gLINC: Identifying Composability using Group Perturbation

David J. Coffin
Department of Computer Science
University College London
Gower Street, London, WC1E 6BT
d.coffin@cs.ucl.ac.uk

Christopher D. Clack
Financial Computing and Quantitative Finance
Department of Computer Science
University College London
Gower Street, London, WC1E 6BT
c.clack@cs.ucl.ac.uk

ABSTRACT

We present two novel perturbation-based linkage learning algorithms that extend LINC [5]; a version of LINC optimised for decomposition tasks (oLINC) and a hierarchical version of oLINC (gLINC). We show how gLINC decomposes a fitness landscape significantly faster than both LINC and oLINC.

We present details of LINC, oLINC and gLINC, an empirical comparison of their speed, accuracy and sensitivity to population size on a concatenated trap function, and a discussion of their complexity and correctness.

Categories and Subject Descriptors

F.2 [**Analysis of algorithms and problem complexity**]: General

General Terms

Algorithms, Experimentation

Keywords

Genetic Algorithms, Linkage Learning, Epistasis, Composition, Perturbation, Hierarchical

1. INTRODUCTION

Several forms of phenotypic linkage exist. They measure interesting properties of the interactions of genes' contributions to fitness (e.g. linearity, monotonicity). This contrasts with what could be called inheritability linkage, which is interdependence of the genes probability of being inherited. Phenotypic linkage is a property of the fitness landscape, whereas inheritability linkage is a property of the representation and genetic algorithm (GA).

Building blocks are groups of alleles that positively effect fitness and whose genes are strongly phenotypically linked. The schema theorem has shown that if building blocks do not also have inheritability linkage they are likely to be disrupted and the problem will be difficult for the given GA and representation [2]. However, if phenotypic linkage is known, the GA or the representation may be altered to minimise this disruption.

Particular cases of phenotypic linkage permit a decomposition of the fitness landscape into sub-problems. If two genes are not linked then they are separable. If, for two disjoint groups of genes, every gene in one group is separable from every gene in the other, then those groups are separable. If those two groups partition the search space then the search space is also separable[1]. For search algorithms that run on linearly separable problems with greater than linear complexity, decomposing the problem into sub-problems and combining the sub-solutions will speed up search.

Linkage learning has implications for practical applications of GAs. Many real-world problems contain substantial amounts of linkage — for example, in Financial Portfolio Optimisation the inbound data streams are often either strongly correlated or strongly contra-correlated (both examples of linkage). This example is particularly problematic because the degrees of correlation are often not known in advance and may vary in time. However such problems may also contain significant separable groups. For example, a global portfolio may contain stocks from different political regimes whose movements are entirely uncorrelated.

This paper extends and specialises an existing linkage learning algorithm (linkage indentification by nonlinearity check — LINC) that has previously been applied to decomposing search spaces based on separability[2] within a fitness landscape [4]. As with the order-2 perturbation mechanism used to detect non-linearity in LINC, the group perturbation mechanism presented here could be used with alternative measures of linkage without loss of generality.

Our new linkage learner gLINC ("greedy LINC") is significantly faster than LINC at this decomposition task.

1.1 Related Work

Implicit linkage learning techniques, where the representation or properties of the genetic operators have adapted

GECCO'06, July 8–12, 2006, Seattle, Washington, USA.

[1]In other words, treating the loci as nodes and linkage as edges, two partitions are separable if they are disconnected graphs. This definition of 'separability' is stricter than normal — often two partitions are considered separable even if a low number of loci are linked between them.

[2]In this paper we use 'decomposition' to refer to a partitioning of the search space.

together with individuals or alongside the population, were considered at least as far back as Holland's inversion operator [8]. The messy GA [3] uses a 'messy' phase to search for useful representations *before* the search proper commences. The linkage learning GA (LLGA) [7] uses value/loci tuples rather than fixed loci representation *throughout* the evolutionary run. These algorithms are thought of as implicit linkage learners in that there is selective pressure for less disruptive representations or operators, so that — if the algorithm is successful — they become implicit representations of the phenotypic linkage.

Explicit linkage learning techniques manipulate a model of phenotypic linkage. Like implicit linkage learning, this may also occur before or during search. The perturbation-based linkage-learning algorithms (LINC being the canonical example[5]) build this model based on systematic sampling of the fitness landscape. Many varieties of perturbation-based linkage learning algorithms have been proposed, including use of different linkage measures [6], application to real-valued problems [10] and incorporation into Estimation of Distribution algorithms [12]. The hierarchical Linkage Identification by Epistasis Measure algorithm uses the same group perturbation mechanism as gLINC but does not use it to perform any kind of optimisation [11]. LINC itself is discussed in detail in the next section.

The Symbiogenic Evolutionary Adaptation Model (SEAM) [9] and the Hierarchical GA (HGA) [1] are forms of perturbation-based linkage-learning algorithms that share similarities with gLINC. SEAM combines under-specified individuals, without replacement, if their combination is fitter than either individual in a selection of contexts generated randomly from the rest of the population. This 'stability' check is similar to a group perturbation except the fitness of the context alone is not measured. The HGA uses a similar mechanism, combining loci (rather than loci values) into modules where that will reduce the number of context-dependent optima.

1.2 Linkage Indentification By Nonlinearity Check

The 'LINC' linkage-learning algorithm[5] uses an order-2 perturbation operator.

An order-2 perturbation operator checks whether a particular relation — linear separability in LINC's case — holds between the variables at two loci by comparing the fitness of a series of individuals with the values of those variables changed. By inspecting how independent and combined perturbation alters the fitness, the perturbation operator can determine whether or not that relation holds for a particular region of the search space. Since enumerating the entire search space would beg the point of learning its linkage, perturbation-based linkage learning algorithms typically act on a null hypothesis basis over a smaller sample. If the hypothesised relation holds over this sample, the algorithm assumes it holds over the whole search space, or at least any regions of the search space that might affect search.

For example, given the individual **001** (with a binary representation) whose fitness is given by $f(001)$, we may choose to perturb loci 0 and 1. The fitnesses of **101** and **011** are therefore evaluated. Finally, both loci are perturbed and the fitness of **111** is evaluated. By reference to Fig. 1, loci i and j in individual s (where $i \neq j$) are not linearly separable if

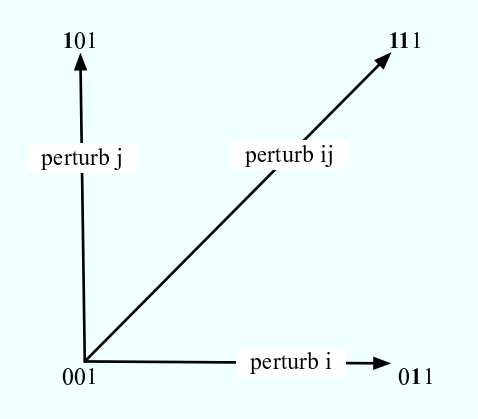

Figure 1: Examining fitness perturbations (i=0, j=1).

$$\Delta f_{ij}(s) \neq \Delta f_i(s) + \Delta f_j(s) \tag{1}$$

where

$$\Delta f_i(s) = f(\ldots \bar{s_i} \ldots) - f(\ldots s_i \ldots)$$

$$\Delta f_j(s) = f(\ldots \bar{s_j} \ldots) - f(\ldots s_j \ldots)$$

$$\Delta f_{ij}(s) = f(\ldots \bar{s_i} \ldots \bar{s_j} \ldots) - f(\ldots s_i \ldots s_j \ldots)$$

For noisy fitness landscapes we can replace the equality with an error threshold e.

$$|\Delta f_{ij}(s) - (\Delta f_i(s) + \Delta f_j(s))| > e \tag{2}$$

The LINC algorithm and it's variants use three nested loops:

- The outer loop iterates over each individual in a population.
- The middle loop traverses the current individual's chromosome from start to finish, looking at each allele.
- The inner loop iterates over the alleles in the remainder of the chromosome. This permits the algorithm to investigate every pair of loci, in the context of every individual in the population. If linkage is discovered between a pair of loci, each loci is added to the other's linkage set.

The pseudocode for LINC is given in Fig. 2.

If this linkage information is exploited by decomposing the search space, the linkage information must be used to generate partitionable sets. If linkage groups are non-overlapping, then this simply involves removing duplicates from the linkage set [4]. However if linkage groups overlap, then they must be combined — otherwise at least two variables in different sub-problems would be linked, so the sub-problems would not be linearly separable and optimising the sub-problems would not guarantee an optimal solution overall.

For example:

```
linkage_set = [[0] .. [length-1]]
for s in pop:
  fs = f(s)
  for i in [0 .. length-1]:
    s1 = perturb(s,i)
    df1 = f(s1) - fs
    for j in [i+1 .. length-1]:
      s2 = perturb(s,j)
      df2 = f(s2) - fs
      s12 = perturb(s2,i)
      df12 = f(s12) - fs
      if abs(df12 - (df1 + df2)) > e:
        linkage_set[i].add(j)
        linkage_set[j].add(i)
```

Figure 2: LINC pseudocode

1. Given an 4-variable fitness function where the loci $\{0, 1\}$ are epistatically linked and $\{2, 3\}$ are epistatically linked, the LINC algorithm initially assumes that each locus is independent and therefore initialises the linkage vector as [[0], [1], [2], [3]]. Then for an individual *abcd* each pair of alleles is inspected; (a,b), (a,c), (a,d), (b,c), (b,d), and (c,d). If linkage is detected via non-linear fitness variations in perturbations, the linkage information for *both* loci is updated. Assuming both non-linear relations are detected perturbing this individual[3], the linkage vector would become $[[0, 1], [1, 0], [2, 3], [3, 2]]$.

2. If the same example had linkage groups $\{0, 1\}$ and $\{1, 2\}$, the linkage vector would become $[[0, 1], [1, 0, 2], [2, 1], [3]]$.

1.3 Optimised Linkage Indentification By Non-linearity Check

Noting that LINC repeats perturbation-based linkage checks between loci that are already known to be part of the same composition group (because they are known to be linked or indirectly connected), and that one instance of linkage is enough to consider two loci as linked, an obvious optimisation presents itself. Our optimised LINC (oLINC) algorithm incorporates a 'no repetition' rule — these unnecessary repetitions are avoided.

- The outer loop iterates over each individual in a population.
- The middle loop traverses the current individual's chromosome from start to finish, looking at each allele.
- The inner loop iterates over the alleles in the remainder of the chromosome. If those alleles' loci are both in the same composition group, the algorithm moves on to the next pair of alleles, otherwise, LINC linkage learning proceeds as normal. If linkage is discovered between a pair of loci, their composition sets are combined.

The pseudocode for oLINC is given in Fig. 3.

[3]Non-linearity may hold between two variables without being present in all parts of the search space. Deceptive landscapes often contain non-linearity only in the region of the search space surrounding the solution.

```
composability = [[0] .. [length-1]]
for s in pop:
  fs = None
  for ci in [0 .. length-1]:
    fi = None
    for cj in [ci+1, length-1]:
      for i in composability[ci]:
        for j in composability[cj]:
          if composability[ci] and composability[cj]:
            if fs == None:
              fs = f(s)
            if fi == None:
              si = perturb(s, i)
              fi = f(si)
            sj = perturb(s, j)
            fj = f(sj)
            sij = perturb(si, j)
            fij = f(sij)
            if abs((fs + fij) - (fi + fj)) > e:
              composability[cj] += composability[ci]
              composability[ci] = []
```

Figure 3: oLINC pseudocode

1.4 Greedy Linkage Indentification By Non-linearity Check

In the first example of the LINC algorithm (see section 1.2), there was no overlap in the linkage groups. However, in the second example there was overlap between the two linkage groups $\{0, 1\}$ and $\{1, 2\}$. Note that linkage is not transitive — although 0 and 1 are linked, and 1 and 2 are linked, 0 and 2 are not linked. However, if this information were to be exploited by decomposing the fitness function into separable sub-problems, the separable sub-problems would be determined by the loci sets $\{0, 1, 2\}$ and $\{3\}$.

An important general observation is that whilst linkage is intransitive, *composability* is transitive. Exploiting linkage information through decomposition involves partitioning a search space. For this purpose, the LINC algorithm is inefficient in that it identifies too much detail (linkage groups) and must then post-process that information to determine the final decomposition.

We aim to improve substantially the performance of decomposition via perturbation-based linkage learning by directly detecting composability (rather than detecting linkage and then post-processing to derive composition groups). We do this via a new algorithm which identifies transitive relationships and agglomerates separable groups on-the-fly — we have implemented this algorithm as an extension to LINC (though it is not restricted to this context) and we call this the Greedy LINC (gLINC) algorithm. 'Greedy' is intended to suggest an accumulative process similar to greedy search in regular expressions. The combination of direct detection and inspection of group-to-group composability saves a significant number of evaluations in highly linked fitness landscapes.

gLINC incorporates the 'no repetition' optimisation of oLINC, and adds the following key difference:

- gLINC introduces a group perturbation linkage learning operator. By perturbing groups of loci simultaneously, multiple non-linearities can be tested concur-

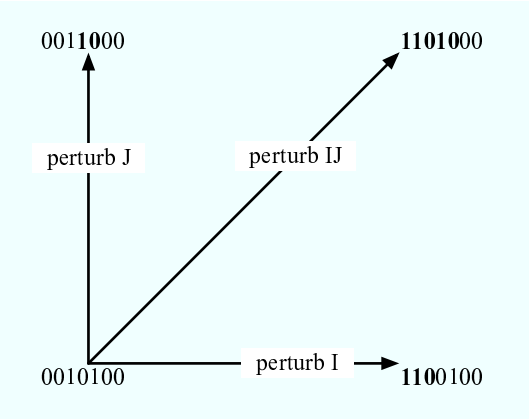

Figure 4: Examining fitness perturbations (I={0,1,2}, J={3,4}).

rently. gLINC exploits this wherever possible, leading to combinatorial savings in fitness evaluations.

For example, given the individual **0010100**, we may already know that the loci $\{0, 1, 2\}$ are composed *and* that loci $\{3, 4\}$ are composed. We wish to know whether the two groups either belong to the composition group — i.e. for linkage between any of the loci in $\{0, 1, 2\}$ with any of the loci in $\{3, 4\}$. The gLINC algorithm perturbs bits at loci 0,1,and 2 *concurrently*, and then perturbs bits at loci 3 and 4 concurrently, evaluating the fitnesses of **1100100** and **0011000**. Next, bits at loci 0,1,2,3,and 4 are perturbed and the fitness of **1101000** is evaluated. By reference to Fig. 4, if

$$|\Delta f_{\{0,1,2,3,4\}}(s) - (\Delta f_{\{3,4\}}(s) + \Delta f_{\{0,1,2\}}(s))| > e$$

then there must be at least one non-linear relation between loci in both groups, so they must comprise a single composition group $\{0, 1, 2, 3, 4\}$).

Formally, loci groups $I = \{I_1 \dots I_n\}$ and $J = \{J_1 \dots J_m\}$ (where I and J are disjoint sets), form a single composition group if

$$\Delta f_{IJ}(s) \neq (\Delta f_I(s) + \Delta f_J(s)) \tag{3}$$

where

$$\Delta f_I(s) = f(s_{\bar{I}_1} \dots s_{\bar{I}_n} \dots) - f(\dots s_{I_1} \dots s_{I_n} \dots)$$

$$\Delta f_J(s) = f(\dots s_{\bar{J}_1} \dots s_{\bar{J}_m} \dots) - f(\dots s_{J_1} \dots s_{J_m} \dots)$$

$$\begin{aligned}\Delta f_{IJ}(s) = f(\dots s_{\bar{I}_1} \dots s_{\bar{I}_n} \dots s_{\bar{J}_1} \dots s_{\bar{J}_m} \dots) \\ - f(\dots s_{I_1} \dots s_{I_n} \dots s_{J_1} \dots s_{J_m} \dots)\end{aligned}$$

Again, for noisy fitness landscapes we can replace the equality with an error threshold e.

$$|\Delta f_{IJ}(s) - (\Delta f_I(s) + \Delta f_J(s))| > e \tag{4}$$

The cases in which I and J may form a single composition group without the threshold being exceeded are discussed in Section 3.2.

The gLINC algorithm modifies LINC's three loops as follows:

- The outer loop iterates over each individual in a population.
- The middle loop traverses over a list of composed loci (the same length as the chromosome) from start to finish, looking at each set of composed loci in turn.
- The inner loop traverses the remainder of the list of composed loci.

  If both sets of composed loci are non-empty, the algorithm performs a group-perturbation based non-linearity check between these two groups of composed loci. If the groups are linked, that composition is stored by appending all the loci in the first group to the second, and the first group is emptied.

The pseudocode for gLINC is given in Fig. 5. For example:

1. Given an individual chromosome with loci $\{0, 1, 2, 3\}$ where 0 and 1 are epistatically linked and 2 and 3 are epistatically linked, the gLINC algorithm initially assumes that each loci is independent and therefore initialises the composability vector as $[[0], [1], [2], [3]]$. If composability is detected via non-linear fitness variations in perturbations, the composability information for *the second* allele is updated and the composability information for the first allele is emptied. Hence, the final composability vector in this example would be $[[], [0, 1], [], [2, 3]]$.
2. If the same example had linkage groups $\{0, 1\}$ and $\{1, 2\}$, the final composability vector would be $[[], [], [0, 1, 2], [3]]$.

2. EXPERIMENTS

Three key research questions arise from the oLINC and gLINC algorithms discussed above:

1. is gLINC actually any faster than oLINC or LINC?
2. does gLINC require a different population size to oLINC or LINC? and
3. how do the three algorithms compare in terms of accuracy?

These three questions are investigated through two experiments described in this section. The first primarily addresses the issue of performance and also provides some information about accuracy; the second directly addresses the issues of accuracy and population size.

2.1 Experiment 1

The number of fitness evaluations that LINC, oLINC and gLINC required to decompose a deceptive landscape — a shuffled series of 5-bit trap functions — was compared in a scale-up experiment.

For each i^{th} trap, the trap function is given below, where u_i is the number of 1's.

```
composability = [[0] .. [length-1]]
for s in pop:
  fs = None
  for i in [0 .. length-1]:
    gi = composability[i]
    if gi != []:
      df1 = None
      for j in [i+1 .. length-1]:
        gi = composability[i]
        gj = composability[j]
        if gi != [] and gj != []:
          if fs == None:
            fs = f(s)
          if df1 == None:
            s1 = gperturb(s,gi)
            df1 = f(s1) - fs
          s1 = gperturb(s,gi)
          df1 = f(s1) - fs
          s2 = gperturb(s,gj)
          df2 = f(s2) - fs
          s12 = gperturb(s2,gi)
          df12 = f(s12) - fs
          if abs(df12 - (df1 + df2)) > e:
            composability[j] += composability[i]
            composability[i] = []
```

Figure 5: gLINC pseudocode

$$f_i(u_i) = \begin{cases} 4 - (u_i) & \text{if } 0 \geq u_i \geq 4 \\ 5 & \text{otherwise} \end{cases} \quad (5)$$

The traps were then shuffled by choosing a random one-to-one mapping between loci in the representation, and loci in the series of trap functions. Shuffling was used because the contiguous nature of linkage in non-shuffled traps might have unfairly benefited gLINC.

Because the trap functions are non-overlapping, extracting composition information from LINC's linkage sets just involved removing duplicates.

The number of traps was scaled from 1 to 60 (bitstring lengths from 5 to 300). The population size was constant at 30, seeded with uniformly random individuals. At each problem size, LINC was applied once, and oLINC and gLINC were each applied 10 times; the number of evaluations LINC makes is fully determined by the population and bitstring size, so only one run at each problem size was necessary, but for oLINC and gLINC, the number of evaluations made are affected by the population and the order in which linkage is discovered, so a larger sample was necessary. The results are given in Fig. 6 and discussed in Section 3.

The number of decomposition sets found by LINC and gLINC was also recorded. Although LINC always identified all composition sets correctly, oLINC and gLINC sometimes failed to correctly identify some groups as linked, as shown in Fig. 7; this is further investigated in Experiment 2 and an analysis is given in Section 3.

2.2 Experiment 2

In the second experiment the number of traps was kept constant — at 5 traps (string size 25) — and in a single run the population size was incrementally increased (one individual at a time) from 1 to 100. After each new individual was added, the linkage learning algorithm was run and recordings made of the cumulative number of evaluations and the number of composition groups found so far. Ten runs were made for each of the three algorithms LINC, oLINC and gLINC.

The aim of this experiment was to determine how these algorithms respond to changes in population size in terms of both performance (evaluations) and accuracy (extra groups found). The results are given in Figs. 8, 9 and 10, and are discussed in Section 3.

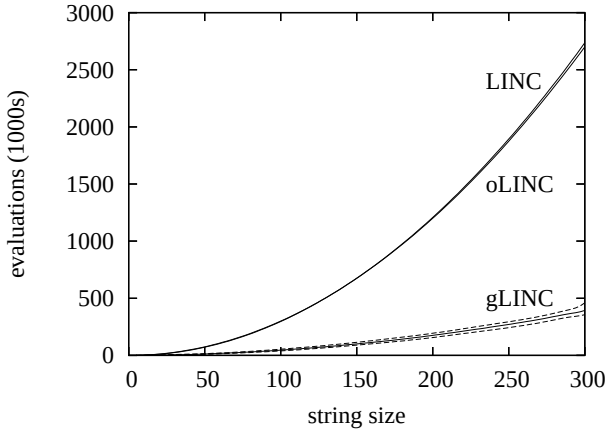

Figure 6: Fitness evaluations against string size (population size 30).

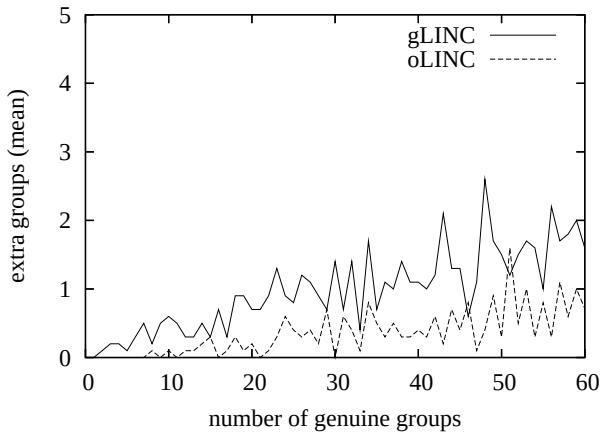

Figure 7: Number of additional composition groups in oLINC and gLINC results against number of genuine groups (population size 30).

3. DISCUSSION OF RESULTS

Fig. 6 plots the number of fitness evaluations used by the three algorithms LINC, oLINC and gLINC (in the case of gLINC these are means, with the minimum and maximum indicated by the error bars) against the size of the strings. gLINC clearly outperforms LINC and oLINC at this task.

The failure of oLINC to provide any appreciable increase in performance shows that the gLINC improvement can be attributed to the group perturbation operator rather than

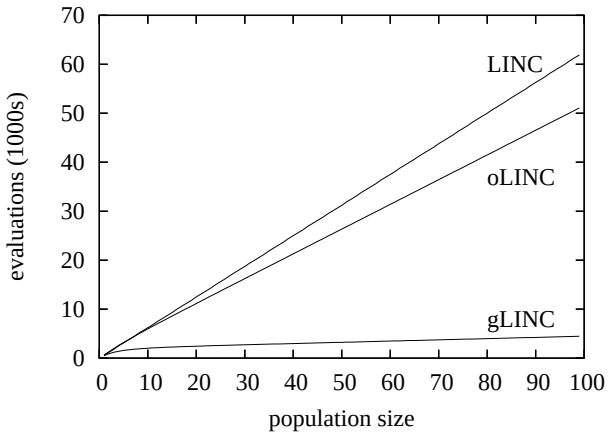

Figure 8: Mean evaluations for complete run against population size for shuffled 5 5-bit trap function.

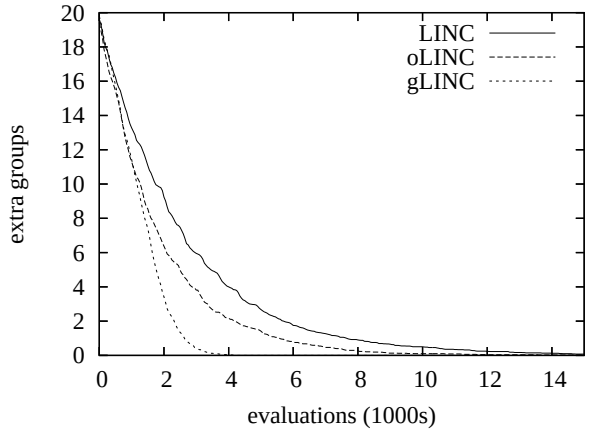

Figure 9: Mean additional groups against evaluations on shuffled 5 5-bit trap function.

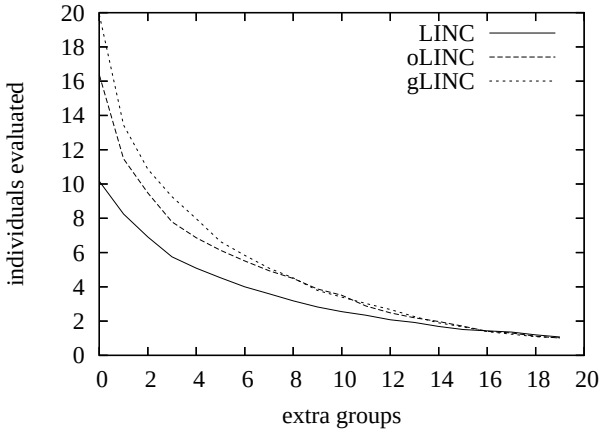

Figure 10: Population size required to achieve a given accuracy.

the no-repetition enhancement of oLINC. Section 3.1 contains an analysis of the time complexity of LINC and gLINC.

Although LINC uses far more evaluations, Fig. 7 shows that gLINC does not accurately join all non-separable groups when compared to oLINC with the same population size. This initially appears problematic. However, the results of Experiment 2 show that gLINC can be applied to a larger population to obtain an accuracy equivalent to LINC, and still use less evaluations.

Consider Fig. 9, which indicates that gLINC will have on average zero extra groups (i.e. is fully accurate) when it makes at least 4,000 evaluations (for our test problem). By reference to Fig. 8, gLINC will perform 4,000 evaluations when the population size is at least 100. Compare this with LINC: Fig. 9 indicates that in order to obtain full accuracy LINC must perform at least 12,000 evaluations, and Fig. 8 shows that this number of evaluations will be performed when the population size is at least 20. Thus, for this problem, for full accuracy on a string of length 25, gLINC requires a population size that is about 5 times bigger yet requires 3 times fewer evaluations (the performance difference increases as the string size increases).

The population size that is required by gLINC, oLINC and LINC in order to achieve a given accuracy (in terms of mean number of extra groups reported) is illustrated in Fig. 10, and the cases in which LINC and gLINC may fail to detect linkage are analysed further in Section 3.2.

3.1 Complexity

Fitting the observed number of evaluations of oLINC and gLINC against a quadratic function gives (where l is string size):

$$fit(oLINC) \approx 30l^2 \tag{6}$$

$$fit(gLINC) \approx 4.3l^2 \tag{7}$$

The number of evaluations LINC takes can be obtained exactly by inspection of the code:

$$fit(LINC) = 30 + 30l^2 \approx 30l^2 \tag{8}$$

oLINCs enhancements are in themselves insignifant. However gLINC is approximately seven times faster than LINC.

The computational complexity for gLINC depends on the order in which it discovers linkage, but is bounded by the best-case in which it discovers linkage between each group for the first individual it permutes, and the worst-case in which there is no linkage.

This theoretical best-case complexity for gLINC is linear, and the worst-case just under that of LINC. However note that although the theoretical best-case is linear, the number of evaluations will always scale quadratically as the number of composition groups are increased, because of the cominatorial nature of the checks for linkage between groups that have to be made.[4]

$$o_{best}(gLINC) = 3(l-1) + 1 = O_{best}(l) \tag{9}$$

$$o_{worst}(gLINC) = nl^2 = O_{worst}(nl^2) \tag{10}$$

[4]These also apply to any other order-2 perturbation-based linkage learner.

3.2 Accuracy

If groups are incorrectly identified as linearly separable then linear combination of the fittest solutions to those sub-groups may not be optimal, even if all the sub-solutions are optimal.

Both gLINC and LINC will fail to correctly compose linked groups without enough individuals, if linkage is confined to a particular region of the fitness landscape and the population does not contain any individuals that allow them to investigate it. The group perturbation mechanism introduces a second possibility of error; when there are several instances of linkage between to composition groups that cancel each other out.

Both these cases are discussed below.

Higher-Order Linkage

gLINC and LINC/oLINC may all fail to identify linkage with order > 2 if the population is not large enough.

If linkage has order > 2, it can be detected between *two* loci, but only if the individual being perturbed has the correct bits. For example in the case of one 5-bit trap function, if no individual in the population matches the schema ##111, then linkage between the loci 0 and 1 (as part of the linkage set $\{0, 1, 2, 3, 4\}$), cannot be identified.

However the population was large enough in our experiment that LINC could detect every instance of linkage correctly.

Equal and Opposite Non-Linearities

gLINC may also fail to identify linkage if there are two or more instances of linkage between loci in the perturbation groups such that the non-linearities cancel each other out.

A minimal example is;

schema	Δ fitness
11#	+10
1#1	+1
#00	-1

Assume the loci $\{0, 1\}$ have been identified as one composition group (e.g. due to the linkage shown in the first row), so that $I = \{0, 1\}$ and $J = \{2\}$ and the current individual is **000**. The group perturbations will result in the evaluation of individuals **110**, **001** and **111** (see Fig. 11). Since the loci 0 and 1 are changed concurrently, the linkage between loci 0 and 2 as well as between loci 1 and 2 (shown in the second and third rows) cannot be tested for independently.

Noting that the group non-linearity detection rule given in Equation 3 can be rewritten as:

$$\begin{aligned} \Delta f_{IJ}(s) &= \Delta f_I(s) + \Delta f_J(s) \\ f_{IJ}(s) - f(s) &= f_I(s) + f_J(s) - 2f(s) \\ f_{IJ}(s) + f(s) &= f_I(s) + f_J(s) \end{aligned} \tag{11}$$

if two or more non-linearities alter $f_{IJ}(s)$ and $f(s)$ such that the sum of their changes equals 0, they will not be detected by group perturbation. The same reasoning applies to $f_I(s)$ and $f_I(s)$.

In this fitness landscape, the sum of the non-linearities between loci 0 and 2 and between loci 1 and 2 equals 0, and

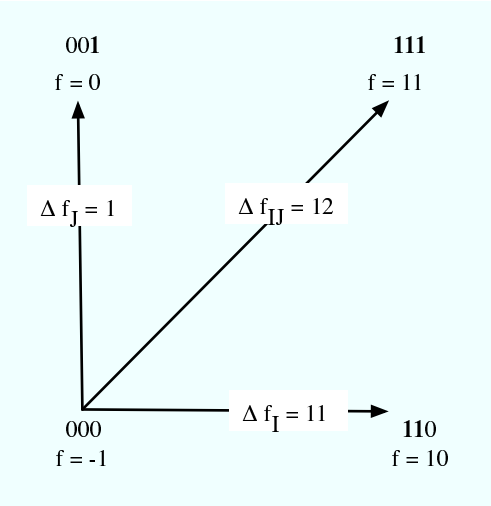

Figure 11: Hidden non-linearity for s=000 (I={0,1}, J={2}).

for this individual they apply to f_{IJ} and f respectively, so neither non-linearity will be detected:

$$\begin{aligned} f_{IJ}(000) + f(000) &= f(111) + f(000) \\ &= 11 + -1 = 10 \\ f_I(000) + f_J(000) &= f(110) + f_J(001) \\ &= 0 + 10 = 10 \end{aligned}$$

$$\therefore f_{IJ}(000) + f(000) = f_I(000) + f_J(000)$$

In the example given above gLINC would have discovered the linkage given any other individual. However a fully deceptive case can be created that will be insoluble by gLINC for all individuals, by adding non-linearities to an otherwise linearly separable fitness landscape a pair at a time, where each pair of non-linearities are two alterations to fitness whose sum is zero and that apply to inversely-related schemata[5].

To see why this is the case consider the process by which gLINC creates it's composed groups of loci. For any genuine composition group, at some point gLINC will have found two subsets of that group that together would form the complete group. Assume (without loss of generality) that in fact every loci in the string is linked, so that the genuine group in this example is every loci. In this case since I contains every loci not in J, and IJ contains all loci, s_I must be the inverse of s_J, and s_{IJ} must be the inverse of s. So by adding non-linearities in pairs to inversely-related schemata, we can guarantee that at this point in gLINCs run, for any non-linearity that effects s, there will have an equal and opposite non-linearity that effects it's inverse, s_{IJ}, and vice versa. Equally for $_I$ and $_J$, any non-linearity that alters the fitness of s_I will have a pair that alters the fitness of it's inverse, s_J, such that together both non-linearites cancel

[5]Where a inverse of a schemata is the schemata that contains the inverse of all individuals in S. For example, if $S = \#01$, $S^I = \#10$.

each other out.

For these fully deceptive cases, gLINC will never be able to detect linkage between the final pair of groups.

Caveats

Given these two additional sources of inaccuracy in gLINC, one that can be solved by perturbing additional individuals, and one that can't, it's worth considering how artificial these situations are.

For non-binary representations, perturbations would not simply invert alleles, so s_{IJ} can never be fully determined from s at any point during the run of gLINC (equally for s_I and s_J). The fully deceptive case presented above exploits the fact that s_{IJ} must always be the inverse of s in particular circumstances, so it would be impossible to construct such a problem for non-binary representations.

For real-valued fitness functions, the probability of non-linearites cancelling each other out is likely to be smaller.

Finally, if the number of instances of linkage within composition groups is increased, we suggest the more non-linearities there are, the less likely they are to cancel each other out.

3.3 Future Work

Like order-2 perturbation and LINC, group perturbation and gLINC are generic linkage learning algorithms that could be applied to other specific linkage measures. One obvious extension of the work presented in this paper is to apply group perturbation to other linkage measures, such as non-monotonicity. With other measures the probability of two or more instances of linkage cancelling each other out will differ.

This work was motivated by the belief that a faster linkage-learner could be applied cost-effectively during GA runs. We envisage an algorithm that cycles between GA and linkage learning phases, searching composition groups with independent GAs. In such an algorithm accuracy would be less important than in a pre-search linkage learner, since composition groups could be joined, or broken apart, later on.

4. CONCLUSIONS

We have presented *group perturbation*, a linkage-learning perturbation mechanism applicable to strict decomposition problems, and *gLINC* — a modification of the LINC linkage-learning algorithm that exploits the group perturbation with a 'greedy' amalgamation of linked loci. This, together with a 'no repetition' mechanism, allows it to perform far faster than LINC at decomposition tasks.

A scale-up experiment on the shuffled 5-bit trap fitness function shows that gLINC performs significantly faster than LINC. A second experiment showed that this was true even when increasing the population size to maintain comparable accuracy. Additionally, comparison with oLINC shows that this improvement can be attributed to group perturbation rather than simply avoiding needless perturbations. Complexity analysis of the algorithm confirms that gLINC will never perform worse than LINC, but will perform *linearly* in the best-case.

An analysis of group perturbation for non-linearity detection exposes a fully deceptive case which gLINC could never fully solve. However, the probability of such cases occurring in real-world problems seems small since it depends on non-linearities localised to specific regions of a binary fitness landscape (inversely related schemata), that cancel each other out.

5. ACKNOWLEDGMENTS

David Coffin's research is partially funded by two EPSRC awards, CLHB GR/P03247/01 and CLGN EP/P500559/1.

6. REFERENCES

[1] E. D. de Jong, R. A. Watson, and D. Thierens. On the complexity of hierarchical problem solving. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, volume 2, pages 1201–1208. ACM Press, 2005.

[2] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley, 1989.

[3] D. E. Goldberg. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 3:493–530, 1989.

[4] D. E. Goldberg and M. Munetomo. Designing a Genetic Algorithm Using the Linkage Identification by Nonlinearity Check, IlliGAL Report No. 98014, Illinois Genetic Algorithm Laboratory, 1998.

[5] M. Munetomo and D. E. Goldberg. Identifying linkage by Nonlinearity Check, IlliGAL Report No. 98012, Illinois Genetic Algorithm Laboratory, 1998.

[6] M. Munetomo and D. E. Goldberg. Linkage Identification by Non-Monotonicity Detection for Overlapping Functions, IlliGAL Report No. 99005, Illinois Genetic Algorithm Laboratory, 1999.

[7] G. R. Harik and D. E. Goldberg. Learning linkage. In *Proceedings of Foundations of Genetic Algorithms 4*, pages 247–262. Morgan Kaufmann, 1997.

[8] J. H. Holland. *Adaptation in natural artificial systems*. University of Michigan Press, 1975.

[9] R. A. Watson and J. B. Pollack. A computational model of symbiotic composition in evolutionary transitions. *BioSystems* 69(2-3):187–209, 2003.

[10] M. Tezuka, M. Munetomo, and K. Akama. Linkage identification by nonlinearity check for real-coded genetic algorithms. In *Proceedings of the 2004 conference on Genetic and evolutionary computation*, volume 2, pages 222–233. Kluwer Academic Publishers, 2004.

[11] M. Tsuji, M. Munetomo, and K. Akama. Metropolitan Area Network Design Using GA Based on Hierarchical Linkage Identification, Genetic and Evolutionary Computation. In *Proceedings of the 2003 conference on Genetic and evolutionary computation*, volume 2, pages 1616–1617. Springer, 2003.

[12] M. Tsuji, M. Munetomo, and K. Akama. Modeling dependencies of loci with string classification according to fitness differences. In *Proceedings of the 2004 conference on Genetic and evolutionary computation*, volume 2, pages 246–257. Kluwer Academic Publishers, 2004.