Generative Adversarial Construction of Parallel Portfolios

摘要

由于自动算法配置方法的效能不断提升，越来越多的研究旨在利用这些方法以实现自动求解器构造，产生了数种值得关注的方法。这些方法都是建立在给定的训练集可以充分表示目标用例的假设前提下，使得构造的求解器具有较好的泛化能力。然而现实中，这样的假设往往不能成立，所能提供的训练数据可能相当少且不具代表性。本文就旨在研究如何在训练数据缺乏且不具代表性的情况下，高效的构造并行算法组。和先前提出的方法不同，本文中所提出的方法在一个对抗的过程中同时考虑了实例生成和算法组构造，前者旨在生成对当前算法组来说更具挑战的实例，后者则旨在为算法组找到新的组件求解器以更好的解决新生成的实例。所提出的方法在训练数据稀缺的情况下，所构造的并行算法组比现有方法所构造的算法组有着更好的泛化能力和性能，甚至能够和最新的手动设计的并行算法组所媲美。

关键词：自动算法组构造、生成对抗方法、并行算法组、参数调优。

1 引言

现有的许多用于解决计算困难问题的高性能算法，往往都包含了大量需要仔细调节的参数，才能实现算法的最优性能。一般来说，找到最优性能的参数设置一般是手动完成的。然而，手动调节的不足之处在于，该过程需要大量的人力成本且往往限制在对几个参数设置的探索，使得其性能无法达到最优。因此，大量的研究正在尝试使该过程自动化，也就是自动算法配置（AAC）[1]。在过去几年中，几种高性能的自动算法配置器如 ParamILS [2]、GGA [3]、irace [4] 和 SMAC [5] 被提出，大大的提升了 AAC 的效率。基于 AAC 方法的快速发展，自动算法配置器也被用于构造序列求解器、序列算法组和并行算法组。并行算法组，表示的是在解决问题实例时，一组独立且并行运行求解器。并行算法组的优势在于在解决计算困难的问题时，使用一组求解器是较为简单且有效的方法，并且，得益于并行计算架构的发展，使得这种并行性更易实现。在算法配置的过程中，往往需要训练集（研究问题领域的一组问题实例）以实现求解器的构造。普遍的观点都认为所提供的训练集需要具有较强的代表性，使得构造的求解器在该问题领域中有着较好的泛化能力。然而在训练实例的数量较少且无法覆盖可能的目标用例，又或者所给的训练实例已经过时，无法充分反映当前实例属性的情况时，训练集往往无法具备较强的代表性。这样的情况在现实生活中十分常见。为了解决这个问题，作者提出了一个新的方法生成对抗求解训练器（GAST）[6] 实现自动算法组的构造。和现有

的方法不同，GAST 会生成额外的训练实例并在动态变化的训练集上构造并行算法组。具体来说，GAST 将实例生成和算法组构造置于对抗博弈中，实例生成的目的是生成当前组合不能很好地解决的困难实例，而算法组组构造的目的是为算法组寻找新的组件求解器，以更好地解决这些具有挑战性的实例。在这个博弈中的竞争驱使算法组解决更多的问题实例，从而获得更好的泛化性能。本次课程的论文复现工作便是将 GAST 方法进行复现，实现方法中所使用的对抗博弈过程。

2 相关工作

2.1 自动算法配置

自动算法配置的基本框架如图 1所示。自动算法配置的具体过程可以描述如下：给定一个参数化算法 A 及其配置空间 Θ，一组问题训练实例 I $=i_1$, $\cdots$, i_n 和性能指标$m:\times I \rightarrow R$, 算法配置的目的是找到一个参数配置 c Θ，使得算法 A 的在训练实例 I 上的性能最佳。

$$f(c)=\frac{1}{|I|}\sum_{i\in I} m(i,c) \tag{1}$$

现有的算法配置的方法主要有 ParamILS、GGA、irace 和 SMAC。这几种方法都可以看作是共享一个共同的迭代搜索框架，即迭代地生成和测试候选配置。最大的不同点在于他们生成候选配置的方法。在搜索配置空间时，ParamILS 和 GGA 使用的都是直接搜索的方法，ParamILS 采用了迭代局部搜索算法，GGA 则采用了基于性别机制的遗传算法。而 SMAC 和 irace 则依赖于构建元模型来指引配置空间的采样。

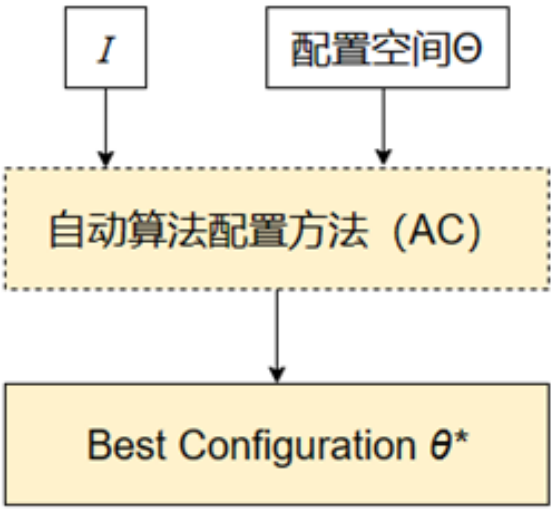

图 1. 自动算法配置基本框架

2.2 自动算法组构造

自动算法组构造 [7] 是基于自动算法配置实现的，通过自动算法配置不断配置算法并加入到算法组中，使得最终生成的算法组中的算法有着更强的互补性，以提升算法组在解决问题实例时的性能。自动算法组构造的过程可以描述如下，给定一个参数化算法 A 及其配置

空间 Θ，一组问题训练实例 $I = i_1, ..., i_n$ 和性能指标 m，算法组构造的目的是找到一组算法 $P = p_1, ..., p_k$ 使得其在训练实例上的平均性能最佳（假设 $m(i, p)$ 越小，性能越好）。

$$f(P) = \frac{1}{|I|} \sum_{i \in I} \min_{p \in P} m(i, c) \tag{2}$$

自动算法组构造的基本框架如图 2所示。目前已有的自动算法组构造的方法主要有 PARHYDRA、GLOBAL、CLUSTERING 和 PCIT。GLOBAL 方法通过同时运行 n 组算法配置器（AC），生成 n 组算法，再将其中性能最优的一组选为最终的算法组。该方法过程较为简单，但一次为算法组配置 k 个算法会导致其配置空间会随着算法组中算法的数量而呈指数型增长，使得计算开销增大。PARHYDRA 方法与 GLOBAL 方法同样运行了 n 组算法配置器，但不同之处在于，算法配置器每次生成 n 个配置，将其中性能最优的算法配置添加到算法组中，并重复上述过程 k 次，生成最终的算法组。PARHYDRA 方法消除了 GLOBAL 中配置空间过大的问题，但由于其采用的贪心算法策略，使得其存在无法充分考虑组件潜在优势的问题。CLUSTERING 方法和 PCIT 方法都采用了实例聚类的方法，即先将训练实例分为 k 个子集，再为每个子集配置最优的算法，得到最终的算法组。不同之处在于，CLUSTERING 会先计算各实例的归一化特征，再根据归一化特征值将实例聚类。而 PCIT 则是先将训练实例随机分为 k 个子集，再运行 k 个算法配置器，每个配置器为一个子集找到最优的算法并添加到算法组中，在配置的过程中，如果 AC 无法为子集中的每个实例找到通用的高性能配置，则可认为该子集中的部分实例需要转移至其他子集中，再将这部分实例转移到其他的子集中，重复上述配置过程，直到满足程序的终止条件。但这几类方法都是建立在训练实例较为充足且具有代表性的前提下所提出的，并未考虑到现实场景下训练实例数据不充足的情况。

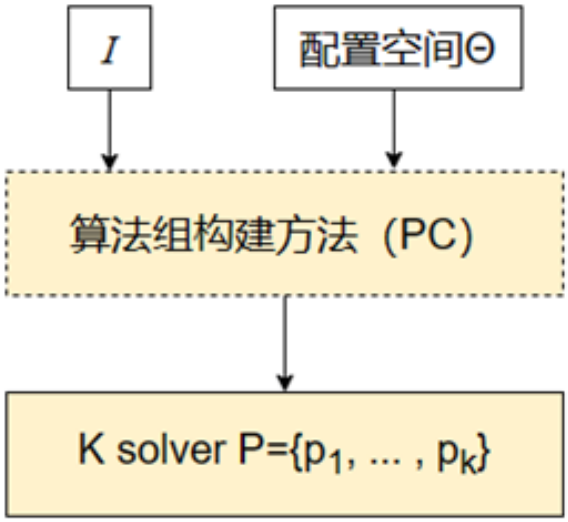

图 2. 方法示意图

2.3 生成对抗网络（GAN）

[?] 生成对抗网络（GAN）[8] 是两个网络的组合：生成网络（Generator）负责生成模拟数据；判别网络 Discriminator）负责判断输入的数据是真实的还是生成的。生成网络要不断优化自己生成的数据让判别网络判断不出来，判别网络也要优化自己让自己判断得更准确。GAN 网络的整体示意图如图 3所示。

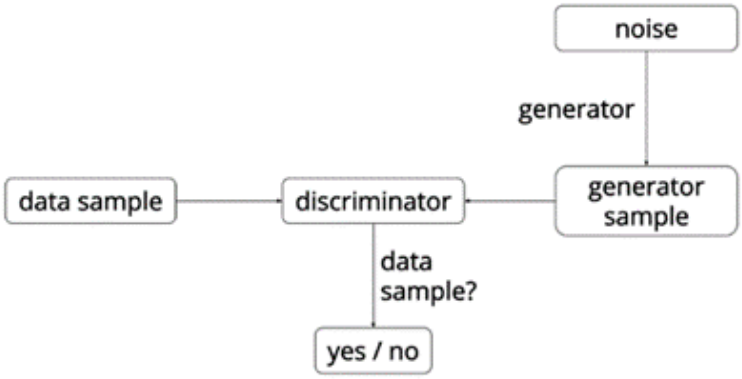

图 3. GAN 网络示意图

基于 GAN 网络的思想，通过模拟该生成对抗的过程，不断生成更多的训练实例，消除训练实例数据不足的问题，提升最终构造的算法组的性能。

3 本文方法

3.1 GAST

总的来说，GAST 方法中有两个关键的设计原则。第一个需要关注的是生成有用的训练实例。如果一个训练集不具代表性，那么则意味着部分的目标用例未被覆盖，需要生成额外的训练实例。此外，对于那些不存在于训练集中的问题实例，可能在已经构造了的算法组中有能够很好处理它们的求解器，因此这些实例尽管不在训练集中，但它们对提升算法组的性能并没有帮助。因此，需要生成的实例应该是不存在于训练集中，且对已经构造了的算法组来说难以解决的。第二个关键的原则是算法组中各组件求解器之间的互补性。互补性在并行算法组中是至关重要的，它决定了一个算法组的性能效率。前面提到过，并行算法组在实例上的性能取决于在该实例上性能最优的组件求解器。根据 No Free Lunch theorem，不存在一个在所有问题上都表现极佳的算法。因此，需要有多个组件求解器用于处理不同的实例。GAST 方法的简要流程图如图 4所示。

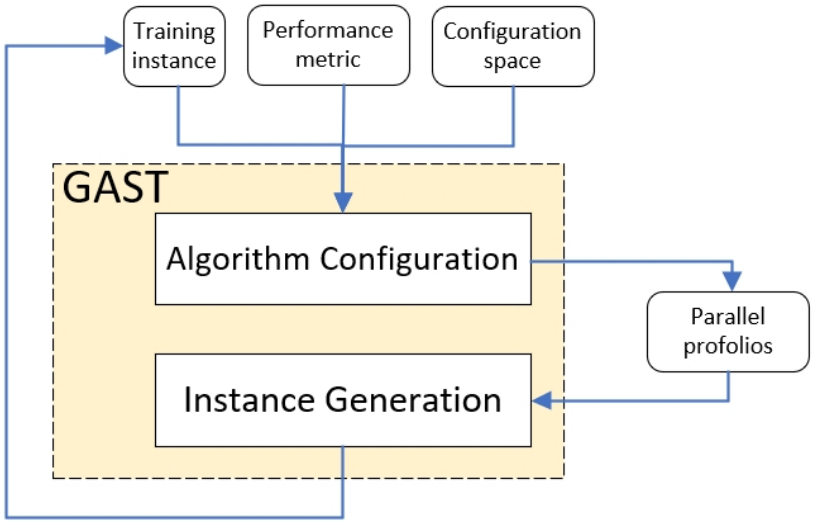

图 4. GAST 流程图

可以看出，本文所提出的算法主要由两个阶段所组成，一个是算法配置阶段，一个是生成实例阶段。本文最大的亮点便是引入了这么一个实例生成的阶段，使得该方法能够应用于实例不足的场景之下，通过已有的实例以生成新的实例。

3.2 伪代码分析

GAST 的伪代码如算法 1 所示。总的来说，GAST 方法具有迭代结构，它的每次迭代都由两个子阶段组成：配置阶段和生成实例阶段。配置阶段和前文中提到的 PARHYDRA 方法相同，在每次的迭代过程中，使用 AC 配置器对 c_i 进行配置，并添加到当前的算法组 $c_{1:i-1}$ 中，使得算法组 $c_{1:i}$ 在实例集 I 上的性能被优化。而后，在生成实例阶段，会对现有训练数据进行扰动，生成新的困难实例，再利用配置阶段时的性能评估数据，将简单实例移除，添加困难实例到训练实例中。重复这两个过程，直到为算法组配置了 k 个求解器。

4 复现细节

4.1 与已有开源代码对比

本篇论文所提的 GAST 方法作者并未提供源代码，通过其他途径，获取到了 GAST 的源代码。图 5提供了 GAST 源代码结构图。

图 5. GAST 代码结构图

其中，包含训练实例的实例集文件 instanceSet，针对特定实例的求解器文件夹 Solver，自动算法配置器 AC，验证结果和配置结果输出的文件夹 validationOutput 和 ACOutput，src 文件夹包含的程序主入口及主要代码。论文中作者介绍了两种实验场景：TSP 和 SAT。TSP 是旅行商问题，SAT 是布尔可满足性问题。在这两个场景下，GAST 方法所使用的实例生成方法是不一样的，针对于 TSP 问题，源代码中给出了对应的实例生成代码，但对于 SAT 问题，源代码中却并未给出，而且相应的 SAT 数据集也并未提供。通过对论文进行阅读，发现作者在 GAST 场景下所使用的实例生成方法是 Spig 方法。再查阅相应论文，找到论文中所

Algorithm 1 GAST

Input: base solvers B with configuration space C; number of component solvers k;instance set I; performance metric m;algorithm configurator AC; independent configurator runs n; time budgets t_C, t_V, t_I for configuration, validation and instance generation respectively

Output: parallel portfolio $c_{1:k}$

for $i \leftarrow 1 : k$ **do**
  /*——————**configuration phase**——————*/
  for $j \leftarrow 1 : n$ **do**
    obtain a portfolio $c^j_{1:i}$ by running AC on configuration space $c_{1:i-1} \times c | c \in C$using m for time t_C
  end for
  validate $c^1_{1:i}, ..., c^n_{1:i}$on I using m for time t_V
  let $c_{1:i} \leftarrow arg\min_{c^j_{1:i} | j \in \{1,...,n\}} P(c^j_{1:i}, I)$ be the portfolio with the best validation performance
  /*——————**instance-generation phase**——————*/
  if $i = k$ **then break** //skip instance generation
  according to the validation results, assign the quality score of each $s \in I$ as $w_s \cdot P(c_{1:i}, s)$
  $\overline{I} \leftarrow I$
  while time spent in this phase not exceeds t_I **do**
    $I_{new} \leftarrow \emptyset$
    for each $s \in I$ **do**
      $refset \leftarrow$randomly sample from $I \setminus \{s\}$
      $s_{new} \leftarrow$ **variation**$(s, refset)$
      $I_{new} \leftarrow I_{new} \bigcup \{s_{new}\}$
    end for
    test $c_{1:i}$ with each $s \in I_{new}$ and assign the quality score of s as $w_s \cdot P(c_{1:i}, s)$
    $I \leftarrow I \bigcup I_{new}$
    remove $|I_{new}|$ instances from I with binary tournament selection
  end while
  $I \leftarrow I \bigcup \overline{I}$
end for
return $c_{1:k}$

提到的 Spig 方法，并找到其源代码，将其嵌入 GAST 场景中，便实现了用 GAST 解决 SAT 问题。源代码和补充后的代码结构对比如图 6所示。

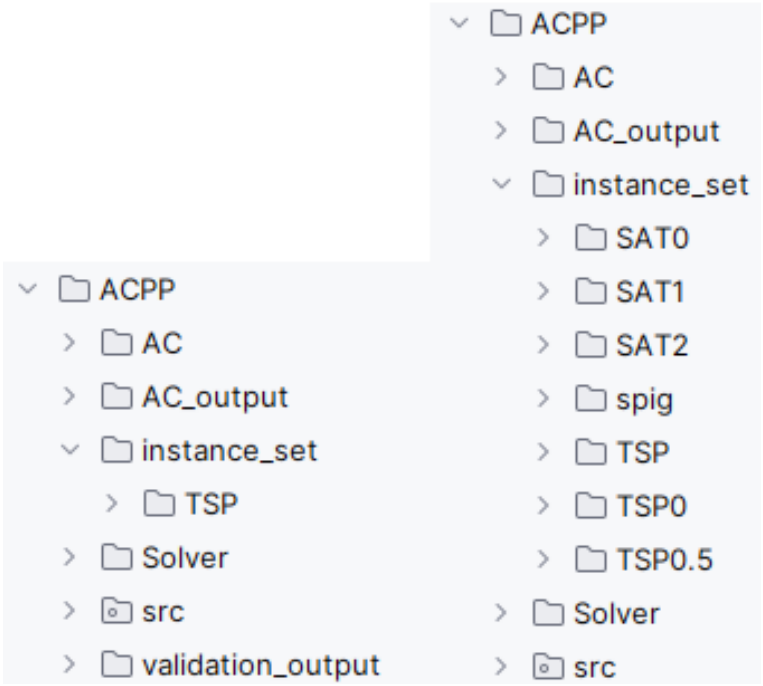

图 6. 前后对比图，左边为源代码，右边为补充后

4.2 实验环境搭建

硬件要求：Windows10 系统的计算机一台，服务器。软件要求：Python2.7，Pycharm。

4.3 使用说明

GAST 方法是基于 python2.7 进行编写的，因此需要配置 python2.7 的项目环境。将代码中相应的文件路径修改为本机路径，而后按照作者给出的设置要求对参数进行设置。如图 7所示。

	Instance Sets	Cut-off Time	Base Solvers
TSP-SMALL	Randomly select 50 instances from TSP_{whole} as training instances and the rest 250 instances are used as test instances	1s	LKH version 2.0.7 [65] with 23 parameters
TSP-BIAS	Randomly select 50 instances from 150 “uniform” instances or 150 “clustering” instances in TSP_{whole} as training instances and the rest 250 instances are used as test instances		
SAT-SMALL	Randomly select 50 instances from SAT_{whole} as training instances and the rest 250 instances are used as test instances	150s	lingeling-ala [66] with 118 parameters
SAT-BIAS	Randomly select 50 instances from 150 “IBM HV” instances or 150 “BMC” instances in SAT_{whole} as training instances and the rest 250 instances are used as test instances		

图 7. 参数配置

针对 TSP 问题，终止时间为 1 秒，算法配置时间、验证时间和实例生成时间分别设置为 1.5 小时、0.5 小时和 5 小时。至于 SAT 问题，终止时间则设置为 150 秒，算法配置时间、验证时间和实例生成时间分别设置为 8 小时、4 小时和 40 小时。值得注意的是，针对 SAT 问

题的实例生成方法 Spig 存在条件要求，对生成的实例会进行评估，若难度过低，该方法会将生成的实例删除。因此，需要对方法的条件进行相应设置，以免发生没有新实例生成的情况。

5 实验结果分析

代码运行结束后，会得到一组算法，其中包含了各算法的配置信息，如图 8所示。

```
-@1:ASCENT_CANDIDATES '47' -@1:BACKBONE_TRIALS '0' -@1:BACKTRACKING 'NO' -@1:CANDIDATE_SET_TYPE 'DELAUNAY'
-@1:EXTRA_CANDIDATES '4' -@1:GAIN23 'YES' -@1:GAIN_CRITERION 'YES' -@1:INITIAL_STEP_SIZE '1'
-@1:INITIAL_TOUR_ALGORITHM 'SIERPINSKI' -@1:INITIAL_TOUR_FRACTION '0.32141709711769402' -@1:KICKS '3'
-@1:KICK_TYPE '4' -@1:LKH:prefix '-' -@1:LKH:solver '/home/liuwei/GAST/ACPP/Solver/LKH-2.0.7/LKH'
-@1:MAX_BREADTH '1816941942' -@1:MAX_CANDIDATES '5' -@1:MOVE_TYPE '2' -@1:PATCHING_A '2' -@1:PATCHING_C '0'
-@1:POPULATION_SIZE '16' -@1:RESTRICTED_SEARCH 'YES' -@1:RUNS '10000' -@1:SUBGRADIENT 'YES'
-@1:SUBSEQUENT_MOVE_TYPE '4' -@1:SUBSEQUENT_PATCHING 'NO' -@2:ASCENT_CANDIDATES '52' -@2:BACKBONE_TRIALS '0'
-@2:BACKTRACKING 'NO' -@2:CANDIDATE_SET_TYPE 'DELAUNAY' -@2:EXTRA_CANDIDATES '4' -@2:GAIN23 'YES'
-@2:GAIN_CRITERION 'YES' -@2:INITIAL_STEP_SIZE '4' -@2:INITIAL_TOUR_ALGORITHM 'GREEDY'
-@2:INITIAL_TOUR_FRACTION '0.578450957078711' -@2:KICKS '2' -@2:KICK_TYPE '5' -@2:LKH:prefix '-'
-@2:LKH:solver '/home/liuwei/GAST/ACPP/Solver/LKH-2.0.7/LKH' -@2:MAX_BREADTH '1339748031' -@2:MAX_CANDIDATES
'5' -@2:MOVE_TYPE '4' -@2:PATCHING_A '1' -@2:PATCHING_C '0' -@2:POPULATION_SIZE '57' -@2:RESTRICTED_SEARCH
'NO' -@2:RUNS '10000' -@2:SUBGRADIENT 'YES' -@2:SUBSEQUENT_MOVE_TYPE '0' -@2:SUBSEQUENT_PATCHING 'NO'
-@3:ASCENT_CANDIDATES '59' -@3:BACKBONE_TRIALS '0' -@3:BACKTRACKING 'NO' -@3:CANDIDATE_SET_TYPE
'NEAREST-NEIGHBOR' -@3:EXTRA_CANDIDATES '5' -@3:GAIN23 'NO' -@3:GAIN_CRITERION 'YES' -@3:INITIAL_STEP_SIZE '3'
-@3:INITIAL_TOUR_ALGORITHM 'NEAREST-NEIGHBOR' -@3:INITIAL_TOUR_FRACTION '0.6128264541037108' -@3:KICKS '4'
-@3:KICK_TYPE '0' -@3:LKH:prefix '-' -@3:LKH:solver '/home/liuwei/GAST/ACPP/Solver/LKH-2.0.7/LKH'
-@3:MAX_BREADTH '1268844799' -@3:MAX_CANDIDATES '6' -@3:MOVE_TYPE '3' -@3:PATCHING_A '3' -@3:PATCHING_C '0'
-@3:POPULATION_SIZE '89' -@3:RESTRICTED_SEARCH 'YES' -@3:RUNS '10000' -@3:SUBGRADIENT 'YES'
-@3:SUBSEQUENT_MOVE_TYPE '0' -@3:SUBSEQUENT_PATCHING 'NO' -@4:ASCENT_CANDIDATES '59' -@4:BACKBONE_TRIALS '0'
-@4:BACKTRACKING 'NO' -@4:CANDIDATE_SET_TYPE 'QUADRANT' -@4:EXTRA_CANDIDATES '5' -@4:GAIN23 'NO'
-@4:GAIN_CRITERION 'YES' -@4:INITIAL_STEP_SIZE '1' -@4:INITIAL_TOUR_ALGORITHM 'SIERPINSKI'
-@4:INITIAL_TOUR_FRACTION '0.45711204791568516' -@4:KICKS '3' -@4:KICK_TYPE '0' -@4:LKH:prefix '-'
-@4:LKH:solver '/home/liuwei/GAST/ACPP/Solver/LKH-2.0.7/LKH' -@4:MAX_BREADTH '1590187023' -@4:MAX_CANDIDATES
'7' -@4:MOVE_TYPE '4' -@4:PATCHING_A '0' -@4:PATCHING_C '2' -@4:POPULATION_SIZE '70' -@4:RESTRICTED_SEARCH
'YES' -@4:RUNS '10000' -@4:SUBGRADIENT 'YES' -@4:SUBSEQUENT_MOVE_TYPE '0' -@4:SUBSEQUENT_PATCHING 'NO'
```

图 8. 实验结果示意

其中 @1 表示的是算法组中的第一个算法，而后便是相应的配置以及该配置的值。可以看出，生成的算法组共包含 4 个算法。

6 总结与展望

通过复现论文，我对 GAST 方法的具体细节有了更为清晰的认识，对其各部分所设计的细节内容也更为了解，其中对实验所需的参数设置，各模块的协调调用及其各自所产生的作用都了解的更为透彻。通过其实验结果可以证明 GAST 方法在少样本数据场景下的高效性能，说明其所提出的生成对抗方法的可行性，为少样本数据情况下的自动算法组构造提供了新的方法及思路。在本次复现过程中，由于对 python 语言的不熟悉，在理解代码的过程中给我带来了不小的挑战，同时，运行代码过程中遇到的一系列问题也暴露了我在编程上的不足。也正是如此，我也更能体会到研究工作的困难之处，也知道了自身需要不断提升的短板，发现问题，克服问题，才能做出好的科研成果。

参考文献

[1] Holger H Hoos. Automated algorithm configuration and parameter tuning. In *Autonomous search*, pages 37–71. Springer, 2012.

[2] Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, and Thomas Stützle. Paramils: an automatic algorithm configuration framework. *Journal of artificial intelligence research*, 36:267–306, 2009.

[3] Carlos Ansótegui, Meinolf Sellmann, and Kevin Tierney. A gender-based genetic algorithm for the automatic configuration of algorithms. In *International Conference on Principles and Practice of Constraint Programming*, pages 142–157. Springer, 2009.

[4] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[5] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization: 5th International Conference, LION 5, Rome, Italy, January 17-21, 2011. Selected Papers 5*, pages 507–523. Springer, 2011.

[6] Shengcai Liu, Ke Tang, and Xin Yao. Generative adversarial construction of parallel portfolios. *IEEE Transactions on Cybernetics*, 52(2):784–795, 2022.

[7] Carla P Gomes and Bart Selman. Algorithm portfolios. *Artificial Intelligence*, 126(1-2):43–62, 2001.

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.