Mining potentially explanatory patterns via partial solutions.

CATALANO, G.A.P.I., BROWNLEE, A.E.I., CAIRNS, D., MCCALL, J. and AINSLIE, R.

2024

Mining Potentially Explanatory Patterns via Partial Solutions

GianCarlo A.P.I. Catalano
Alexander E.I. Brownlee
David Cairns
g.a.catalano@stir.ac.uk
alexander.brownlee@stir.ac.uk
dec@cs.stir.ac.uk
University of Stirling
Computing Science and Mathematics
Stirling, UK

John McCall
j.mccall@rgu.ac.uk
Robert Gordon University
National Subsea Centre
Aberdeen, UK

Russell Ainslie
russell.ainslie@bt.com
BT Technology
Applied Research
Ipswich, UK

ABSTRACT

We introduce Partial Solutions to improve the explainability of genetic algorithms for combinatorial optimization. Partial Solutions represent beneficial traits found by analyzing a population, and are presented to the user for explainability, but also provide an explicit model from which new solutions can be generated. We present an algorithm that assembles a collection of explanatory Partial Solutions chosen to strike a balance between simplicity, high fitness and atomicity, that are shown to be able to solve standard optimization benchmarks.

CCS CONCEPTS

• **Computing methodologies** → **Genetic algorithms**; • **Theory of computation** → *Models of computation.*

KEYWORDS

Genetic Algorithms, Explainable AI (XAI), Combinatorial Optimization Problems

ACM Reference Format:
GianCarlo A.P.I. Catalano, Alexander E.I. Brownlee, David Cairns, John McCall, and Russell Ainslie. 2024. Mining Potentially Explanatory Patterns via Partial Solutions. In *Genetic and Evolutionary Computation Conference (GECCO '24 Companion), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA.

INTRODUCTION

Genetic Algorithms (GAs) have proven their ability to solve many optimization problems, but have made relatively little progress in the aspect of **explainability** and still struggle to gain the user's trust. As GAs are adopted in critical applications [9, 16], the proposed solutions cannot be accepted at face value due to the many issues that can negatively affect the search process [16].

Combinatorial optimization problems are often tackled using GAs, since they are known to be computationally hard (such as Graph Coloring [6], Knapsack [15] and resource allocation [12]). These kinds of problems are the topic of this paper, as they present open questions relating to explainability:

- **What characterizes good solutions?**
- **Why is a solution good, or better than another?**

This work offers one way to address these questions and introduces the concept of a *Partial Solution* (PS), a positive trait commonly found in high-fitness solutions. More specifically, in this work they are patterns where certain variables are fixed and others can vary (indicated using `*`). For example, if the solutions `0001`, `0011`, `0101` and `0111` have high fitness, these would be described using `0**1`. A population of solutions is analyzed to find a set of recurring positive traits, the **Partial Solution catalog**. This acts as a model that is then sampled to generate new solutions. In doing so, the algorithm is inherently explainable, offering the ability for the end user to identify key components of the recommended solutions. Partial Solutions are an explicit decomposition of the problem, which offers the benefits suggested in [4]. Partial Solutions are **interpretable**:

- **Global explanations**: describing the fitness landscape by finding simple and atomic Partial Solutions which are associated with high-fitness.
- **Local explanations**: describing a solution by pointing out the Partial Solutions it contains.

Addionally, Partial Solutions **assist the search process**, because they act as a model that describes high-fitness regions of the solution space (similar to Probabilistic Graphical Models) from which solutions may be constructed. The system we propose consists of:

- **PS Miner**: an algorithm which obtains the PS catalog from a reference population (Section 4),
- **Pick & Merge**: the algorithm which forms full solutions by combining elements from the PS catalog (Section 5).

They are composed in the following (one pass) sequence:

(1) Generate a reference population P_{Ref}, and evaluate it.
(2) Apply the PS Miner on P_{Ref} to generate the PS catalog.
(3) Apply Pick & Merge on the PS catalog to obtain full solutions.

These Partial Solutions offer a novel way of providing both global and local explanations, but at the same time improve the search efficiency when solving the problem. Using benchmark functions which have known underlying structures, in Section 6 we test whether we're able to consistently find these Partial Solutions, and how their ability to solve the optimization problem by finding the optima compares against traditional methods.

GECCO '24 Companion, July 14–18, 2024, Melbourne, VIC, Australia

2 RELATED WORK

Schemata and **backbones** are similarly structured to Partial Solutions, but they are used very differently. **Schemata** are abstractions that were used for theoretical work on GAs, although there are works which do explicitly construct them in order to improve the search efficiency [5, 8], but they are not used for explainability.

Backbones relate to SAT problems where multiple satisfactory solutions share a common sub-configuration. For most problems there is only one backbone, and it is preferred for it to have many fixed variables in order to shrink the search space [13], which is in contrast with the objectives used here.

Since in this work the PS catalog is treated as a model, EDAs are also an adjacent topic. In this sense, there are some related works: **Linkage learning** [1, 10] and **Family-of-Subsets** [14] aim to find which variables are interacting in the fitness function, and are then assembled into structures such as sets or cliques. These structures only determine which variables are interacting, but their optimal value assignments are not specified, whereas PSs contain both. In other words, PSs are instances of the cliques / linked sets found by the aforementioned approaches.

3 PARTIAL SOLUTIONS

3.1 Formal definition

Table 1: Notation

Symbol	Meaning
F	Fitness function to be maximized
F^{ψ}	Fitness function for partial solutions
n	The number of parameters in the solutions
$a, b, \ldots, x$	Variables used to denote a *full* solution
$x_1, x_2, \ldots, x_n$	The parameters of x, a full solution
$a^{\psi}, b^{\psi}, \ldots, x^{\psi}$	Variables used to denote a *Partial* Solution
$x_1^{\psi}, x_2^{\psi}, \ldots, x_n^{\psi}$	The parameters of x^{ψ}, a *Partial Solution*
P_{Ref}	A collection of evaluated full solutions

Table 1 summarizes our notation. A Partial Solution x^{ψ} can be described as "*a sub-configuration of parameter values which is associated with high-fitness solutions within a reference population*", where that reference population will be denoted as P_{Ref}.

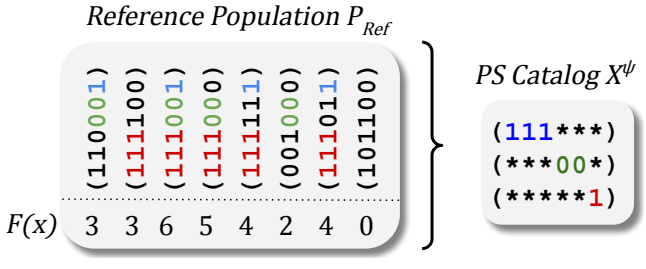

Figure 1: The positive traits in a collection of full solutions P_{Ref} can be described the PS catalog X^{ψ}

In the same way that a full solution is a tuple of values $x = (x_1, x_2, ..., x_2)$, a Partial Solution is $x^{\psi} = (x_1^{\psi}, x_2^{\psi}, ...x_n^{\psi})$ where each x_i^{ψ} can take any value that x_i can, as well as "$*$", the "any" value. A solution x is said to "contain" the Partial Solution x^{ψ} when they agree on the fixed values found in x^{ψ} (Equation (1)). Conversely, it is possible to find all the solutions in P_{Ref} which contain the Partial Solution x^{ψ}, the **observations** (Equation (2)).

$$\text{contains}(x, x^{\psi}) \Leftrightarrow x_i = x_i^{\psi} \; \forall \; i \in [1, n], \; x_i^{\psi} \neq * \tag{1}$$

$$\text{obs}_{P_{\text{Ref}}}(x^{\psi}) = \{x \in P_{\text{Ref}} \mid \text{contains}(x, x^{\psi})\} \tag{2}$$

Within all of possible PSs for an optimization problem, there is a small subset which is preferable for explainability: the **catalog**. These are the PSs which are optimal with respect to three metrics described in Section 3.3: **`simplicity`**, **`mean fitness`** and **`atomicity`**.

3.2 Explainability of Partial Solutions

A collection of PSs, found by analyzing a population, provides a succinct and expressive representation of positive traits (as seen in Figure 1), also described as "**innovization**" in the literature [4]. In a real problem, PSs can offer useful insight into what variables are linked, and what values they should take.

In an employee-allocation problem, for instance, these subconfigurations might indicate that it is beneficial to assign certain groups of employees to specific tasks. The solution `1111111` might have its high fitness explained by the presence of `111***` and `1****1`, although `111001` is even better because of the presence of `***00*`.

Additionally, PSs might show the presence of positive traits that cannot coexist (*e.g.*, `**00` and `*11*`), which might be otherwise unclear. The ability to find such disagreeing PSs can help in explaining situations where generally positive traits are not necessarily found in the global optima.

3.3 Partial Solution metrics

This section will define three metrics, which are used as objectives in the search process to find PSs. **`simplicity`** (Equation (3)) represents the idea that having more parameters with value $*$ (as opposed to fixed values) improves interpretability. **`meanFitness`** (Equation (4)) is the mean fitness of the observations of a PS, measuring to the effect the PSs generally has on fitness. **`atomicity`** (Equation (5)) is the more novel of the metrics, and represents the idea that a PSs should be "irreducible" and not "composite". This is motivated by explainability, since we're aiming to break down the problem into small, atomic parts. This is detected by measuring the contribution of each variable to the rest of the PS (Equation (6)).

$$\text{simplicity}(x^{\psi}) = |\{i \; s.t. \; x_i^{\psi} = *\}| \tag{3}$$

$$\text{meanFitness}(x^{\psi}) = \text{average}(F(\text{obs}_{P_{\text{Ref}}}(x^{\psi})) \tag{4}$$

$$\text{atomicity}(x^{\psi}) = \min \left\{ \text{contribution}(x^{\psi}, k)) \;\middle|\; \begin{matrix} k \in [1, n] \\ x_k^{\psi} \neq * \end{matrix} \right\} \tag{5}$$

$$\text{contribution}(x^\psi, k) = F^\Sigma(x^\psi) \cdot \log\left(\frac{F^\Sigma(x^\psi)}{F^\Sigma(k^\psi) \cdot F^\Sigma(\overline{k}^\psi)}\right) \quad (6)$$

where $k_i^\psi = (x_i^\psi \text{ if } i = k,\ * \text{ otherwise})$

$\overline{k}_i^\psi = (* \text{ if } i = k,\ x_i^\psi \text{ otherwise})$

$m = \min_{x \in P_{\text{Ref}}} F(x)$

$S = \text{sum}\{F(x) - m \mid x \in P_{\text{Ref}}\}$

$F^\Sigma(x^\psi) = \text{sum}\{(F(x) - m) \ / \ S \mid x \in \text{obs}_{P_{Ref}}(x^\psi)\}$

4 FINDING PARTIAL SOLUTION CATALOGS

The algorithm to find the PS catalog (Algorithm 1) is implemented using the metrics in Section 3.1, but the design of the search method involves some non-trivial design choices:

- The **Reference Population** remains fixed throughout the process, making the system usable within a GA
- The three metrics are treated as a **single objective** by normalizing all their values in the range [0, 1] between individuals, and averaging them within individuals.
- An **archive** is used in the search, storing all of the individuals which have been selected and that are not allowed in any future population. The final outputs come from the archive.
- The search could be implemented using a standard GA, or by starting from empty PSs which get progressively "**specialized**", or even from full solutions which get progressively "**simplified**" as PSs. All are tested via different implementations of `get_local` and `get_init` in Algorithm 1.

5 PICK & MERGE ALGORITHM

The PS catalog X^ψ obtained via Algorithm 1 can be used as a model to construct high-quality full solutions. **Pick & Merge** (Algorithm 2) simply picks items from the PS catalog and merges them when possible. At the end, any remaining unfixed parameters are filled with random values, so that a full solution can be returned.

6 EXPERIMENTS

We designed tests to answer the following research questions:

- **RQ1**: Which algorithm parameters are best for finding the PS catalog ?
- **RQ2**: How is the ability to extract PSs from a reference population affected by evolution
- **RQ3**: Can the PS catalog be used to construct good solutions?

These were answered with testing rounds T1, T2 and T3 respectively. T1 and T2 consist of hyperparameter tuning, and T3 checks whether the original optimization problem is still being solved, all using the benchmark problems discussed in the next section.

6.1 Benchmark Problems

Three problems were chosen as benchmarks: Royal Road (RR, defined as [11], Trap-k (defined as [7]) and Royal Road with Overlaps (RRO). Each problem instance was constructed to have 5 "target" PSs: the cliques that we know the algorithm should find (*e.g.* 111**..** for Royal Road). Royal Road with Overlaps is a novel

Algorithm 1: Archive-based PS Miner

```
Def normalize(values):
    return (values − min(values)) / (max(values) − min(values))
Def F^ψ(X^ψ):
    M ← normalize(meanFitness(X^ψ))
    S ← normalize(simplicity(X^ψ))
    A ← normalize(atomicity(X^ψ))
    return {avg(m, s, a) for m, s, a ∈ zip(M, S, A)}
Def top(X^ψ, quantity):
    sorted ← sort(X^ψ, key = F^ψ, order = descending)
    return sorted[:quantity]
get_init and get_local are discussed in Section 4
Def mine_ps(get_init, get_local, pop_size, qty_ret):
    X^ψ ← get_init()
    archive ← {}
    while termination_criteria_not_met do
        selected ← TournamentSelection(X^ψ)
        localities ← ⋃_{x^ψ ∈ seleted} get_local(x^ψ)
        archive ← archive ∪ selected
        X^ψ ← (X^ψ ∪ localities) ∖ archive
        X^ψ ← top(X^ψ, pop_size)
    return top(archive, qty_ret)
```

Algorithm 2: Pick & Merge algorithm

```
Def merge(x^ψ, y^ψ):
    return z^ψ s.t. z_i^ψ = x_i^ψ if x_i^ψ ≠ * else y_i^ψ
Def merge_from(X^ψ, limit = ⌈√n⌉):
    Y^ψ ← copy(X^ψ)
    x^ψ ← **..**
    added ← 0
    while Y^ψ ≠ ∅ & added < limit & has_*(x^ψ) do
        y^ψ ←weighted_random_choice(Y^ψ)
        Y^ψ ← Y^ψ ∖ {y^ψ}
        if mergeable(x^ψ, y^ψ) then
            x^ψ ← merge(x^ψ, y^ψ)
            added ← added + 1
    return x^ψ
Def fill_gaps(x^ψ):
    for x_i^ψ ∈ x^ψ do
        if x^ψ = * then
            x_i^ψ ← random.randrange(cardinalities[i])
    return as_solution(x^ψ)
Def generate_via_pick_and_merge(X^ψ, merge_limit):
    return fill_gaps(merge_from(X^ψ, merge_limit))
```

problem introduced here to test whether the algorithms struggle with overlapping building blocks. The fitness function of the problem is similar to Royal Road, but the groups of interest are allowed to overlap and to consists of all 0's as well. The cliques of RR and RRO consist of 4 bits, whereas Trap-k will have cliques of 5 bits.

6.2 Setup

Each algorithm configuration is executed 100 times, with a freshly generated P_{Ref}, and restricted to return at most 50 PSs.

T1 consists of Hyper-parameter tuning the PS Miner in PS population size, use of archive and other aspects, with fixed $|P_{\text{Ref}}| = 10^4$ (uniformly randomly generated) and F^{ψ} evaluation budget of 10^5. A run is considered successful if all the target PSs of the problem are returned.

T2 investigates the effect of P_{Ref} on the miner, based on its size and the generations it has been evolved for.

Finally, in **T3** the entire system is used: a run consists of finding the PS Catalog using the PS miner (configured using T1 and T2) and passing it to Pick & Merge to generate 100 full solutions. The total evaluation budget varies, with differing percentages being dedicated to F^{ψ} evaluations. A run is considered successful when the global optima is found.

7 CONCLUSION

The results of the experiments are the following:

- **RQ1**: The best approach to finding PSs was **specialization**: to start from empty items and gradually fix the variables.

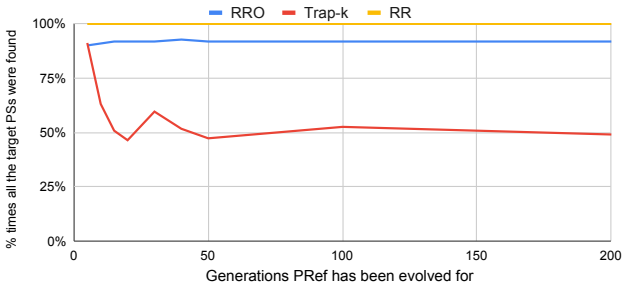

Figure 2: Results from T2, plotting the success rate (finding PS Catalogs) with a fixed P_{Ref} size of 10^4

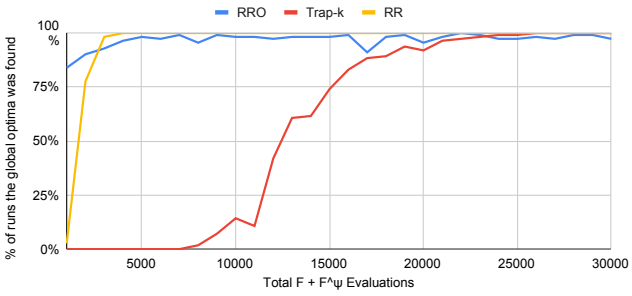

Figure 3: Plot for T3, percentage of runs where the global optima was found, with F^{ψ} consuming 50% of the budget.

- **RQ2**: The proposed PS mining algorithm performs better on a large unevolved population
- **RQ3** Applying Pick & Merge on the PS Catalog allows for the global optima to be found consistently, especially when F budget is close to the F^{ψ} budget.

The system proposed in this paper is capable of finding the intended PSs for the benchmark problems, and to solve the original optimization problem, at the cost of extra F^{ψ} evaluations. Source code for the system can be found at [2], and an extended version of this document can be found at [3].

8 FUTURE WORK

Overall, the system could be readapted to be a full EDA, for example by feeding the outputs of `pick_and_merge` into P_{Ref}.

The problems discussed in this document are admittedly simpler than **real-world problems**, and therefore it would be useful to see how well the PSs perform with some **inherently inelegant problems**, such as rostering and scheduling tasks.

Acknowledgment: PhD project supported by The Data Lab and BT Group plc. Many thanks go to the patience of my supervisors.

REFERENCES

[1] Shumeet Baluja and Scott Davies. 1997. Using Optimal Dependency-Trees for Combinational Optimization. In *Proceedings of the Fourteenth ICML*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 30–38.
[2] Giancarlo Catalano. 2024. *PS Assisted Explainability.* https://github.com/Giancarlo-Catalano/PS_Minimal_Showcase
[3] GianCarlo A.P.I Catalano, Alexander E. I. Brownlee, David Cairns, John McCall, and Russell Ainslie. 2024. Mining Potentially Explanatory Patterns via Partial Solutions. arXiv:2404.04388
[4] Kalyanmoy Deb and Aravind Srinivasan. 2008. Innovization: Discovery of innovative design principles through multiobjective evolutionary optimization. *Multiobjective Problem Solving from Nature* (2008), 243–262.
[5] Jack McKay Fletcher and Thomas Wennekers. 2017. A natural approach to studying schema processing. arXiv:1705.04536 [cs.NE]
[6] Michael R. Garey and David S. Johnson. 2006. Computers and Intractability: A Guide to the Theory of NP-Completeness. 24 (7 2006), 90–91. Issue 1. https://doi.org/10.1137/1024022
[7] David E Goldberg. 1989. Genetic algorithms and Walsh functions: Part 2, Deception and its analysis. *Complex systems* 3 (1989), 153–171.
[8] David E. Goldberg, Kumara Sastry, and Yukio Ohsawa. 2003. *Discovering Deep Building Blocks for Competent Genetic Algorithms Using Chance Discovery via KeyGraphs.* Springer, Berlin, Heidelberg, Tokyo, Japan, 276–301. https://doi.org/10.1007/978-3-662-06230-2_19
[9] Katja Grace, John Salvatier, Allan Dafoe, Baobao Zhang, and Owain Evans. 2018. Viewpoint: When Will AI Exceed Human Performance? *Journal of Artificial Intelligence Research* 62 (7 2018), 729–754. https://doi.org/10.1613/JAIR.1.11222
[10] Shih-Huan Hsu and Tian-Li Yu. 2015. Optimization by pairwise linkage detection, incremental linkage set, and restricted/back mixing: DSMGA-II. In *Proceedings of GECCO 2015.* ACM, New York, NY, USA, 519–526.
[11] Melanie Mitchell, Stephanie Forrest, and John Holland. 1992. The Royal Road for Genetic Algorithms: Fitness Landscapes and GA Performance. *European Conference on Artificial Life* 1 (11 1992), 23–33.
[12] L. Rabiner. 1984. Combinatorial optimization:Algorithms and complexity. *IEEE Transactions on Acoustics, Speech, and Signal Processing* 32 (12 1984), 1258–1259. Issue 6. https://doi.org/10.1109/TASSP.1984.1164450
[13] John Slaney and Toby Walsh. 2001. Backbones in Optimization and Approximation. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 1* (Seattle, WA, USA) *(IJCAI'01).* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 254–259. https://doi.org/10.5555/1642090.1642125
[14] Dirk Thierens and Peter Bosman. 2011. Optimal mixing evolutionary algorithms. In *Proceedings of the 13th GECCO conference.* ACM, New York, NY, USA, 617–624.
[15] Pamela H. Vance. 2006. Knapsack Problems: Algorithms and Computer Implementations (S. Martello and P. Toth). 35 (7 2006), 684–685. Issue 4. https://doi.org/10.1137/1035174
[16] Feiyu Xu, Hans Uszkoreit, Yangzhou Du, Wei Fan, Dongyan Zhao, and Jun Zhu. 2019. Explainable AI: A Brief Survey on History, Research Areas, Approaches and Challenges. 11839 LNAI (2019), 563–574. https://doi.org/10.1007/978-3-030-32236-6_51