Pattern-based Preservation of Building Blocks in Genetic Algorithms

Yoshitaka Kameya, Chativit Prayoonsri
Graduate School of Information Science and Engineering, Tokyo Institute of Technology
Ookayama 2-12-1, Meguro-ku, Tokyo, Japan 152-8552
Email: kameya@mi.cs.titech.ac.jp, chativit@mi.cs.titech.ac.jp

Abstract—**As stated in the building block hypothesis, we expect genetic algorithms (GAs) to create building blocks (BBs) and combine them appropriately in the evolutionary process. However, such BBs are often destroyed by unwanted crossovers, soon after they are created. Also, we may suffer from a "loose" encoding of chromosomes since BBs are in general unknown. In this paper, we propose a framework named GAP (GA with patterns), in which key patterns are extracted from significantly "good" chromosomes and protect such key patterns against unwanted crossover. GAP is applicable to optimization problems with fixed-point encoding and permutation encoding in a uniform fashion, and unlike perturbation-based linkage learning methods, GAP does not require extra fitness evaluations. Experimental results with the royal road problems and traveling salesman problems show the performance improvement of GAP over standard GAs.**

I. Introduction

As stated in the building block hypothesis [1], [2], we expect genetic algorithms (GAs) to create building blocks (BBs) and combine them appropriately in the evolutionary process. However, such BBs are often destroyed by unwanted crossovers, soon after they are created. Also, since BBs are in general unknown, we may have a "loose" encoding of chromosomes, where some BBs are scattered over a chromosome.

To cope with this difficulty, in the literature, linkage learning methods have been developed intensively, and they are classified into three categories [3], [4]: (1) linkage adaptation methods such as messy GA [5] and LLGA [6], (2) perturbation-based linkage identification methods such as LINC [7], LIMD [3], D^5 [8] and ILI [9], and (3) model-building methods such as BOA [10] and DSMGA [4]. These linkage learning methods basically assume a fixed-position (binary) encoding of chromosomes and BBs are identified/protected in a locus-wise fashion. EHBSA [11], a probabilistic model-building GA based on a Markov chain, can exceptionally deal with permutation encoding.

In this paper, we propose a framework named GAP (GA with patterns), which can be viewed as a successor of Gero and Kazakov's genetic engineering approach [12], [13]. That is, in GAP, we extract key patterns (substrings with gaps) from significantly "good" chromosomes and protect such key patterns against unwanted crossover. What is new in GAP is the use of sequential pattern mining, which enables us to handle permutation encoding, without repair operator, as well as fixed-position encoding in a uniform fashion. In addition, based on the extracted patterns, we can perform fine-grained (allele-wise) protection of BBs (like LEGO [14]), depending on the parents at each crossover (like CDC [15]). By this feature, overlapping BBs can also be treated naturally in GAP. Moreover, unlike perturbation-based methods, GAP does not require extra fitness evaluations, and rather, by accelerating the evolutionary process, the burden of fitness evaluations could be reduced. Another advantage of GAP is its comprehensibility. Under Chen et al.'s classification [16], both GAs and GAP are classified into a unimetric, physical-linkage and distributed-model approach, so we believe that GAP is still biologically understandable like GAs (as mentioned above, GAP is an instance of genetic engineering approach). Furthermore, the extracted patterns would make it easier to conduct a post-analysis of what happened in the evolutionary process.

The rest of this paper is organized as follows. In Section II, we describe GAP. In particular, Section II-A and Section II-B respectively describe how GAP incorporates a pattern mining technique, and how BBs are protected in crossover. The experimental results are shown in Section III, and finally we mention the related work in Section IV and conclude the paper in Section V.

II. Genetic algorithm with patterns

Fig. 1 roughly outlines the GAP framework. GAP basically follows the standard GA, except that Step 2b and Step 3 are augmented to extract the patterns Π frequently appearing in "good" chromosomes, and that crossover is performed adaptively to Π. Π is extracted from a population Δ_{mine} with very high fitness while Δ_{sel} is obtained as in the standard GA ($\Delta_{\mathrm{mine}} \subseteq \Delta_{\mathrm{sel}}$). Now patterns Π can be seen as *induced* BBs. Currently we adopt a generational replacement strategy with truncation selection and do not change the mutation operation. Step 3 is called the *mining step* and the protection procedure of BBs in the crossover in Step 4 is called the *protection step*. In the next two subsections, we describe the details of the mining step and the protection step, in turn.

A. Finding frequent closed sequence patterns

1) Frequent patterns as induced building blocks: To implement the mining step, we currently adopt an algorithm called BIDE [17], an extension of a well-known mining algorithm PrefixSpan [18], to find the building blocks efficiently. BIDE enumerates all frequent *closed* subsequences in a set Δ of

1) Initialize the population as $\Delta^{(0)}$ and let $t := 0$.
2) From $\Delta^{(t)}$, obtain the following two sets of chromosomes by truncation selection:
   a) Δ_{sel} with truncation rate r_{sel},
   b) Δ_{mine} with truncation rate r_{mine} ($r_{\text{mine}} \leq r_{\text{sel}}$).
3) Extract patterns Π from Δ_{mine}.
4) To Δ_{sel}, apply a crossover based on Π and a mutation, and then obtain a new population $\Delta^{(t+1)}$.
5) Let $t := t + 1$.
6) If some termination condition is met, then stop; otherwise go to Step 2.

Fig. 1. Outline of the GAP framework.

sequences in a depth-first fashion. In the context of GAP, Δ corresponds to a population and each sequence in Δ corresponds to a chromosome. Now we define some terminology and notation. Given a sequence $s = \langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$, a *subsequence* s' of s is a sequence $\langle \beta_1, \beta_2, \ldots, \beta_m \rangle$ where there exist some $i_1, i_2, \ldots, i_m$ such that $1 \leq i_1 < i_2 < \cdots < i_m \leq n$ and $\beta_1 = \alpha_{i_1}$, $\beta_2 = \alpha_{i_2}$, …, $\beta_m = \alpha_{i_m}$ ($m > 0$)[1]. For example, $\langle a \rangle$, $\langle b \rangle$, $\langle c \rangle$, $\langle a, b \rangle$, $\langle a, c \rangle$, $\langle c, b \rangle$ and $\langle a, c, b \rangle$ are subsequences of $\langle a, c, b \rangle$. If s' is a subsequence of s, we say "s' *occurs* in s" and denote it by $s' \subseteq s$. On the other hand, s is called a *supersequence* of s'. If $s' \subseteq s$ but $s' \neq s$, we write $s' \subset s$.

For permutation encoding, a chromosome can be simply a sequence of alleles, and for fixed-position encoding, following the encoding scheme in messy GA and LLGA, we represent a chromosome as a sequence of locus-allele pairs in which a locus-allele pair (l, a) precedes another pair (l', a') in the sequence when $l < l'$. For example, while a permutation $\langle a, c, d, e, b \rangle$ is treated as it is, a binary-encoded chromosome $\langle 1, 0, 1, 1, 0 \rangle$ is translated into a sequence $\langle (1, 1), (2, 0), (3, 1), (4, 1), (5, 0) \rangle$.

Furthermore, let s be a sequence called a *pattern*. Also let $\sigma(s, \Delta)$ be the number of sequences in Δ which are supersequences of s. Then, $\sigma(s, \Delta)$ is called the *support* of s in Δ, and given a population Δ and some threshold $\sigma_{\min} > 0$, a pattern s is said to be *frequent* if $\sigma(s, \Delta) \geq \sigma_{\min}$, i.e. it occurs in Δ for at least $\sigma_{\min}$ times. $\sigma_{\min}$ is called the *minimum support*. If the context is clear, $\sigma(s, \Delta)$ is abbreviated as $\sigma(s)$. If $\Delta (= \Delta_{\text{mine}})$ is a set of significantly "good" chromosomes and $\sigma_{\min}$ is sufficiently high, frequent patterns can be regarded as building blocks. Besides, from the above definition, a pattern is allowed to have gaps. For example, a permutation $\langle a, c, d, e, b \rangle$ can match with patterns like $\langle a, c \rangle$ or $\langle a, d \rangle$, and a translated sequence of a fixed-position chromosome $\langle (1, 1), (2, 0), (3, 1), (4, 1), (5, 0) \rangle$ can match with a pattern $\langle (2, 0), (5, 0) \rangle$, which corresponds to $\langle *, 0, *, *, 0 \rangle$ in schema notation. Thus we can have flexible patterns for both fixed-position encoding and permutation encoding.

[1]Although α_j's and β_j's are sets of objects (called items) in the original description on BIDE, we only consider a simpler case.

2) Use of closed patterns: As mentioned above, BIDE has the ability of finding frequent patterns which are closed. A closed subsequence s is a subsequence whose proper supersequences s' of s are less frequent than s (i.e. $\sigma(s') < \sigma(s)$). In other words, a sequence s is not closed if there is a proper supersequence s' of s such that $\sigma(s') = \sigma(s)$.

Closed patterns fit to our purpose since they are the most informative among the patterns with the same occurrences. In addition, in the context of GAP, it is important to note that the mining process of closed patterns is efficient even for the population with long and similar chromosomes. To see this, let us suppose that a whole chromosome $\langle a, b, c, d, e \rangle$ occurs for more than $\sigma_{\min}$ times in the population (i.e. $\langle a, b, c, d, e \rangle$ is frequent). Then, all of its subsequences ($\langle a \rangle$, $\langle a, b \rangle$, $\langle a, c \rangle$, and so on) are also frequent. We therefore can have exponentially more frequent patterns than chromosomes. On the other hand, we would only have a reasonable number of frequent closed patterns since any proper subsequence s of $\langle a, b, c, d, e \rangle$ are not closed unless $\sigma(s) > \sigma(\langle a, b, c, d, e \rangle)$. Furthermore, BIDE performs a dynamic checking of the closedness of the pattern candidates, which enables an aggressive (but safe) pruning of the search space. At a later stage of the evolution, the chromosomes tend to be similar to each other, so the use of closed patterns seems indispensable in GAP.

3) Constraints in chromosomes and patterns: BIDE can use two types of constraints to make a further optimization. The first one comes from the encoding we use. For instance, for both permutation encoding and fixed-position encoding, we know that every pattern occurs exactly once in a chromosome. Furthermore, for fixed-position encoding, a locus-allele pair (l, a) should not occur after (l', a') when $l < l'$, and there is no locus-allele pair between (l, a) and $(l + 1, a')$. Exploiting these constraints, BIDE was modified to skip some routine for closedness checking. The second type is the constraints on patterns, which are defined by the user depending on the nature of the optimization problem. Currently we can specify four constraints: minimum length $L_{\min}$, maximum length $L_{\max}$, minimum gap width $G_{\min}$, maximum gap width $G_{\max}$. The last three constraints can contribute to the speedup of BIDE. To correctly handle these constraints, we made a slight modification of the routine for closedness checking in BIDE (details are omitted). For example, it seems always reasonable to specify $L_{\min} = 2$, and we would have local patterns in permutation by specifying $G_{\max} = 1$ or 2.

4) Mining top-K patterns: It is known well that the number of frequent patterns to find is quite sensitive to the setting of the minimum support $\sigma_{\min}$. However, it is not easy to find an appropriate $\sigma_{\min}$ and adjust it manually in the middle of the evolution. That is, too small $\sigma_{\min}$ will cause a flood of frequent patterns, whereas with too large $\sigma_{\min}$, we can find nothing. So we adopt a simple top-K pattern mining technique, called minimum-support raising [19], which only returns to us only the most frequent K patterns by automatically adjusting $\sigma_{\min}$. That is, we start the search with very low minimum support $\sigma_{\min}^{(0)}$ (i.e. $\sigma_{\min} := \sigma_{\min}^{(0)}$), and once we have obtained K candidate patterns during the search, we update $\sigma_{\min} := \sigma(s)$

where s is the least frequent pattern in these top-K candidates. We repeatedly make this updating every time a new pattern is added to the list of top-K candidates, and then the minimum support will be raised. Hereafter, $\sigma_{\min}^{(0)}$ is referred to as the *initial minimum support*.

In many cases, the minimum support is quickly raised, so the overhead due to the top-K mining seems not so high, and would be canceled by the merit that it is much easier for the user to specify a pair of K and $\sigma_{\min}^{(0)}$, than to specify $\sigma_{\min}$. Also we can see that, when combined with BIDE, minimum-support raising is safe. This is because, as mentioned before, BIDE performs a dynamic checking of the closedness, and there are no non-closed patterns in the list of top-K candidates.

B. Soft protection of induced building blocks against unwanted crossover

Once we have induced the BBs from the population of significantly "good" chromosomes, the next problem is how to transfer the induced BBs to the next generation, combining them appropriately. To realize this, in the protection step, we modify the probability distribution over crossover points based on the positions of such induced BBs in two parent chromosomes. This soft approach is taken because the induced BBs may not be correct especially at an early stage of the evolution. Our crossover operator resembles CDC (context-dependent crossover) [15] in that the probability distribution over crossover points is decided at every crossover, depending on the alleles in the parent chromosomes. By this mechanism, like CDC, we can naturally deal with overlapping building blocks.

The rest of this section describes how to modify the distribution in each of single-point crossover (1PTX, for short in this paper), two-point crossover (2PTX), the original edge recombination (ER) and the position-based crossover (PX) [20]. For simplicity, we take a common approach to these crossover operators: we first discount the probability mass from unwanted crossover points, and re-distribute the discounted probability mass to the other (preferred) crossover points. This approach is similar to the strategy taken in Sebag and Schoenauer's crossover control [21].

Before starting, let us add some notation. Suppose that we are given $\pi = \langle \beta_1, \beta_2, \ldots, \beta_m \rangle$, a pattern or an induced BB obtained in the mining step. When π occurs in a chromosome c ($\pi \subseteq c$), $\mathrm{Occ}(\pi, c)$ indicates a set of loci governed by π in the chromosome c. That is, when $c = \langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$ and $\beta_1 = \alpha_{i_1}$, $\beta_1 = \alpha_{i_1}$, $\ldots$, $\beta_m = \alpha_{i_m}$ holds, we have $\mathrm{Occ}(\pi, c) = \{i_1, i_2, \ldots, i_m\}$. For example, for $\pi = \langle b, d \rangle$ and $c = \langle a, b, c, d \rangle$, we have $\mathrm{Occ}(\pi, c) = \{2, 4\}$.

1) Soft protection in single-point crossover: Let us first consider the case of 1PTX, where two parent chromosomes c_1 and c_2 of length n are given. Then, we have $N = (n-1)$ crossover points $\{\phi_1, \phi_2, \ldots, \phi_N\}$, where ϕ_i is the crossover point between the i-th and the $(i+1)$-th loci in the parents. Also we write p_i as the probability of ϕ_i being chosen, and consider $\{p_i \mid 1 \le i \le N\}$ as a probability distribution over the crossover points $\{\phi_i \mid 1 \le i \le N\}$.

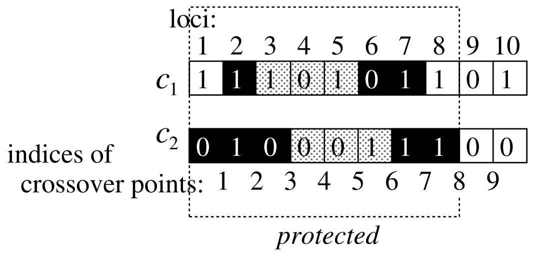

Fig. 2. Soft protection in single-point crossover.

Now suppose that we have obtained K patterns $\Pi = \{\pi_1, \pi_2, \ldots, \pi_K\}$ in the mining step. If none of these patterns occur in the parent chromosomes, then we perform 1PTX as usual, i.e. where the probabilities are unchanged as uniform ($p_i = 1/N$). Also if Π covers all positions of one parent chromosome, the crossover is entirely skipped. Otherwise, we make a soft protection as follows. First, for each parent chromosome c_u ($u = 1, 2$), the leftmost position $i_{\text{left}}^{(u)}$ and the rightmost position $i_{\text{right}}^{(u)}$ of these patterns is computed as $i_{\text{left}}^{(u)} = \mathrm{minOcc}(\Pi, c_u)$ and $i_{\text{right}}^{(u)} = \mathrm{maxOcc}(\Pi, c_u)$, where

$$\mathrm{Occ}(\Pi, c) = \bigcup_{1 \le k \le K : \pi_k \subseteq c} \mathrm{Occ}(\pi_k, c). \qquad (1)$$

Here we would like to protect the portion from the $i_{\text{left}}^{(u)}$-th locus to the $i_{\text{right}}^{(u)}$-th locus. To do this, letting $\Phi = \{\phi_i \mid i_{\text{left}}^{(1)} \le i < i_{\text{right}}^{(1)}\} \cup \{\phi_i \mid i_{\text{left}}^{(2)} \le i < i_{\text{right}}^{(2)}\}$, we first discount the probabilities for the crossover points in Φ as follows:

$$p_i := \frac{\delta}{N} \quad (\phi_i \in \Phi), \qquad (2)$$

where δ ($0 < \delta < 1$) is a user-defined control parameter called the *discount rate*. Then, the discounted probability mass $h(1-\delta)/N$ is re-distributed as follows:

$$p_i := \frac{1}{N} + \frac{1}{N-h} \cdot \frac{h(1-\delta)}{N} = \frac{N - h\delta}{N(N-h)} \quad (\phi_i \notin \Phi), \qquad (3)$$

where $h = |\Phi|$ is the number of protected crossover points.

As an illustration, let us suppose that we have two parent chromosomes c_1 and c_2 of length 10 in Figure 2. We also have nine crossover points indexed from 1 to 9 between loci ($N = 9$). Each of the black-colored bits indicates that one of the patterns in Π occurs at the position, and these black-colored bits in c_1 (resp. c_2) correspond to $\mathrm{Occ}(\Pi, c_1) = \{2, 6, 7\}$ (resp. $\mathrm{Occ}(\Pi, c_2) = \{1, 2, 3, 7, 8\}$). On the other hand, the shaded bits indicate a gap between such black-colored bits. From the definition, we have $i_{\text{left}}^{(1)} = \min\{2, 6, 7\} = 2$, $i_{\text{right}}^{(1)} = 7$, $i_{\text{left}}^{(2)} = 1$ and $i_{\text{right}}^{(2)} = 8$. Hence, as shown in Figure 2, we obtain $\Phi = \{\phi_1, \phi_2, \ldots, \phi_7\}$ as the set of indices of crossover points whose probabilities will be discounted. Also we have $h = |\Phi| = 7$. Finally, given some δ (e.g. $\delta = 0.5$), the distribution $\{p_1, p_2, \ldots, p_9\}$ over the crossover points are modified following Eqs. 2 and 3.

One may see that the shaded bits in Figure 2 are protected together with the black bits, while they are not included in the

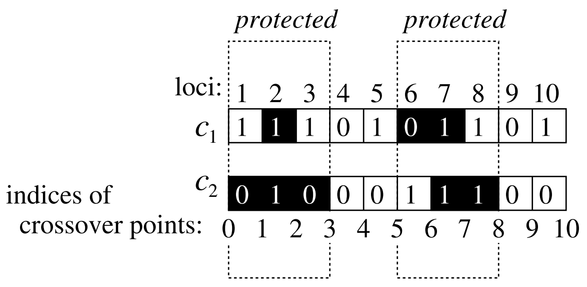

Fig. 3. Soft protection in two-point crossover.

induced BB. So the soft protection in 1PTX has a problem that the patterns (the induced BBs) could be over-protected, and this problem will be solved in 2PTX, as described next.

2) Soft protection in two-point crossover: In the case of 2PTX for two parent chromosomes c_1 and c_2 of length n, we have $N = n(n+1)/2$ crossover points $\{\phi_{ij} \mid 0 \leq i < j \leq n\}$, where ϕ_{ij} indicates that we swap the portion from the $(i+1)$-th locus to the j-th locus. Here, for brevity, we describe the protection method for 2PTX only by illustration. That is, Figure 3 depicts how the indices of the crossover points are given for two parent chromosomes c_1 and c_2 (length $n = 10$), and shows that the loci to be protected are $\mathrm{Occ}(\Pi, c_1) \cup \mathrm{Occ}(\Pi, c_2) = \{2, 6, 7\} \cup \{1, 2, 3, 7, 8\} = \{1, 2, 3, 6, 7, 8\}$. So differently from the case of 1PTX, we can swap the bits at the loci 4 and 5 as well as the loci 9 and 10. Furthermore, let us introduce $\Phi_{\text{full}} = \{\phi_{ij} \mid 0 \leq i < j \leq n\}$ as the set of all crossover points, $\overline{\Phi}_1 = \{\phi_{ij} \mid i, j \in \{3, 4, 5\}, i < j\}$ as the set of crossover points between the protected loci, and $\overline{\Phi}_2 = \{\phi_{ij} \mid i, j \in \{0, 8, 9, 10\}, i < j\}$ as the set of crossover points outside the protected loci. Finally, we get the set Φ of the crossover points to be protected by $\Phi = \Phi_{\text{full}} \setminus (\overline{\Phi}_1 \cup \overline{\Phi}_2)$. Then, the probability distribution $\{p_{ij} \mid 0 \leq i < j \leq n\}$ over the crossover points is modified similarly to the case of 1PTX:

$$p_{ij} := \frac{\delta}{N} \quad (\phi_{ij} \in \Phi) \tag{4}$$

$$p_{ij} := \frac{N - h\delta}{N(N-h)} \quad (\phi_{ij} \notin \Phi) \tag{5}$$

where δ is the discount rate and $h = |\Phi|$ as defined above. It is not difficult to generalize the above procedure for computing Φ. Comparing Fig. 2 and Fig. 3, we can easily see that the protected part is kept minimal in the case with 2PTX.

3) Soft protection in edge recombination: Edge recombination is used as a crossover operator for permutation encoding. Here we consider parent chromosomes of length n with permutation encoding, where each of $\{1, 2, \ldots, n\}$ appears exactly once as an allele in a chromosome, e.g. we have $\langle 2, 4, 3, 1, 5, 6 \rangle$ as a chromosome of length 6. Hereafter, in consideration of traveling salesman problems, we regard each allele as a city. In ER, for a pair of parent chromosomes, we first build a table called a *edge map* that records the cites connected to each city. $M[\gamma]$ denotes the set of cities connected to a city γ, and is called a *edge list* of γ. For example, following [20], let us consider two parent chromosomes $\langle 1, 2, 3, 4, 5, 6 \rangle$ and $\langle 2, 4, 3, 1, 5, 6 \rangle$. Then, the edge map is built as follows:

City γ	Connected cities $M[\gamma]$
1	{2, 3, 5, 6}
2	{1, 3, 4, 6}
3	{1, 2, 4}
4	{2, 3, 5}
5	{1, 4, 6}
6	{1, 2, 5}

Using an edge map like above, ER works as shown in Fig. 4, where $\langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$ is the offspring to be returned. One may see that there are random choices in Steps 4 and 5, so we modify the probability distribution for each choice using the information from the patterns (the induced BBs) Π.

```
1) Choose the initial city from one of two parent chromosomes. Let $\gamma$ be this initial city, $t := 1$ and $\alpha_1 := \gamma$.
2) Remove $\gamma$ from $M[\gamma']$ (if possible) for all $1 \leq \gamma' \leq n$.
3) If $M[\gamma] \neq \emptyset$ (i.e. there is some edge to another city) then go to Step 4; otherwise go to Step 5.
4) Choose a city at random from the cities in $M[\gamma]$ that have the smallest edge list. Let $\gamma$ be the chosen city. Go to Step 6.
5) If $t = n$ (i.e. there is no unvisited city), then return the offspring $\langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$; otherwise, choose at random an unvisited city from $\{1, 2, \ldots, n\} \setminus \{\alpha_1, \ldots, \alpha_t\}$. Let $\gamma$ be the chosen city. Go to Step 6.
6) $t := t + 1$ and then $\alpha_t := \gamma$. Go to Step 2.
```

Fig. 4. Edge recombination, which returns an offspring $\langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$.

To achieve this, we first build another table M^* called a *frequent edge map*. Now suppose that we have a chromosome $c = \langle \alpha_1, \alpha_2, \ldots, \alpha_n \rangle$ in the population Δ. Then, the edges covered by the patterns Π are obtained by $E_c^* = \{(\alpha_i, \alpha_{i+1}) \mid (i, i+1) \in \mathrm{Occ}(\Pi, c)\}$, where $\mathrm{Occ}(\Pi, c)$ is defined in Eq. 1. The edges in E_c^* are called *frequent edges* in c. Finally we obtain the list of frequent edges including a city γ as $M^*[\gamma] = \bigcup_{c \in \Delta} \{\gamma' \mid (\gamma, \gamma') \in E_c^* \text{ or } (\gamma', \gamma) \in E_c^*\}$.

After obtaining M^*, we modify the probability distribution for the choice of the next city in Step 4 and 5. Let γ be the current city, $\Gamma = \{\gamma_1, \gamma_2, \ldots, \gamma_N\}$ be a set of the candidates for the next city, and $\{p_1, p_2, \ldots, p_N\}$ are the probability distribution where we choose γ_i with probability p_i. In the original edge recombination, we have $p_i = 1/N$. In GAP, on the other hand, for a city $\gamma_i \in M^*[\gamma]$, we discount the probability of γ_i *not* being chosen. That is, we perform:

$$p_i := 1 - \left(1 - \frac{1}{N}\right) \delta \quad (\gamma_i \in M^*[\gamma]) \tag{6}$$

$$p_i := 1 - \left(1 - \frac{1}{N}\right) \frac{N - h\delta}{N - h} \quad (\gamma_i \notin M^*[\gamma]) \tag{7}$$

where δ is the discount rate and $h = |M^*[\gamma]|$.

4) Soft protection in position-based crossover: PX is another crossover operator for permutation encoding. Fig. 5 (left) shows how an offspring c_1' is created from two parent

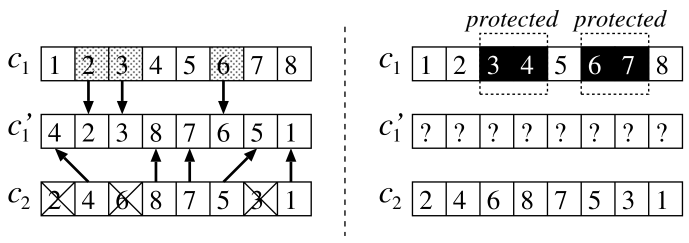

Fig. 5. Position-based crossover.

chromosomes c_1 and c_2 (this example is taken from [20]). In the first step of PX, we choose several loci (shaded in the figure) in one parent c_1, and the chosen cities in these loci are copied into the offspring, with the positions unchanged. Then, in the second step, the remaining cities are copied from the other parent c_2, with the order unchanged. We create another offspring by swapping the roles of c_1 and c_2.

In GAP, using the information from the patterns Π, we aim to give a bias to the choice of the loci in the first step. Now let us consider to choose approximately $(n{\cdot}r_{\mathrm{pos}})$ loci in the parent c_1, where n is the length of chromosomes and r_{pos} is a new control parameter called the the *position-keeping rate*. If we do not give any bias, this is achieved by choosing each locus i independently with probability $p_i = r_{\mathrm{pos}}$ $(1 \leq i \leq n)$. On the other hand, using $\mathrm{Occ}(\Pi, c_1)$, a set of loci to be protected, we modify the probabilities p_i as follows:

$$p_i \ := \ \kappa \cdot (1 - (1 - r_{\mathrm{pos}})\delta) \qquad (i \in \mathrm{Occ}(\Pi, c_1)) \quad (8)$$

$$p_i \ := \ \kappa \cdot r_{\mathrm{pos}}\delta \qquad (i \notin \mathrm{Occ}(\Pi, c_1)) \quad (9)$$

where κ is adjusted so that we can choose $(n \cdot r_{\mathrm{pos}})$ loci on average. For example, if we have $\mathrm{Occ}(\Pi, c_1) = \{3, 4, 6, 7\}$, the black-colored loci in Fig. 5 (right) are the ones to be protected. In Eq. 8, the probability of the protected locus i not being chosen is discounted, while in Eq. 9, the probability of the unprotected locus i being chosen is discounted.

III. Experiments

In the comparative experiments, we picked up two kind of problems: the royal road problems with fixed-position encoding, and the traveling salesman problems with permutation encoding. The royal road problems were firstly introduced by Mitchell et al. [2], and for the traveling salesman problems, Larrañaga et al. provided an elaborate survey which includes an in-depth empirical comparison with various crossover and mutation operators, using standard benchmark datasets [20]. In both types of problems, we basically compare the standard GA (SGA, for short) and GAP to see whether or how the protection of the induced BBs can improve the performance. Additionally, for the royal road problems, BOA (Bayesian Optimization Algorithm) [10], a popular probabilistic model building GA method based on Bayesian networks, is compared.

A. Royal road problems

A royal road problem is to optimize the fitness function for a chromosome c, defined as $F(c) = \sum_{s \in S} w(s)\sigma(s, c)$, where S is the set of user-defined schemata, $w(s)$ is the weight of the schema $s \in S$, and $\sigma(s, c)$ takes 1 if the schema s occurs in the chromosome c; 0 otherwise. In our comparative experiments with the royal road problems, we used two fitness functions under "tight" encoding and "loose" encoding. These two fitness function commonly have 15 schemata, and the schemata and their weights are defined in Fig. 6. From this figure, we can see that the schemata in "tight" encoding are formed by the neighboring bits, and so the destruction of the schemata is less likely to occur. On the other hand, in "loose" encoding, the schemata are easily destroyed by crossover since the bits in a schema are apart from each other.

TABLE I
Control parameters for the royal road problems.

Method	Control parameter	Value
SGA/GAP	Crossover rate	0.7
	Mutation rate	0.01
	Truncation rate (r_{sel})	0.5
GAP	Max. # of patterns (K)	30
	Min. length of patterns ($L_{\min}$)	3
	Mining rate (r_{mine})	0.05, 0.1, 0.2
	Discount rate (δ)	0.1, 0.3, 0.5, 0.7, 0.9
	Initial min. support ($\sigma^{(0)}_{\min}$)	$\max\{5, (0.2 \cdot r_{\mathrm{mine}}\lvert\mathrm{Pop}\rvert)\}$
BOA	Truncation rate	0.01, 0.05, 0.1, 0.2
	Max. # of parents	1, 2, 5
	Pseudo count (C)	0.01, 0.05, 0.1, 0.5, 1

We varied the population size, indicated by $\lvert\mathrm{Pop}\rvert$, from 64 to 512, and compared five evolutionary methods: SGA with 1PTX, SGA with 2PTX, GAP with 1PTX, GAP with 2PTX, and BOA. It is obvious that 1PTX and 2PTX have high position dependency, so in this experiment, we would like to observe whether the protection of the induced BB can alleviate this position dependency.[2] In Mitchell et al.'s experiments, the population size is fixed at 128, and they use the roulette wheel selection with fitness scaling. On the other hand, BOA originally uses the truncation selection, and hence the truncation selection is used for comparison. Furthermore, both SGA and GAP use uniform mutation. The evolutionary cycle is repeated until reaching the optimal fitness 256 or 5,000 generations.

The setting of the user-defined control parameters is shown in Table I. As in Table I, we used more than one value for some of the control parameters, and the results with the best parameter values are presented in this paper. In GAP, we conducted top-K mining with the initial minimum support $\max\{5, (0.2 \cdot r_{\mathrm{mine}}\lvert\mathrm{Pop}\rvert)\}$, which indicates that the patterns need to appear in at least 20% of Δ_{mine}, the chromosomes which have been selected for pattern mining (Fig. 1). For a small population, on the other hand, the initial minimum support is fixed at 5.

The control parameters of BOA are also shown in Table I. The pseudo count C in the table corresponds to a

[2]Accordingly we did not try the case with uniform crossover and the case with no crossover operation in this experiment. The comparison including these cases is left as future work.

s_1 = 11111111**; $w(s_1)$ = 8
s_2 = ********11111111**; $w(s_2)$ = 8
s_3 = ****************11111111**; $w(s_3)$ = 8
s_4 = ************************11111111********************************; $w(s_4)$ = 8
s_5 = ********************************11111111************************; $w(s_5)$ = 8
s_6 = **11111111****************; $w(s_6)$ = 8
s_7 = **11111111********; $w(s_7)$ = 8
s_8 = **11111111; $w(s_8)$ = 8
s_9 = 1111111111111111**; $w(s_9)$ = 16
s_{10} = ****************1111111111111111********************************; $w(s_{10})$ = 16
s_{11} = ********************************1111111111111111****************; $w(s_{11})$ = 16
s_{12} = **1111111111111111; $w(s_{12})$ = 16
s_{13} = 11111111111111111111111111111111********************************; $w(s_{13})$ = 32
s_{14} = ********************************11111111111111111111111111111111; $w(s_{14})$ = 32
s_{15} = 1111111111111111111111111111111111111111111111111111111111111111; $w(s_{15})$ = 64

s_1 = 1*******1*******1*******1*******1*******1*******1*******1*******; $w(s_1)$ = 8
s_2 = *1*******1*******1*******1*******1*******1*******1*******1******; $w(s_2)$ = 8
s_2 = **1*******1*******1*******1*******1*******1*******1*******1*****; $w(s_3)$ = 8
s_4 = ***1*******1*******1*******1*******1*******1*******1*******1****; $w(s_4)$ = 8
s_5 = ****1*******1*******1*******1*******1*******1*******1*******1***; $w(s_5)$ = 8
s_6 = *****1*******1*******1*******1*******1*******1*******1*******1**; $w(s_6)$ = 8
s_7 = ******1*******1*******1*******1*******1*******1*******1*******1*; $w(s_7)$ = 8
s_8 = *******1*******1*******1*******1*******1*******1*******1*******1; $w(s_8)$ = 8
s_9 = 1***1***1***1***1***1***1***1***1***1***1***1***1***1***1***1***; $w(s_9)$ = 16
s_{10} = *1***1***1***1***1***1***1***1***1***1***1***1***1***1***1***1**; $w(s_{10})$ = 16
s_{11} = **1***1***1***1***1***1***1***1***1***1***1***1***1***1***1***1*; $w(s_{11})$ = 16
s_{12} = ***1***1***1***1***1***1***1***1***1***1***1***1***1***1***1***1; $w(s_{12})$ = 16
s_{13} = 1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*; $w(s_{13})$ = 32
s_{14} = *1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1*1; $w(s_{14})$ = 32
s_{15} = 1111111111111111111111111111111111111111111111111111111111111111; $w(s_{15})$ = 64

Fig. 6. (above) Royal road function with "tight" encoding, and (below) with "loose" encoding.

prior information about the number of chromosomes in the population, which is denoted by $m'(x_i, \pi_{X_i})$ in the original paper [10]. In our implementation, the conditional probabilities are also estimated using this pseudo count.[3] Furthermore, we tried the addition of edges at most 1,000 times until the acyclicity satisfied, and tried to create at most 100 networks until the model score (the BD metric) improved. For all evolutionary methods including BOA, we performed 100 trials with different seeds of random numbers.

Table II (above) shows the results for the royal road problem with "tight" encoding. Here, each entry is the average number of generations to achieve the optimal fitness, with its standard error. We can see from this table that 2PTX generally works better than 1PTX, and the performance gets better with a larger population, as expected. What is not expected is that GAP with 1PTX performs worse than SGA with 1PTX. One possible reason is that, in GAP with 1PTX, the induced BBs might have been over-protected by the modification of the probability distribution over crossover points. On the other hand, with 2PTX, GAP outperforms SGA for all sizes of population. We can further see that BOA steadily works, but the performance

TABLE II
COMPARATIVE RESULTS FOR TWO ROYAL ROAD PROBLEMS.

Tight	SGA		GAP		BOA
|Pop|	1PTX	2PTX	1PTX	2PTX	
64	684.6 ± 36.9	548.4 ± 35.6	624.5 ± 31.4	538.2 ± 27.5	384.9 ± 20.6
128	266.5 ± 16.0	207.5 ± 14.5	284.3 ± 18.8	186.0 ± 13.7	309.8 ± 14.3
256	89.5 ± 7.1	74.4 ± 6.7	132.3 ± 11.2	55.6 ± 4.7	155.8 ± 7.9
512	28.0 ± 3.5	28.9 ± 2.4	49.3 ± 4.9	23.7 ± 0.8	85.4 ± 3.2

Loose	SGA		GAP		BOA
|Pop|	1PTX	2PTX	1PTX	2PTX	
64	578.2 ± 26.3	537.3 ± 23.5	577.9 ± 27.5	541.7 ± 26.0	371.0 ± 21.6
128	294.4 ± 13.4	288.6 ± 11.7	325.4 ± 13.6	285.6 ± 12.0	312.2 ± 14.1
256	185.8 ± 7.1	164.4 ± 7.6	208.8 ± 6.5	151.8 ± 6.1	145.5 ± 5.6
512	110.4 ± 4.0	97.9 ± 3.1	135.6 ± 5.3	95.0 ± 3.7	78.8 ± 2.9

improvement with respect to the population size is slower than that of traditional GA methods including GAP.

Table II (below), on the other hand, shows the results for the case with "loose" encoding. This table firstly shows that the performance of traditional GA methods degrades compared to the case with "tight" encoding, while the performance of BOA does not change. This is because the performance of 1PTX and 2PTX heavily depends on the encoding style, while BOA uniformly treats all positions in a chromosome by its design. Despite this, with 2PTX, GAP works better than SGA for a larger population, and is still comparable with BOA. So we can say that the protection of the induced BBs alleviates the position dependency of 2PTX at least to some extent. It should

[3]To be more specific, we estimated the conditional probabilities by:

$$p(x_i \mid \pi_{X_i}) := \frac{m(x_i, \pi_{X_i}) + C}{m(\pi_{X_i}) + d_i C},$$

where x_i is an instantiation of the i-th variable X_i in the Bayesian network, π_{X_i} is an instantiation of the parent variables of X_i, $m(x_i, \pi_{X_i})$ is the number of chromosomes satisfying both x_i and π_{X_i}, $m(\pi_{X_i}) = \sum_{x_i} m(x_i, \pi_{X_i})$, and d_i is the number of distinct values of X_i.

TABLE III
CONTROL PARAMETERS FOR THE TRAVELING SALESMAN PROBLEMS.

Method	Control parameter	Value
SGA/GAP	Crossover rate	0.7
	Mutation rate	0.05
	Truncation rate (r_{sel})	0.5
	Position-keeping rate (r_{pos})	0.25 (for PX only)
GAP	Max. # of patterns (K)	30
	Min. length of patterns ($L_{\min}$)	2
	Max. gap width ($G_{\max}$)	0, 1, 2
	Mining rate (r_{mine})	0.01, 0.05, 0.1
	Discount rate (δ)	0.1, 0.3, 0.5, 0.7, 0.9
	Initial min. support ($\sigma^{(0)}_{\min}$)	$\max\{5, (0.2 \cdot r_{\mathrm{mine}}\lvert\mathrm{Pop}\rvert)\}$

be also noted that GAP does not work well with a smaller population, presumably due to the lack of sufficient amount of chromosomes to induce "correct" BBs.

B. Traveling salesman problems

In comparative experiments with traveling salesman problems, we use two crossover operators: the original edge recombination (ER) and the position-based crossover (PX), which are illustrated in [20]. We compared the evolutionary methods with Grötschels48 and Grötschels120, which are provided in TSPLIB95.[4] Grötschels48 and Grötschels120 respectively have 48 cities optimally connected by the route of length 5,046, and 120 cities optimally connected by the route of length 6,942. We varied the population size from 200 to 2000, and compared four methods: SGA with ER, SGA with PX, GAP with ER and GAP with PX. Both SGA and GAP use the truncation selection and the inverse mutation (which is also described in [20]). The evolutionary cycle is repeated until it reaches the optimal fitness or there is no change in the best fitness over the last 50 generations. Table III lists the control parameters we used. For the traveling salesman problems, we newly introduced a constraint on the maximum gap width to extract promising local routes as the induced BBs. Similarly to the case of the royal road problems, we performed 100 trials with different seeds of random numbers. In the experiment, we did not compare GAP with EHBSA (Edge Histogram Based Sampling Algorithm) [11], a probabilistic model building GA for permutation encoding, since these methods adopt different replacement strategies: the original EHBSA adopts a steady-state strategy while GAP adopts a generational one.

Table IV shows the results for Grötschels48. Each entry in Table IV (above) is the best fitness averaged over the 100 trials, and each entry in Table IV (below) is the average number of generations until termination, under the same setting of control parameters. From the tables in Table IV, it can be seen that, in most cases, GAP improves the best fitness and requires a less number of generations until termination. So in this experiment, we can say here that GAP successfully accelerated the evolution. Since the number of fitness evaluations is proportional to the number of generations, we may expect that GAP reduces

[4]http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

TABLE IV
COMPARATIVE RESULTS FOR GRÖTSCHELS48.

Fitness	SGA		GAP	
|Pop|	ER	PX	ER	PX
200	5441.7 ± 14.3	5665.1 ± 31.6	5462.1 ± 16.0	5663.2 ± 32.2
500	5152.0 ± 5.5	5326.2 ± 21.8	5143.7 ± 5.0	5262.0 ± 14.4
1000	5111.2 ± 4.2	5224.7 ± 12.7	5103.9 ± 4.2	5223.8 ± 11.7
2000	5098.3 ± 2.9	5189.0 ± 9.4	5081.5 ± 2.4	5192.1 ± 8.9
5000	5085.1 ± 2.0	5161.5 ± 4.8	5072.1 ± 1.8	5164.9 ± 6.1

Gens.	SGA		GAP	
|Pop|	ER	PX	ER	PX
200	157.8 ± 3.0	318.3 ± 25.7	154.1 ± 2.7	303.0 ± 21.1
500	165.1 ± 2.3	406.2 ± 45.9	163.8 ± 2.7	287.2 ± 12.6
1000	164.6 ± 1.5	358.4 ± 39.8	146.1 ± 2.2	422.5 ± 50.8
2000	166.1 ± 1.9	445.8 ± 55.0	135.8 ± 1.7	489.4 ± 60.9
5000	163.9 ± 1.9	467.5 ± 58.7	148.6 ± 3.0	511.8 ± 65.3

TABLE V
COMPARATIVE RESULTS FOR GRÖTSCHELS120.

Fitness	SGA		GAP	
|Pop|	ER	PX	ER	PX
200	9434.5 ± 34.8	9742.3 ± 69.1	9542.3 ± 38.5	9730.2 ± 85.4
500	8411.0 ± 29.1	8075.4 ± 34.2	8432.3 ± 23.1	8071.6 ± 33.6
1000	8258.1 ± 28.9	7733.1 ± 29.2	8254.1 ± 27.9	7694.2 ± 23.7
2000	8148.2 ± 34.8	7579.0 ± 21.1	8103.7 ± 21.6	7570.5 ± 21.9
5000	8012.6 ± 47.7	7489.2 ± 18.2	7805.3 ± 17.4	7477.2 ± 18.0

Gens.	SGA		GAP	
|Pop|	ER	PX	ER	PX
200	557.0 ± 8.7	488.2 ± 10.9	544.3 ± 8.9	513.5 ± 21.2
500	874.7 ± 20.1	528.2 ± 17.8	856.9 ± 19.8	490.5 ± 9.2
1000	1039.9 ± 20.8	574.2 ± 31.1	928.9 ± 20.4	553.6 ± 22.9
2000	1275.0 ± 29.4	649.7 ± 40.0	1044.3 ± 18.8	580.5 ± 29.4
5000	1561.8 ± 32.4	827.0 ± 57.4	1183.0 ± 22.6	856.0 ± 61.0

the burden of fitness evaluation. A similar tendency is observed more clearly in the case of Grötschels120, whose results are shown in Table V.

IV. RELATED WORK

The methods for pattern-based extraction/protection of BBs have been largely developed in the field of genetic programming (e.g. [22], [23], [24], [25], [26]), in which the BBs are often position-independent and the size of a chromosome can vary. This paper shows that these extraction/protection techniques can be applied to sequential chromosomes. Optimization problems with variable-length chromosomes can also be dealt with in GAP.

Gero and Kazakov's genetic engineering approach was firstly proposed in [12] and later fully described in [13]. Although the full description shows that their method can deal with both position-dependent and position-independent encodings, the treatments for these encodings are rather different. In GAP, on the other hand, extraction and protection of BBs are performed in a uniform fashion, thanks to frequent pattern mining techniques. In addition, Gero and Kazakov's method sophisticatedly handles a suffix tree to find BBs from the chromosomes in position-independent encoding, but unlike in GAP, such BBs must not contain the gaps.

In GA research, machine learning techniques have already been used in probabilistic modeling GAs and some of perturbation-based methods (e.g. [8], [9]), and Sebag and Schoenauer proposed a method for controlling crossover based on inductive learning [21]. On the other hand, to the best of our knowledge, GAP is the first attempt to introduce a frequent pattern mining technique into GAs.

V. Conclusion and future work

In this paper, we proposed a framework named GAP (GA with patterns), which can be viewed as a successor of Gero and Kazakov's genetic engineering approach. In GAP, we use an advanced technique for sequential pattern mining (BIDE with top-K pattern mining) to induce BBs from significantly "good" chromosomes, and modify the probability distribution over crossover points to enable a fine-grained (allele-wise) protection of the induced BBs against unwanted crossover. GAP can handle permutation encoding as well as fixed-position encoding, and the experimental results tell us that GAP can accelerate the evolutionary process and consequently reduce the number of fitness evaluations.

There is still a room for improvement in GAP. Instead of frequent pattern mining, it seems promising to adopt a more advanced pattern mining technique, such as emerging pattern mining [27]. That is, we extract patterns that frequently appear in "good" chromosomes but infrequently appear in "bad" chromosomes. This contrastive criterion filters out unimportant patterns and would achieve a more precise extraction of BBs. In addition, although GAP currently performs the mining step at every generation, it may be more efficient to perform the mining step only once in several generations, as Sebag and Schoenauer suggested [21]. Furthermore, in this paper, we have described GAP using a sequence mining algorithm for simplicity and ease of implementation. For fixed-position encoding, on the other hand, a chromosome can be seen as a *set* of locus-allele pairs, and hence the mining step would be more optimized using an advanced itemset mining method such as LCM [28].

Acknowledgment

The authors would like to thank the anonymous reviewers for their insightful and helpful comments.

References

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

[2] M. Mitchell, S. Forrest, and J. H. Holland, "The royal road for genetic algorithms: fitness landscapes and GA performance," in *Proc. of the 1st European Conf. on Artificial Life (ECAL-92)*, 1992, pp. 245–254.

[3] M. Munetomo and D. E. Goldberg, "Linkage identification by non-monotonicity detection for overlapping functions," *Evolutionary Computation*, vol. 7, no. 4, pp. 377–398, 1999.

[4] T.-L. Yu, D. E. Goldberg, K. Sastry, C. F. Lima, and M. Pelikan, "Dependency structure matrix, genetic algorithms, and effective recombination," *Evolutionary Computation*, vol. 17, no. 4, pp. 595–626, 2009.

[5] D. E. Goldberg, B. Korb, and K. Deb, "Messy genetic algorithms: motivation, analysis, and first results," *Complex Systems*, vol. 3, pp. 493–530, 1989.

[6] G. R. Harik and D. E. Goldberg, "Learning linkage," Dept. of General Engineering, University of Illinois at Urbana-Champaign, IlliGAL Report No.96006, 1996.

[7] M. Munetomo and D. E. Goldberg, "Identifying linkage by nonlinearity check," Dept. of General Engineering, University of Illinois at Urbana-Champaign, IlliGAL Report No.98012, 1998.

[8] M. Tsuji, M. Munemoto, and K. Akama, "Linkage identification by fitness difference clustering," *Evolutionary Computation*, vol. 14, no. 4, pp. 383–409, 2006.

[9] C.-Y. Chuang and Y.-p. Chen, "Linkage identification by perturbation and decision tree induction," in *Proc. of the 2007 IEEE Congress on Evolutionary Computation (CEC-07)*, 2007, pp. 357–363.

[10] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, "Linkage problem, distribution estimation, and Bayesian networks," *Evolutionary Computation*, vol. 8, no. 3, pp. 311–340, 2000.

[11] S. Tsutsui, "Probabilistic model-building genetic algorithms in permutation representation domain using edge histogram," in *Proc. of the 7th Int'l Conf. on Parallel Problem Solving from Nature (PPSN VII)*, 2002, pp. 224–233.

[12] J. S. Gero and V. Kazakov, "Evolving building blocks for genetic algorithms using genetic engineering," in *Proc. of the 1995 IEEE Int'l Conf. on Evolutionary Computation*, 1995, pp. 340–345.

[13] ——, "A genetic engineering approach to genetic algorithms," *Evolutionary Computation*, vol. 9, no. 1, pp. 71–92, 2001.

[14] J. Smith and T. C. Fogarty, "Recombination strategy adaptation via evolution of gene linkage," in *Proc. of the 1996 IEEE Int'l Conf. on Evolutionary Computation*, 1996, pp. 826–831.

[15] M. Tsuji, M. Munemoto, and K. Akama, "A crossover for complex building blocks overlapping," in *Proc. of the 2006 Genetic and Evolutioary Computation Conf. (GECCO-06)*, 2006, pp. 1337–1344.

[16] Y.-p. Chen, T.-L. Yu, K. Sastry, and D. E. Goldberg, "A survey of linkage learning techniques in genetic and evolutionary algorithms," University of Illinois at Urbana-Champaign, IlliGAL Report No.2007014, 2007.

[17] J. Wang and J. Han, "BIDE: Efficient mining of frequent closed sequences," in *Proc. of the 20th Int'l Conf. on Data Engineering (ICDE-04)*, 2004, pp. 79–90.

[18] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth," in *Proc. of the 17th Int'l Conf. on Data Engineering (ICDE-01)*, 2001, pp. 215–224.

[19] P. Tzvekov, X. Yan, and J. Han, "TSP: Mining top-K closed sequential patterns," in *Proc. of the 3rd Int'l Conf. on Data Mining (ICDM-03)*, 2003, pp. 347–354.

[20] P. Larrañaga, C. Kuijpers, R. Murga, I. Inza, and S. Dizdarevich, "Genetic algorithms for the travelling salesman problem: A review of representations and operators," *Artificial Intelligence Review*, no. 13, pp. 129–170, 1999.

[21] M. Sebag and M. Schoenauer, "Controlling crossover through inductive learning," in *Proc. of the 3rd Int'l Conf. on Parallel Problem Solving from Nature (PPSN III)*, 1994, pp. 209–218.

[22] P. J. Angeline and J. B. Pollack, "The evolutionary induction of subroutines," in *Proc. of the 14th Conf. of the Cognitive Science Society*, 1992, pp. 236–241.

[23] W. A. Tackett, "Mining the genetic program," *IEEE Expert*, vol. 10, no. 3, pp. 28–38, 1995.

[24] J. P. Rosca and D. H. Ballard, "Discovery of subroutines in genetic programming," in *Advances in Genetic Programming II*. MIT Press, 1996.

[25] R. D. Howard and J. R. Koza, "Evolving modules in genetic programming by subtree encapsulation," in *Proc. of the 4th European Conf. on Genetic Programming (EuroGP 2001)*, 2001, pp. 160–175.

[26] Y. Kameya, J. Kumagai, and Y. Kurata, "Accelerating genetic programming by frequent subtree mining," in *Proc. of the 2008 Genetic and Evolutioary Computation Conf. (GECCO-08)*, 2008, pp. 1203–1210.

[27] G. Dong and J. Li, "Efficient mining of emerging patterns: discovering trends and differences," in *Proc. of the 5th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*, 1999, pp. 43–52.

[28] T. Uno, M. Kiyomi, and H. Arimura, "LCM ver. 2: efficient mining algorithms for frequent/closed/maximal itemsets," in *Proc. of the IEEE ICDM Workshop on Frequent Itemset Mining Implementation (FIMI-04)*, 2004.