, resultados similares.

# Comparativa y conclusiones

A continuación, exponemos una comparativa entre todos los algoritmos, desglosada por dataset.



Tomaremos el algoritmo Greedy como referencia, ya que es el que mejores resultados obtiene para cada dataset.

El algoritmo de búsqueda aleatoria da resultados pésimos, no solo por su coste, que es el peor o segundo peor en la mayoría de los datasets, sino por su baja eficiencia (al hacer gran cantidad de llamadas a la función de coste) y ausencia de heurísticas.

Por otro lado, vemos una gran diferencia entre búsqueda local del mejor vecino y del primer mejor. Esto se debe a que la búsqueda del primer mejor, consigue avanzar la búsqueda consiguiendo soluciones de menor coste gastando menos iteraciones. Ambas suelen terminar por el límite de iteraciones, pero la búsqueda del primer mejor consigue descender el coste más rápidamente, aunque sacrificando todo el potencial de exploración que tiene generar todos los vecinos.

En cuanto a los algoritmos con mejor resultado, destacan el de Enfriamiento Simulado y Búsqueda Tabú, obteniendo este último notables mejores resultados para los datasets a280 y pa654. La capacidad que tiene este último algoritmo de re-inicialización para diversificar/intensificar con la memoria a largo plazo es un plus.

Con respecto al análisis de la desviación típica del coste, igualmente el algoritmo Greedy no tiene rival, ya que es un algoritmo totalmente determinista y por ello, presenta una desviación típica de 0. Solo es necesario ejecutarlo una vez por dataset.

En cuanto al resto de algoritmos, destaca el de Búsqueda Aleatoria, que como vimos en clase, se puede modelar la probabilidad de obtener una solución buena en un espacio de búsqueda según las iteraciones y también Enfriamiento Simulado al presentar baja desviación típica y, por tanto, mostrar que es un algoritmo robusto y consistente.

Por último, cabe decir que los parámetros están configurados para realizar un número de evaluaciones muy similar en cada algoritmo y, por tanto, poder comparar sus resultados en igualdad de condiciones.