Metaheuristics for the MDMKP

June 17, 2019

1 Goals

Metaheuristics commonly have a wide array of parameters that must be tuned in order to obtain good performace. This task is often time consuming, and can lead to brittle systems ill-equiped to handle changing problems. Instead, we focus solely on metaheuristics that do not require any parameter tuning. These methods often have the added effect of being simple and easy to implement, while still giving competitive results.

2 Algorithms Used

2.1 Jaya

Jaya is a metaheuristic proposed by Dr. Rao of the Sardar Vallabhbhai National Institute of Technology. It was originally developed to work on continous problems, but has been modified to work on binary problems.

Jaya creates a new solution by moving an existing solution towards the best solution found, while simultaneously moving it away from the worst solution found. In the continous space, each movement vector is scaled by some continous value between zero and one. However, as the MDMKP is a binary formulation, there are no fractional movements. Instead, each bit is scaled either 0% or 100% of it's distance vector.

A strict conversion of the Jaya algorithm would require that the 0% solution is selected over the 100% solution according to two randomly chosen probabilites, one for the best solution, and one for the worst. However, testing has shown that a strategy that uniformly selects scaling factors outperforms weighted selection. This uniform selection strategy is formally defined as follows:

Given a population X, define *best* as the best performing solution, and *worst* as the worst performing solution. r is the random function, and uniformly selects an element from the set. For every solution S in the population, construct a modified solution according to the following transformation applied to every bit i in S:

$$S'_i = S_i + r(\{0,1\}) * (best_i - S_i) - r(\{0,1\}) * (worst_i - S_i) > 0 \tag{1}$$

For the faithful but worse performing method, the following transformation is used, where bs is the scaling factor from 0 to 1 for the best solution, and ws is the worst solution scaling factor:

$$S'_i = S_i + (r([0,1]) < bs) * (best_i - S_i) - (r([0,1]) < ws) * (worst_i - S_i) > 0 \tag{2}$$

Jaya instances using equation 2 are refered to as "jaya_v2" in this implementation, while the uniform method is just "jaya".

If the new solution generated by the Jaya transformation is valid and performs better than the old solution, replace it in the population. One iteration of Jaya will attempt this transformation for every solution in the population.

In this implementation, a modification of Jaya is used. Instead of using the best and worst solution every time, parameters m and n are generated, such that the best solution is selected from $random(sorted\_population[start : m])$, and the worst solution is selected from $random(sorted\_population[end - n : end])$. This introduces two parameters to the metaheuristic, a violation of the goals. However, higher m and n values improve performance.

2.2 TLBO

Teaching-Learning Based Optimization is another metaheuristic developed by Dr. Rao for continous problems . It is a two phase algorithm: the Teaching phase, and the Learning phase.

2.2.1 TBO

The first phase, Teaching Based Optimization, constructs new solutions by moving a given solution along the distance vector found by moving from the mean solution to the best solution. The vector is sometimes doubled. Once again, partial movements are impossible, so a 0% or 100% strategy is applied to each individual bit. Formally, it is defined as follows: For every bit index i in a solution S, update a new solution according to:

$$S'_i = \Big(S_i + r(\{0,1\}) * \big(best_i - r(\{1,2\}) * average_i\big)\Big) > 0 \tag{3}$$

where *best* is the best scoring solution in the population, and *average* is the average of all solutions in the population.

There are two different methods for calculating the average value. The first is to order the population by objective function values, and then use the middle solution as the median average. However, this has a different intent than the continous method: instead of acting as the centerpoint of the population, it is an effectively randomly chosen midtier solution.

Instead, the mean is calculated as it would be in the continous space. This will results in a vector of numbers between 0 and 1. This can be converted to a discrete representation by treating the values as a likelihood chance of being turned on, like so:

$$S'_i = \Big(S_i + r(\{0,1\}) * \big(best_i - r(\{1,2\}) * (r([0,1]) < average_i)\big)\Big) > 0 \quad (4)$$

Notice that the random function r was given a continous range from 0 to 1, not a discrete set. This probability method has been shown to give better results than the median method in testing.

Equation 5 uses a uniform selection strategy, much like the uniform Jaya variant. A weighted TBO variant has also been implemented, and is reffered to as"TBO_v2" in this implementation. With a weight w uniformly selected from 0 to 1, the weighted perturbation is defined as follows:

$$S'_i = \Big(S_i + (r([0,1]) < w) * \big(best_i - r(\{1,2\}) * (r([0,1]) < average_i)\big)\Big) > 0 \quad (5)$$

The TBO transform is used in the same way as the jaya transform: apply to every solution in the population, and replace the solutions if the new solutions have better objective function scores.

Dr. Rao originally specified that, much like Jaya, only the absolutely best solution is used. But one again, performance is improved by randomly selecting a solution from some n of top solutions.

Similarly, the mean could be calculated from a subset of the population, instead of the whole, in order to increase the variety of generated solutions. However, this would be computationally expensive, and TBO already produces some of the most diverse populations of any metaheuristic implemented here.

CBO In the same way solutions are moved according to the vector from the mean to a selected good solution, they can be moved according to the vector from a selected bad solution to the mean. This is referred to as Clown Based Optimation, and is otherwise implemented identically to TBO.

2.2.2 LBO

The second phase of TLBO, Learning Based Optimization, acts by selecting two solutions, and moving the worse performing solution some distance along the vector between them[1]. In the binary space, the vector between points becomes a set of shortest paths from one solution to another, from which a new solution is selected. If the new solution has a better score than the worse performing solution, it is replaced. Solutions cannot be the same. In this implementation, a single instance of LBO is guaranteed to act on every solution at least once, like so:

Algorithm 1 Learning Based Optimization

```
for $sol_1 \in X$ do
  $sol_2 = rand(X)$
  while $sol_1 = sol_2$ do
    $sol_2 = rand(X)$
  end while
  $student, teacher = sort(sol_1, sol_2)$
  $new\_student = copy(student)$
  for $i \in 0..length(student)$ do
    $new\_student_i \mathrel{+}= rand(\{0,1\}) * (teacher_i - student_i)$
  end for
  if $score(new\_student) > score(student)$ then
    $X_{student} = new\_student$
  end if
end for
```

Once again, an alternative, weighted perturbation exists, referred to as "LBO_v2". It is defined as:

$$S_i' = S_i + (r([0,1]) < w) * (teacher_i - S_i) \tag{6}$$

where, once again, w is a randomly selected probability from 0 to 1 that is the same for each bit.

In Rao's implementation, a check to make sure the two selected solutions have different objective values is included. This check is omitted from this implementation, as it is arbitrary, and the uniqueness check is sufficient.

[1] It is worth noting that the vector from the better to the worse solution contains all the same points as the vector from the worse to the better—the only difference is what solution is replaced, if the new solution yields an improved score.

While TBO and LBO are typically used together to form the TLBO algorithm, each can be used individually.

2.3 Genetic Algorithm

The classic Genetic Algorithm for binary problems is to take two solutions, randomly select a breakpoint, and then construct two new solutions like so:

$$\begin{aligned} new\_solution\_1 &= sol\_1[: breakpoint] + sol\_2[breakpoint :] \\ new\_solution\_2 &= sol\_2[: breakpoint] + sol\_1[breakpoint :] \end{aligned} \tag{7}$$

This allows for very fast construction of new solutions, but is limited in the amount of new solutions it can produce. For this implementation of a genetic algorithm, a probability based approach is used.

The procedure for generating a single new solution is as follows. First, n solutions are selected. Let M be the column-wise mean of the matrix of the selected solutions. The percentage chance of a bit b being turned on in the new solution is given by M_b. More than one solution could be generated from a single probability matrix M, but this implementation does not do so.

The Genetic Algorithm is used to generate new solutions, and if the new solution scores better than its worst performing parent, the parent is replaced. This process continues until some termination criteria is met.

This Genetic Algorithm is not a parameterless method, as the amount of parents must be determined by the user.

2.4 Variable Neighborhood Descent

Neighborhood Descent is a local method that acts on only one solution at a time. Neighborhood search works as follows: for a solution S, a neighborhood N of related solutions is generated. The best performing solution from this neighborhood is selected, and if it outperforms S, S is replaced. If S is not outperformed, stop the search. Variable Neighboorhood Descent uses more than one neighborhood.

For this implementation, two neighborhoods are used. The first is based on the flip transformation: for every bit in the solution, generate a new solution with the bit flipped. The second is based on the swap transformation: for every two unalike bits in the solution, generate a new solution with the bits flipped.

2.5 The Repair Operator

MDMKP problems often have very sparse regions of feasibility within the possible solution space. The repair operator is a function that attempts to move an infeasible solution into the realm of feasibility. This method works as so:

Algorithm 2 Repair Operator

```
prev_infeas = $\infty$
curr_infeas = $\infty - 1$
**while** cur_infeas < prev_infeas **do**
  feasible_solutions = $\emptyset$
  least_infeas_b = -1
  **for** b $\in$ solution **do**
    new_sol = copy(solution)
    new_sol$_b$ = !new_sol$_b$
    **if** feasibility(new_sol) < curr_infeas **then**
      curr_infeas = feasibility(new_sol)
      least_infeas_b = b
      **if** feasible(new_sol) **then**
        feasible_solutions += new_sol
      **end if**
    **end if**
  **end for**
  **if** feasible_solutions **then**
    **return** best_scoring(feasible_solutions)
  **end if**
  solution$_least\_infeas\_b$ = !solution$_least\_infeas\_b$
**end while**
**return** NULL
```

This a very simple version of the Repair Operator, that has many duplicate operations. In order to be performant in code, it is necessary to cache the infeasibility throughout the algorithm, and update it to account for whatever the currently flipped bit is.

3 Generating an Initial Population

Since the majority of randomly generated solutions for the MDMKP will be infeasible, it can be difficult to generate a viable initial population. The steps to generate a feasible solution are as follows:

1. Create a random permutation I of the numbers $1..n$, where n is the amount of bits in a solution
2. Create a new solution S with every bit disabled
3. For every index i in I, if enabling S_i will not violate a dimension constraint, enable S_i
4. Once every i in I has been checked, run the repair operator on the generated solution S
5. If S is feasible, add it to the set of feasible solutions
6. Create a new solution S with every bit enabled
7. For every i in I, if disabling S_i will not violate a demand constraint, disable S_i
8. Once every i in I has been checked, run the repair operator on the generated solution S
9. If S is feasible, add it to the set of feasible solutions

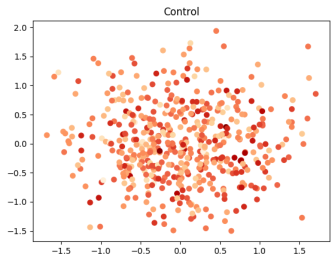
This procedure is run until the target population size has been reached, or some other termination criteria stops the process.

4 Managing Population Diversity

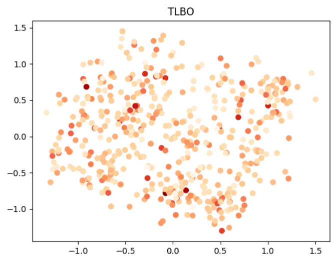
Population based optimization algorithms rely on creating new solution based off of existing solutions. As such, it is important that the diversity of a population of solutions is preserved. However, the strategy of replacing solutions in the population with better performing solutionshas a tendency to cause the population as a whole to cluster around several local minima. For example, compare the Principle Component Analysis of the initial population, and the population produced by TLBO, found at Figure **??**.

Figure **??** shows that the clustering effect is less visible on a higher dataset with more variables. To determine if this is due to the clustering being less prevalent on problems with more variables, or PCA being a less effective visualization, it is necessary to create a metric to measure diversity.

The diversity metric is defined at Algorithm 3.

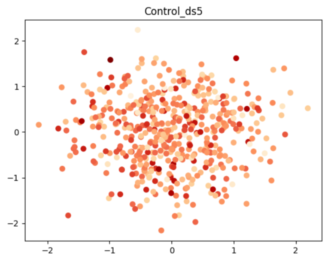


(a) Initial Population

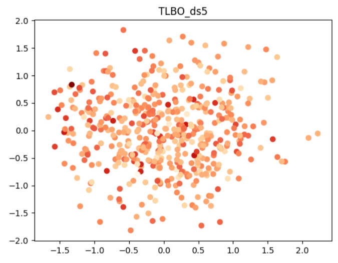


(b) TLBO Population

Figure 1: Example of Metaheuristic Clustering Behavior



(a) Initial Population



(b) TLBO Population

Figure 2: Clustering Behavior on a Higher Dataset

Algorithm 3 Diversity Metric

```
total = 0
for solution ∈ population do
    distances = [euclidean_distance(solution, sol2) for sol2 in population]
    sort(distances)
    total += sum(distances[1:4])
end for
return total÷(length(population)*3)
```

The average diversity metrics for the results for the Beasley MDMKP dataset are shown in Table 1.

We see that all the metaheuristics tested (except Local Flip, a subset of VND) are intensification techniques. As population-based metaheuristics rely on creating new solutions from diverse inspiration solutions, a loss of diversity will lead to a loss of effectiveness. It is therefore beneficial to devise a method to limit the progressive diversity loss.

There are several methods to limit loss of diversity, such as expanding the solution pool, reintroducing old, poorly performing outlier solutions to the population, or ensuring that any new solution replacing an old solution must be suffiently different to every other solution in the population, which is the strategy used in this implementation.

To ensure that new solutions are significantly unique, the VND local search is attempted on every solution generated by other metaheuristics. This makes it impossible for two solutions differing in only one bit to be a part of the population, and has the added benefit of reducing the search space from finding the best solution, to finding the best solution watershed. Results for this method are shown in Table 2. We see that generally, methods with local search embedded have a higher diversity than methods without.

Combining a local search with other metaheuristics is a strategy that has been tried before, for example by Dr. Vasko et. al. [**?**]. However, Dr. Vasko et. al. created a hybrid method consisting on a local search followed by some other metaheuristic. Testing has shown that this implementation's method of embedding a local search, rather than prefixing one, reduces the percent deviation from the optimal by five to ten times for some metaheuristics.

When the embedded local search strategy is used, it is also embedded into the initial population generation algoritm, in order to prevent initial solutions adjacent to a local minima from becoming "stuck" in the population.

5 Comparing Metaheuristics

Some metaheuristics, such as LBO and GA, are very similar for some circumstances. It would be advantageous to be able to quantify exactly how similar these algorithms are. This can be done by totalling the Manhattan distances between the linear assigment of one resultant population to another.

Manhattan distance, also called cityblock distance, is a measure of the discrete steps needed to turn one solution into another. For binary problems,

Table 1: Metaheuristic Diversity Results

algorithm	dataset 4 diversity
LBO	1.92
LBO_v2	1.95
TLBO	2.0
TLBO_v2	2.02
GA20	2.07
GA30	2.08
GA5	2.09
GA10	2.09
GA2	2.13
J30_v2	2.82
J30	3.1
J15_v2	3.57
T30	3.66
T30_v2	3.67
T15	3.8
J15	3.81
T15_v2	3.84
T7_v2	3.93
J3_v2	3.93
T7	3.94
J7_v2	3.94
J3	4.01
T3	4.03
T3_v2	4.06
J7	4.1
T1	4.26
J1	4.26
T1_v2	4.33
J1_v2	4.36
LS	4.63
VND	4.87
C30_v2	5.12
C1	5.13
C30	5.17
C15_v2	5.18
C1_v2	5.18
C3	5.22
C7_v2	5.22
C15	5.24
C3_v2	5.24
C7	5.26
control	5.64
LF	5.66

Table 2: Metaheuristic Local Search Diversity Results

complete diversities	dataset 1
LBO	1.92
LBO_v2	1.95
TLBO	2.0
TLBO_v2	2.02
GA20	2.07
GA30	2.08
GA5	2.09
GA10	2.09
GA2	2.13
LBO_ls	2.27
LBO_ls_v2	2.32
TLBO_ls	2.34
GA5_ls	2.35
TLBO_ls_v2	2.36
GA10_ls	2.36
GA20_ls	2.37
GA30_ls	2.38
GA2_ls	2.38
J30_v2	2.82
J30_ls_v2	2.98
J30_ls	3.03
J30	3.1
J15_ls_v2	3.24
J15_ls	3.31
J7_ls	3.37
J7_ls_v2	3.38
J3_ls	3.38
J3_ls_v2	3.4
T30_ls	3.55
T30_ls_v2	3.56
J15_v2	3.57
T15_ls	3.59
T15_ls_v2	3.62
T7_ls	3.63
T3_ls	3.64
T7_ls_v2	3.66
T3_ls_v2	3.66
T30	3.66
T30_v2	3.67
T1_ls	3.68
J1_ls	3.69
T1_ls_v2	3.69
J1_ls_v2	3.74
T15	3.8
J15	3.81
T15_v2	3.84
C30_ls	3.92
C1_ls	3.92
T7_v2	3.93
J3_v2	3.93
T7	3.94
C3_ls	3.94
J7_v2	3.94
C15_ls	3.95
C7_ls	3.95
C30_ls_v2	3.98
C1_ls_v2	3.99
C15_ls_v2	4.0
C7_ls_v2	4.01
C3_ls_v2	4.01
J3	4.01
T3	4.03
T3_v2	4.06
J7	4.1
T1	4.26
J1	4.26
T1_v2	4.33
J1_v2	4.36
LS	4.63
VND	4.87
C30_v2	5.12
C1	5.13
C30	5.17
C15_v2	5.18
C1_v2	5.18
C3	5.22
C7_v2	5.22
C15	5.24
C3_v2	5.24
C7	5.26
control	5.64
LF	5.66

it is a measure of how many bits are different.

Linear Assignment, or the Assignment Problem, is the process of assigning n agents to n tasks to minimize the cost of assignment[2].

For every problem, an assignment cost can be computed between the resulting populations of two metaheuristics. A cost matrix can be constructed from a set of metaheuristics. Averaging the cost matrixes for every problem in a dataset will increase consistency of the results.

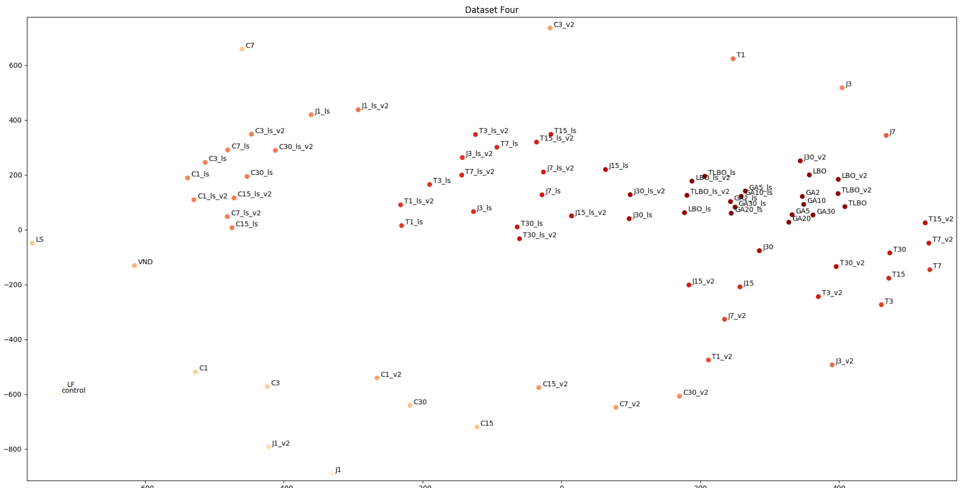
The size of this matrix increases quadratically to the amount of metaheuristics tested. It quickly becomes incomprehensible to look at even a small matrix:

Table 3: Distance Matrix

ds 5	GA2	GA30	T5ls	TL1ls	T10	TL1	TL20ls	jaya	GA20	LS	T20	LF	T30ls	GA10ls	TL10	T5	jayals	GA5ls	TL10ls	GA2ls	GA10	GA20ls	GA5	T1ls	T30	LBOls	TL30ls	TL5ls	TL20	TL30	T10ls	T20ls	GA30ls	T1	TL5	LBO	VND	G2JG4TLls	control	G2JG4TL
GA2	0.0	599.0	846.0	694.0	1911.0	672.0	677.0	826.0	594.0	1864.0	1851.0	2962.0	787.0	612.0	667.0	1942.0	700.0	613.0	683.0	663.0	595.0	617.0	597.0	924.0	1775.0	662.0	674.0	685.0	660.0	645.0	832.0	808.0	608.0	2166.0	656.0	616.0	1795.0	635.0	2988.0	597.0
GA30	599.0	0.0	865.0	746.0	1867.0	666.0	728.0	832.0	600.0	1912.0	1808.0	2968.0	811.0	676.0	648.0	1895.0	758.0	674.0	735.0	724.0	605.0	680.0	596.0	960.0	1731.0	728.0	721.0	734.0	655.0	648.0	852.0	830.0	670.0	2168.0	638.0	631.0	1836.0	703.0	2994.0	605.0
T5ls	846.0	865.0	0.0	820.0	1842.0	869.0	814.0	1012.0	854.0	1823.0	1780.0	2874.0	820.0	814.0	860.0	1874.0	865.0	815.0	812.0	836.0	840.0	813.0	849.0	932.0	1708.0	839.0	806.0	817.0	860.0	852.0	839.0	825.0	814.0	2089.0	857.0	858.0	1754.0	826.0	2900.0	851.0
TL1ls	694.0	746.0	820.0	0.0	2010.0	809.0	643.0	886.0	745.0	1722.0	1948.0	2901.0	774.0	587.0	812.0	2040.0	651.0	587.0	640.0	613.0	734.0	588.0	719.0	864.0	1875.0	611.0	636.0	642.0	807.0	802.0	816.0	793.0	586.0	2157.0	801.0	731.0	1672.0	605.0	2928.0	719.0
T10	1911.0	1867.0	1842.0	2010.0	0.0	1720.0	2008.0	1991.0	1847.0	2671.0	1046.0	2789.0	1856.0	2038.0	1714.0	1089.0	2075.0	2042.0	2005.0	2072.0	1853.0	2043.0	1888.0	2008.0	1060.0	2076.0	1995.0	2005.0	1710.0	1704.0	1839.0	1842.0	2040.0	1991.0	1721.0	1892.0	2547.0	2055.0	2801.0	1886.0
TL1	672.0	666.0	869.0	809.0	1720.0	0.0	791.0	885.0	638.0	1978.0	1658.0	2962.0	816.0	762.0	610.0	1760.0	843.0	766.0	794.0	807.0	638.0	763.0	665.0	1005.0	1581.0	815.0	784.0	799.0	614.0	608.0	856.0	834.0	762.0	2151.0	603.0	682.0	1890.0	788.0	2989.0	668.0
TL20ls	677.0	728.0	814.0	643.0	2008.0	791.0	0.0	872.0	726.0	1730.0	1944.0	2904.0	765.0	574.0	797.0	2035.0	644.0	574.0	630.0	601.0	715.0	575.0	706.0	860.0	1869.0	600.0	625.0	633.0	796.0	783.0	808.0	786.0	575.0	2154.0	783.0	717.0	1678.0	590.0	2931.0	703.0
jaya	826.0	832.0	1012.0	886.0	1991.0	885.0	872.0	0.0	844.0	1888.0	1931.0	2963.0	966.0	827.0	895.0	2018.0	881.0	829.0	876.0	857.0	829.0	822.0	842.0	1069.0	1864.0	848.0	870.0	875.0	885.0	889.0	1003.0	982.0	822.0	2219.0	884.0	839.0	1826.0	848.0	2987.0	841.0
GA20	594.0	600.0	854.0	745.0	1847.0	638.0	726.0	844.0	0.0	1927.0	1786.0	2968.0	798.0	680.0	640.0	1875.0	765.0	675.0	727.0	726.0	599.0	679.0	594.0	962.0	1707.0	731.0	717.0	738.0	639.0	639.0	839.0	818.0	669.0	2163.0	637.0	627.0	1848.0	710.0	2993.0	603.0
LS	1864.0	1912.0	1823.0	1722.0	2671.0	1978.0	1730.0	1888.0	1927.0	0.0	2635.0	1843.0	1818.0	1725.0	1995.0	2667.0	1709.0	1723.0	1729.0	1695.0	1915.0	1723.0	1898.0	1726.0	2592.0	1695.0	1733.0	1726.0	1995.0	1989.0	1828.0	1828.0	1726.0	2325.0	1987.0	1884.0	992.0	1710.0	1852.0	1891.0
T20	1851.0	1808.0	1780.0	1948.0	1046.0	1658.0	1944.0	1931.0	1786.0	2635.0	0.0	2800.0	1793.0	1976.0	1649.0	1096.0	2010.0	1980.0	1944.0	2007.0	1789.0	1979.0	1821.0	1948.0	1034.0	2014.0	1932.0	1942.0	1646.0	1641.0	1778.0	1779.0	1977.0	1978.0	1659.0	1827.0	2513.0	1992.0	2813.0	1828.0
LF	2962.0	2968.0	2874.0	2901.0	2789.0	2962.0	2904.0	2963.0	2968.0	1843.0	2800.0	0.0	2882.0	2932.0	2966.0	2750.0	2924.0	2931.0	2903.0	2918.0	2968.0	2931.0	2967.0	2864.0	2814.0	2919.0	2908.0	2903.0	2967.0	2963.0	2875.0	2881.0	2933.0	2529.0	2966.0	2959.0	1754.0	2924.0	38.0	2965.0
T30ls	787.0	811.0	820.0	774.0	1856.0	816.0	765.0	966.0	798.0	1818.0	1793.0	2882.0	0.0	760.0	815.0	1888.0	817.0	760.0	767.0	785.0	787.0	761.0	790.0	914.0	1720.0	790.0	758.0	767.0	813.0	800.0	813.0	796.0	758.0	2109.0	806.0	806.0	1750.0	772.0	2909.0	794.0
GA10ls	612.0	676.0	814.0	587.0	2038.0	762.0	574.0	827.0	680.0	1725.0	1976.0	2932.0	760.0	0.0	766.0	2066.0	565.0	474.0	576.0	512.0	668.0	476.0	645.0	849.0	1902.0	515.0	568.0	581.0	767.0	757.0	809.0	783.0	471.0	2181.0	758.0	662.0	1677.0	495.0	2958.0	639.0
TL10	667.0	648.0	860.0	812.0	1714.0	610.0	797.0	895.0	640.0	1995.0	1649.0	2966.0	815.0	766.0	0.0	1750.0	842.0	762.0	800.0	818.0	637.0	765.0	665.0	1004.0	1572.0	816.0	788.0	801.0	598.0	587.0	850.0	825.0	761.0	2170.0	597.0	672.0	1903.0	790.0	2992.0	663.0
T5	1942.0	1895.0	1874.0	2040.0	1089.0	1760.0	2035.0	2018.0	1875.0	2667.0	1096.0	2750.0	1888.0	2066.0	1750.0	0.0	2103.0	2071.0	2033.0	2100.0	1887.0	2069.0	1913.0	2038.0	1115.0	2105.0	2024.0	2033.0	1741.0	1742.0	1869.0	1874.0	2070.0	1988.0	1755.0	1922.0	2548.0	2084.0	2761.0	1921.0
jayals	700.0	758.0	865.0	651.0	2075.0	843.0	644.0	881.0	765.0	1709.0	2010.0	2924.0	817.0	565.0	842.0	2103.0	0.0	561.0	647.0	585.0	747.0	563.0	724.0	885.0	1938.0	588.0	640.0	644.0	844.0	836.0	860.0	837.0	568.0	2185.0	833.0	740.0	1666.0	579.0	2951.0	727.0
GA5ls	613.0	674.0	815.0	587.0	2042.0	766.0	574.0	829.0	675.0	1723.0	1980.0	2931.0	760.0	474.0	762.0	2071.0	561.0	0.0	574.0	513.0	676.0	471.0	643.0	849.0	1906.0	512.0	570.0	583.0	766.0	762.0	809.0	785.0	471.0	2181.0	754.0	660.0	1674.0	496.0	2958.0	648.0
TL10ls	683.0	735.0	812.0	640.0	2005.0	794.0	630.0	876.0	727.0	1729.0	1944.0	2903.0	767.0	576.0	800.0	2033.0	647.0	574.0	0.0	602.0	716.0	575.0	708.0	860.0	1870.0	607.0	624.0	638.0	794.0	787.0	808.0	787.0	573.0	2159.0	788.0	714.0	1677.0	589.0	2930.0	709.0
GA2ls	663.0	724.0	836.0	613.0	2072.0	807.0	601.0	857.0	726.0	1695.0	2007.0	2918.0	785.0	512.0	818.0	2100.0	585.0	513.0	602.0	0.0	714.0	511.0	694.0	853.0	1937.0	541.0	594.0	607.0	818.0	809.0	832.0	810.0	511.0	2174.0	801.0	707.0	1653.0	530.0	2945.0	688.0
GA10	595.0	605.0	840.0	734.0	1853.0	638.0	715.0	829.0	599.0	1915.0	1789.0	2968.0	787.0	668.0	637.0	1887.0	747.0	676.0	716.0	714.0	0.0	663.0	582.0	952.0	1713.0	719.0	709.0	725.0	646.0	629.0	833.0	807.0	663.0	2171.0	625.0	619.0	1838.0	695.0	2994.0	597.0
GA20ls	617.0	680.0	813.0	588.0	2043.0	763.0	575.0	822.0	679.0	1723.0	1979.0	2931.0	761.0	476.0	765.0	2069.0	563.0	471.0	575.0	511.0	663.0	0.0	651.0	846.0	1905.0	511.0	570.0	581.0	768.0	758.0	810.0	784.0	471.0	2178.0	753.0	658.0	1675.0	495.0	2958.0	643.0
GA5	597.0	596.0	849.0	719.0	1888.0	665.0	706.0	842.0	594.0	1898.0	1821.0	2967.0	790.0	645.0	665.0	1913.0	724.0	643.0	708.0	694.0	582.0	651.0	0.0	944.0	1746.0	692.0	696.0	707.0	659.0	647.0	840.0	812.0	646.0	2168.0	646.0	615.0	1820.0	673.0	2992.0	596.0
T1ls	924.0	960.0	932.0	864.0	2008.0	1005.0	860.0	1069.0	962.0	1726.0	1948.0	2864.0	914.0	849.0	1004.0	2038.0	885.0	849.0	860.0	853.0	952.0	846.0	944.0	0.0	1878.0	853.0	861.0	862.0	1005.0	989.0	932.0	923.0	847.0	2084.0	998.0	947.0	1682.0	851.0	2890.0	949.0
T30	1775.0	1731.0	1708.0	1875.0	1060.0	1581.0	1869.0	1864.0	1707.0	2592.0	1034.0	2814.0	1720.0	1902.0	1572.0	1115.0	1938.0	1906.0	1870.0	1937.0	1713.0	1905.0	1746.0	1878.0	0.0	1940.0	1858.0	1869.0	1571.0	1565.0	1704.0	1706.0	1902.0	1979.0	1581.0	1752.0	2469.0	1919.0	2830.0	1748.0
LBOls	662.0	728.0	839.0	611.0	2076.0	815.0	600.0	848.0	731.0	1695.0	2014.0	2919.0	790.0	515.0	816.0	2105.0	588.0	512.0	607.0	541.0	719.0	511.0	692.0	853.0	1940.0	0.0	598.0	609.0	815.0	809.0	835.0	812.0	511.0	2179.0	805.0	704.0	1652.0	527.0	2947.0	695.0
TL30ls	674.0	721.0	806.0	636.0	1995.0	784.0	625.0	870.0	717.0	1733.0	1932.0	2908.0	758.0	568.0	788.0	2024.0	640.0	570.0	624.0	594.0	709.0	570.0	696.0	861.0	1858.0	598.0	0.0	629.0	783.0	772.0	803.0	777.0	566.0	2157.0	777.0	709.0	1682.0	586.0	2935.0	697.0
TL5ls	685.0	734.0	817.0	642.0	2005.0	799.0	633.0	875.0	738.0	1726.0	1942.0	2903.0	767.0	581.0	801.0	2033.0	644.0	583.0	638.0	607.0	725.0	581.0	707.0	862.0	1869.0	609.0	629.0	0.0	797.0	789.0	812.0	785.0	578.0	2155.0	793.0	722.0	1674.0	596.0	2930.0	712.0
TL20	660.0	655.0	860.0	807.0	1710.0	614.0	796.0	885.0	639.0	1995.0	1646.0	2967.0	813.0	767.0	598.0	1741.0	844.0	766.0	794.0	818.0	646.0	768.0	659.0	1005.0	1571.0	815.0	783.0	797.0	0.0	596.0	849.0	826.0	760.0	2166.0	604.0	694.0	1902.0	792.0	2994.0	666.0
TL30	645.0	648.0	852.0	802.0	1704.0	608.0	783.0	889.0	639.0	1989.0	1641.0	2963.0	800.0	757.0	587.0	1742.0	836.0	762.0	787.0	809.0	629.0	758.0	647.0	989.0	1565.0	809.0	772.0	789.0	596.0	0.0	842.0	816.0	758.0	2160.0	585.0	683.0	1895.0	786.0	2989.0	657.0
T10ls	832.0	852.0	839.0	816.0	1839.0	856.0	808.0	1003.0	839.0	1828.0	1778.0	2875.0	813.0	809.0	850.0	1869.0	860.0	809.0	808.0	832.0	833.0	810.0	840.0	932.0	1704.0	835.0	803.0	812.0	849.0	842.0	0.0	816.0	810.0	2093.0	844.0	847.0	1757.0	821.0	2901.0	843.0
T20ls	808.0	830.0	825.0	793.0	1842.0	834.0	786.0	982.0	818.0	1828.0	1779.0	2881.0	796.0	783.0	825.0	1874.0	837.0	785.0	787.0	810.0	807.0	784.0	812.0	923.0	1706.0	812.0	777.0	785.0	826.0	816.0	816.0	0.0	783.0	2103.0	820.0	822.0	1757.0	798.0	2908.0	814.0
GA30ls	608.0	670.0	814.0	586.0	2040.0	762.0	575.0	822.0	669.0	1726.0	1977.0	2933.0	758.0	471.0	761.0	2070.0	568.0	471.0	573.0	511.0	663.0	471.0	646.0	847.0	1902.0	511.0	566.0	578.0	760.0	758.0	810.0	783.0	0.0	2175.0	755.0	660.0	1679.0	496.0	2960.0	648.0
T1	2166.0	2168.0	2089.0	2157.0	1991.0	2151.0	2154.0	2219.0	2163.0	2325.0	1978.0	2529.0	2109.0	2181.0	2170.0	1988.0	2185.0	2181.0	2159.0	2174.0	2171.0	2178.0	2168.0	2084.0	1979.0	2179.0	2157.0	2155.0	2166.0	2160.0	2093.0	2103.0	2175.0	0.0	2166.0	2171.0	2299.0	2179.0	2534.0	2177.0
TL5	656.0	638.0	857.0	801.0	1721.0	603.0	783.0	884.0	637.0	1987.0	1659.0	2966.0	806.0	758.0	597.0	1755.0	833.0	754.0	788.0	801.0	625.0	753.0	646.0	998.0	1581.0	805.0	777.0	793.0	604.0	585.0	844.0	820.0	755.0	2166.0	0.0	662.0	1895.0	779.0	2992.0	653.0
LBO	616.0	631.0	858.0	731.0	1892.0	682.0	717.0	839.0	627.0	1884.0	1827.0	2959.0	806.0	662.0	672.0	1922.0	740.0	660.0	714.0	707.0	619.0	658.0	615.0	947.0	1752.0	704.0	709.0	722.0	694.0	683.0	847.0	822.0	660.0	2171.0	662.0	0.0	1809.0	683.0	2985.0	622.0
VND	1795.0	1836.0	1754.0	1672.0	2547.0	1890.0	1678.0	1826.0	1848.0	992.0	2513.0	1754.0	1750.0	1677.0	1903.0	2548.0	1666.0	1674.0	1677.0	1653.0	1838.0	1675.0	1820.0	1682.0	2469.0	1652.0	1682.0	1674.0	1902.0	1895.0	1757.0	1757.0	1679.0	2299.0	1895.0	1809.0	0.0	1665.0	1777.0	1816.0
G2JG4TLls	635.0	703.0	826.0	605.0	2055.0	788.0	590.0	848.0	710.0	1710.0	1992.0	2924.0	772.0	495.0	790.0	2084.0	579.0	496.0	589.0	530.0	695.0	495.0	673.0	851.0	1919.0	527.0	586.0	596.0	792.0	786.0	821.0	798.0	496.0	2179.0	779.0	683.0	1665.0	0.0	2951.0	678.0
control	2988.0	2994.0	2900.0	2928.0	2801.0	2989.0	2931.0	2987.0	2993.0	1852.0	2813.0	38.0	2909.0	2958.0	2992.0	2761.0	2951.0	2958.0	2930.0	2945.0	2994.0	2958.0	2992.0	2890.0	2830.0	2947.0	2935.0	2930.0	2994.0	2989.0	2901.0	2908.0	2960.0	2534.0	2992.0	2985.0	1777.0	2951.0	0.0	2991.0
G2JG4TL	597.0	605.0	851.0	719.0	1886.0	668.0	703.0	841.0	603.0	1891.0	1828.0	2965.0	794.0	639.0	663.0	1921.0	727.0	648.0	709.0	688.0	597.0	643.0	596.0	949.0	1748.0	695.0	697.0	712.0	666.0	657.0	843.0	814.0	648.0	2177.0	653.0	622.0	1816.0	678.0	2991.0	0.0

An algorithm called Multi-Dimensional Scaling will attempt to find a low dimensional mapping of a cost matrix such that the sum of the squares of the differences of the distances between points among the high and low dimensional space is minimized. The MDS graph is much more comprehensible:

[2]See https://bit.ly/2wPh9mQ for a visualization



This is the graph for only dataset four, as generating the cost matrix is quite computationally expensive.

The darkness of the color signifies the relative performance of the best scoring solution: 100% dark signifies the metaheuristic had the best performing solution, and 0% dark is the worst performing. The other colors are scaled linearly.

6 Choosing Parameters

A large search of many different parameter combinations, as has been done to calculate diversities and the MDS graph, requires testing of nearly 80 different metaheuristic instances. Practical reasons therefore limit the amount of time each metaheuristic is allowed to take—so far, each algorithm has been given 4 seconds of run time. A 60 second time limit will more accurately match practical applications of these techniques. Therefore, it is necessary to choose a subset of promising metaheuristics to test.

Metaheuristics worth testing will satisfy at least one of two criteria: they will give competetive results, or they will have unique behavior. The MDS graph will show metaheuristics with unique behavior, and a complete search of a few datasets will give an indication of performance. See Table 4.

We see that the Genetic Algorithm beats out every other metaheuristic, regardless of its parameters. It is followed by TLBO and LBO. Unweighted LBO should be very similar to the Genetic Algorithm with two parents, the only difference lies in how parents are selected from the population, so it is surprising to see it this low in the list. Next lies Jaya and TBO with high

Table 4: Fast Large Percentage Search

algorithm	ds4 percentages
GA2_ls	0.78
GA5_ls	0.79
GA10_ls	0.86
GA20_ls	0.88
GA30_ls	0.89
GA2	0.98
GA20	1.09
GA5	1.13
GA10	1.13
GA30	1.16
TLBO_ls	1.59
TLBO_ls_v2	1.63
LBO_ls	1.68
TLBO	1.84
LBO_ls_v2	1.86
TLBO_v2	1.9
LBO	2.08
LBO_v2	2.17
J30_ls	2.9
J30_v2	2.96
J30	3.0
J30_ls_v2	3.0
T30_v2	4.01
J15_ls_v2	4.16
J15_ls	4.21
T15_v2	4.47
T30_ls_v2	4.57
T30	4.73
T30_ls	4.78
T7_v2	5.28
J15_v2	5.37
J7_ls	5.51
T15_ls	5.53
T15_ls_v2	5.65
J7_ls_v2	5.72
T3_v2	5.86
J15	5.88
T15	5.93
T7_ls_v2	6.08
T7_ls	6.1
T3_ls_v2	6.4
T3_ls	6.61
J3_ls	6.62
T7	6.75
J3_ls_v2	6.97
T1_ls	7.14
T1_ls_v2	7.18
J7_v2	7.45
T3	7.61
T1_v2	8.4
J7	8.86
C30_ls_v2	9.84
C30_ls	9.9
J3_v2	9.94
T1	10.03
C7_ls_v2	10.11
C15_ls_v2	10.14
C3_ls_v2	10.25
C15_ls	10.27
J1_ls_v2	10.42
C1_ls_v2	10.46
C7_ls	10.73
C3_ls	10.81
C30_v2	10.82
C1_ls	11.21
C15_v2	11.43
J1_ls	11.61
J3	11.78
C7_v2	11.99
C3_v2	12.49
C1_v2	12.71
C30	14.21
VND	14.93
C15	15.15
C7	15.47
C3	15.93
C1	16.34
LS	16.85
J1_v2	20.08
J1	23.61
LF	23.65
control	24.9

values for m and n. CBO is very poorly performing, and joins the bottom of the list with low valued Jaya and TBO, as well as local search methods with no accompanying global method.

Since the Genetic Algorithm has similar behavior no matter the amount of parents, only one instance needs to be tested. Since 2 parents should be similar to LBO given more time, 5 parents will be tested. A local search will be embedded, as it makes a small difference in behavior shown by the MDS graph, but strictly improves performance, as shown by the table.

TLBO was only tested with a one teacher TBO. Higher teacher values improve performance of TBO, but this may not be true for TLBO. Therefore, TLBO will be tested with one, 15,and 30 teachers for TBO. The MDS graph shows that TLBO will have a similar behavior regardless of local search or weighted/unweighted perturbations. Looking to the performance table shows that unweighted with local search performs slightly better than any other instance, so those are the parameters that will be used for further testing.

Jaya is a metaheuristic that does show a large variation in behavior depending on parameters. The best performing instance has 30 parents, embedded local search, unweighted perturbation, and the most unique instance that is still performant has 3 parents, weighted perturbation, and no local search. Both these instances will be tested.

CBO behaves very different than any other metaheuristic, as shown by the graph. However, the performance is quite bad, so only one instance needs to be tested. CBO with 30 parents, local search, and a weighted perturbation is the top of the table, so those parameters will be further tested.

Local searches without an accompanying global method would not benefit from more search time, as they can run to completion within four seconds.

6.1 Additional Runtime Parameters

Maximum Failed Attempts When testing a metaheuristic, a single run through is often not enough to extract the full benefits of the method. Instead, it is ran repeatedly, until it fails to produce an improvement for any solution in the population $max\_failed\_attempts$ times in a row. This setting will directly impact the run time of the algorithm. For this longer test, a high value of 25 is used.

Time Constraint Each metaheuristic will be given a minute to run. This is a realistic simulation of what would be found in industry. Additionally, some metaheuristic instances can take several seconds to run through one iteration, so a small time value can allot significantly more time to one metaheuristic than another. This effect is mitigated with more time.

Population Size The population generation procedure will be ran until 30 solutions are generated, or 500,00 attempts are made. If less than 30 solutions are found, which can happen for some datasets, metaheuristic optimization is attempted anyway. Local search is embedded into the initial population generation.

7 Results

Below are tables showing the percent deviation from optimal for each algorithm. Each algorithm had the same initial population, generated using an embedded local search. "Control" is therefore equivalent to the VND metaheuristic.

Optimal values were calculated using a CPLEX program given two hours to search for the best solution. For datasets 5, 7, 8, and 9, the metaheuristics found solutions better than the CPLEX reported optimals. These negative percentages result in a misleadingly low optimal.

Additionally, dataset 7 has many problems that are extremely difficult to find feasible solutions for. Therefore, 7 is more of a test of the initial population generation algorithm than any specific metaheuristic. If the initial population generator finds a solution CPLEX did not, it is reported as 0% error, but if the inverse happens, it is 100% error. This leads to a deceptively high average.

Averaging only datasets 1, 2, 3, 4 and 6 from the hybrid method with embedded local search gives an overall lowest percentage of 0.27%.

Table 5: Metaheuristic 10 Second Results

complete percentages	dataset 1	dataset 2	dataset 3	dataset 4	dataset 5	dataset 6	dataset 7	dataset 8	dataset 9	average
GA2_ls	0.09	0.43	1.47	0.3	0.84	2.24	10.78	2.56	3.47	2.46
TLBO_ls	0.13	0.56	1.25	0.44	0.93	1.87	11.26	2.54	3.37	2.48
jaya_ls	0.3	0.49	1.39	0.77	0.83	2.22	11.79	2.42	3.33	2.62
LBO_ls	0.17	0.46	1.65	0.4	0.87	2.5	11.21	2.6	3.94	2.64
GA2	0.29	0.33	0.43	0.64	0.67	0.88	11.08	3.83	6.18	2.7
TBO_ls	0.33	0.89	1.7	0.81	1.11	2.15	12.29	2.9	3.5	2.85
LBO	0.48	0.53	0.53	0.89	0.79	0.98	11.7	3.82	6.31	2.89
TLBO	0.53	0.54	0.96	0.9	0.95	2.01	11.4	6.74	6.73	3.42
jaya	1.5	1.3	1.02	2.26	1.89	1.66	12.73	4.72	5.75	3.65
TBO	2.19	2.94	4.01	2.84	3.3	4.38	13.35	8.3	6.94	5.36
control	6.5	6.67	6.33	6.72	5.76	5.78	15.64	10.14	7.36	7.88

Table 6: Metaheuristic 60 Second Results

complete percentages	dataset 1	dataset 2	dataset 3	dataset 4	dataset 5	dataset 6	dataset 7	dataset 8	dataset 9	average
GA2_ls	0.09	0.17	0.32	0.23	0.35	0.8	10.45	0.31	1.32	1.56
GA2	0.33	0.32	0.29	0.61	0.55	0.47	11.03	0.47	0.66	1.64
LBO_ls	0.16	0.2	0.34	0.4	0.38	0.79	11.15	0.51	1.31	1.69
TLBO_ls	0.12	0.23	0.46	0.42	0.44	0.8	11.02	0.66	1.32	1.72
jaya_ls	0.35	0.33	0.38	0.69	0.54	0.82	11.6	0.85	1.18	1.86
LBO	0.51	0.55	0.49	0.93	0.82	0.73	11.71	0.81	0.72	1.92
TLBO	0.51	0.55	0.63	0.9	0.82	0.75	11.56	1.02	2.08	2.09
TBO_ls	0.37	0.53	0.97	0.77	0.7	1.18	12.06	1.42	1.68	2.19
jaya	1.34	1.16	1.05	2.16	1.84	1.51	12.63	3.49	2.16	3.04
TBO	2.27	2.95	3.66	2.96	3.2	3.8	13.23	5.97	6.17	4.91
control	6.54	6.56	6.37	6.9	5.95	5.73	15.59	10.02	7.38	7.89

Table 7: Hybrid 60 Second Results

complete percentages	dataset 1	dataset 2	dataset 3	dataset 4	dataset 5	dataset 6	dataset 7	dataset 8	dataset 9	average
G2JG4TL_ls	0.06	0.15	0.3	0.14	0.33	0.7	11.22	0.29	1.14	1.59
G2JG4TL	0.17	0.24	0.26	0.48	0.46	0.46	11.61	0.4	0.63	1.64
control	6.25	6.52	6.31	7.1	6.03	5.9	16.2	9.81	7.11	7.91

8 Creating a Hybrid Method

A hybrid method consisting of many metaheurisitcs chained together has the potential to outperform any one metaheuristic, but the order of metaheuristics must be chosen with care. Putting a strongly intensifying metaheuristic too early in the chain can cripple the population for the next metaheuristic. This can be somewhat mitigated by running each metaheuristic only once before looping through the chain, as opposed to running one technique to exhaustion before moving on to the next. However, the order can still have a significant effect.

Using the table of diversities, as well as the general principle that intensification and diversification should oscillate, and the goal of including each metaheuristic only once, the patterns TLJB, TBJL, JBTL, and TLJB result. Keeping TBO and LBO consecutive due to the effectiveness of the TLBO method, and including the GA twice, once with two parents, once with four parents, gives the chain GA[parents=2], Jaya, GA[parents=4], TBO, LBO.

8.1 Additional Visualizations

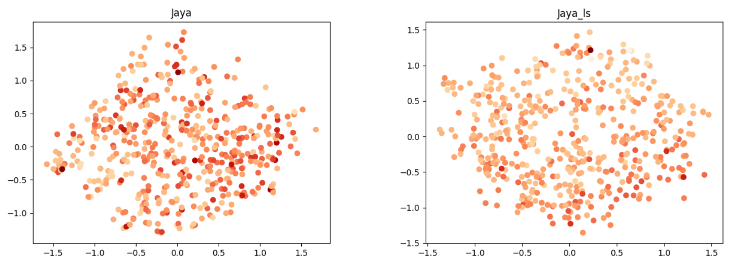


Figure 3: Jaya and Jaya Local Search

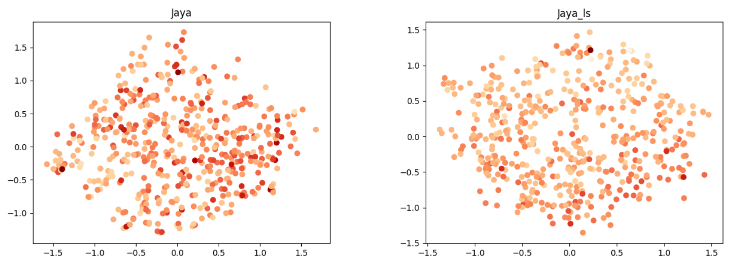


Figure 4: Jaya and Jaya Local Search

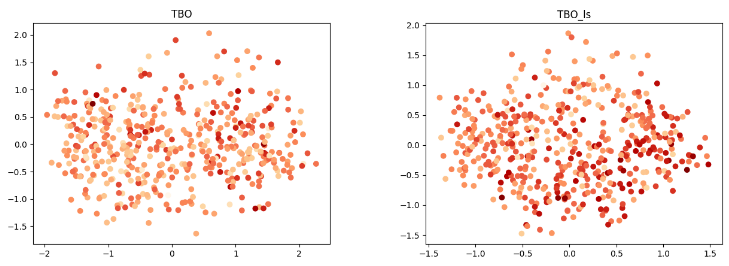


Figure 5: TBO and TBO Local Search

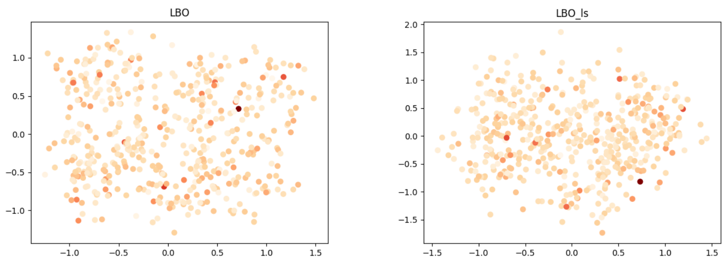


Figure 6: LBO and LBO Local Search

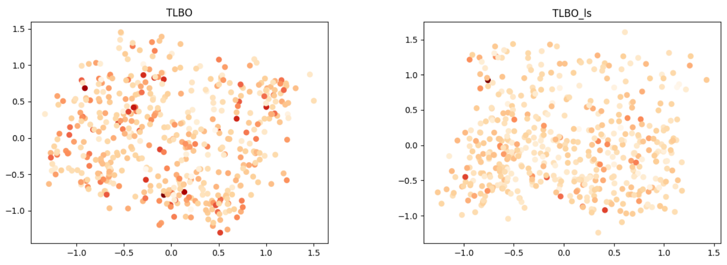


Figure 7: TLBO and TLBO Local Search

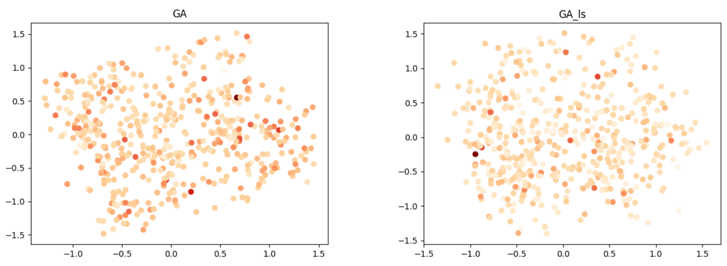


Figure 8: GA and GA Local Search

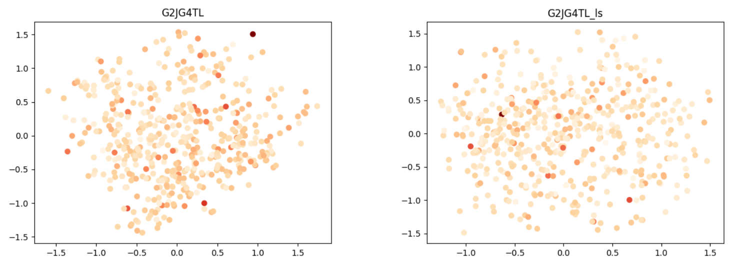


Figure 9: G2JG4TL and G2JG4TL Local Search