Electronic Colloquium on Computational Complexity - Reports Series 1995 - available via:
ECCC
TR95-031
FTP: ftp.eccc.uni-trier.de:/pub/eccc/
WWW: http://www.eccc.uni-trier.de/eccc/
Email: ftpmail@ftp.eccc.uni-trier.de with subject 'help eccc'

Selecting the median

Dorit Dor * *Uri Zwick* *

June 25, 1995

Abstract

Improving a long standing result of Schönhage, Paterson and Pippenger we show that the *median* of a set containing n elements can always be found using at most $2.95n$ comparisons.

1 Introduction

The *selection problem* is defined as follows: Given a set X containing n distinct elements drawn from a totally ordered domain, and given a number $1 \leq i \leq n$, find the i-th *order statistic* of X, i.e., the element of X larger than exactly $i-1$ elements of X and smaller than the other $n-i$ elements of X. The *median* of X is the $\lceil n/2 \rceil$-th order statistic of X.

The selection problem is one of the most fundamental problems of computer science and it has been extensively studied. Selection is used as a building block in the solution of other fundamental problems such as sorting and finding convex hulls. It is somewhat surprising therefore that only in the early 70's, it was shown, by Blum, Floyd, Pratt, Rivest and Tarjan [BFP+73], that the selection problem can be solved in $O(n)$ time. As $\Omega(n)$ time is clearly needed to solve the selection problem, the work of Blum *et al.* completely solves the problem. Or does it?

A very natural setting for the selection problem is the *comparison model*. An algorithm in this model can access the input elements only by performing pairwise comparisons between them. The algorithm is only charged for these comparisons. All other operations are free. The comparison model is one of the few models in which *exact* complexity results may be obtained. What is then the exact comparison complexity of finding the median?

The comparison complexity of many comparison problems is exactly known. It is clear, for example, that exactly $n-1$ comparisons are needed, in the worst case, to find the maximum or minimum of n elements. Exactly $n + \lceil \log n \rceil - 2$ comparisons are needed to find the second largest (or second smallest) element (Schreier [Sch32], Kislitsyn [Kis64]). Exactly $\lceil 3n/2 \rceil - 2$ comparisons are needed to find both the maximum and the minimum of n elements (Pohl [Poh72]). Exactly $2n-1$ comparisons are needed to merge two sorted lists each of length n (Stockmeyer and Yao [SY80]). Finally, $n \log n + O(n)$ comparisons are needed to sort n elements (e.g., Ford and Johnson [FJ59]).

A relatively large gap, considering the fundamental nature of the problem, still remains however between the known lower and upper bounds on the exact complexity of finding the median. After presenting a basic scheme by which an $O(n)$ selection algorithm can be obtained, Blum *et al.* [BFP+73] try to optimise their algorithm and present a selection algorithm that performs at most $5.43n$ comparisons. They also obtain

*Department of Computer Science, School of Mathematical Sciences, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel Aviv University, Tel Aviv 69978, ISRAEL. E-mail addresses: {ddorit,zwick}@math.tau.ac.il.

the first non-trivial lower bound and show that $1.5n$ comparisons are required, in the worst case, to find the median. The result of Blum *et al.* is subsequently improved by Schönhage, Paterson and Pippenger [SPP76] who present a beautiful algorithm for the selection of the median, or any other element, using at most $3n + o(n)$ comparisons. In this work we improve the long standing result of Schönhage *et al.* and present a selection algorithm that uses at most $2.95n$ comparisons.

Bent and John [BJ85] (see also John [Joh88]), improving previous results of Kirkpatrick [Kir81], Munro and Poblete [MP82] and Fussenegger and Gabow [FG78], obtained a $(1 + H(\alpha))\cdot n - o(n)$ lower bound on the number of comparisons needed to select the αn-th element of a set of n elements, where $H(\alpha) = \alpha \log \frac{1}{\alpha} + (1-\alpha)\log \frac{1}{1-\alpha}$ is the binary entropy function (all logarithms in this paper are taken to base 2). We have shown recently [DZ95a] (using somewhat different methods from the ones used here) that the αn-th element can be selected using at most $(1 + \alpha \log \frac{1}{\alpha} + O(\alpha \log\log \frac{1}{\alpha}))\cdot n$ comparisons. This for small values of α is almost optimal. The bound of Bent and John gives in particular a $2n - o(n)$ lower bound on the number of comparisons needed to find the median.

Our work slightly narrows the gap between the best known lower and upper bounds on the comparison complexity of the median problem. Though our improvement is quite modest, many new ideas were required to obtain it. These new ideas shed some more light on the intricacy of the median finding problem.

Algorithms for selecting the i-th element for small values of i were obtained by Hadian and Sobel [HS69], Hyafil [Hya76], Yap [Yap76], Ramanan and Hyafil [RH84]. See also Aigner [Aig82] and Eusterbrock [Eus93].

All the results mentioned so far deal with the number of comparisons needed in the *worst case*. Floyd and Rivest [FR75] showed that the i-th element can be found using an *expected* number of $n + i + o(n)$ comparisons. Cunto and Munro [CM89] had shown that the bound of Floyd and Rivest is tight.

The central idea used by Schönhage *et al.* in their $3n + o(n)$ median algorithm is the idea of *factories*. Schönhage *et al.* use factories for the mass production of certain partial orders at a much reduced cost. To obtain our results we extend the notion of factories. We introduce *green* factories and perform an *amortised* analysis of their production costs. We obtain improved green factories using which we can improve the $3n + o(n)$ result of Schönhage, Paterson and Pippenger.

The performance of a green factory is mainly characterised by two parameters u_0 and u_1 (the *upper* and *lower* element costs). Using a green factory with parameters u_0 and u_1 we obtain an algorithm for the selection of the αn-th element using at most $(u_0\alpha + u_1(1-\alpha))\cdot n + o(n)$ comparisons. To select the median, we use a factory with $u_0, u_1 \approx 2.95$. Actually, there is a tradeoff between the lower and upper costs of a factory. For every $0 < \alpha \leq 1/2$ we may choose a factory that minimises $u_0\alpha + u_1(1-\alpha)$. We can select the $n/4$-th element, for example, using at most $2.69n$ comparisons, by using a factory with $u_0 \approx 4$ and $u_1 \approx 2.25$. In this paper, we concentrate on factories for median selection. It is easy to verify that the algorithm described here, as the median finding algorithms of both Blum *et al.* and Schönhage *et al.*, can be implemented in linear time in the RAM model.

The best green factories that we have explicitly constructed have lower and upper element costs $u_0, u_1 \simeq 2.942$, yielding a median selection algorithm that uses at most $2.942n$ comparisons. These factories are extremely complicated. They employ in particular sixteen different sub-factories (sub-factories are introduced in Section 5). A full description of these factories is too long for a journal paper. A full description of them can be found in [Dor95]. We describe instead a simpler construction, though still quite complicated, that uses only four different sub-factories and has $u_0, u_1 \simeq 2.956$. This simpler construction depicts all the ideas used in the construction of the more complicated factories. After describing these simpler factories in full, we give a partial description of the more complicated factories.

A preliminary version of this paper appeared in [DZ95b]. There we sketched the construction of a factory that uses only two sub-factories and achieves $u_0, u_1 \simeq 2.968$. These are about the simplest factories using which the $3n$ median algorithm of Schönhage *et al.* can be improved. They do not demonstrate however

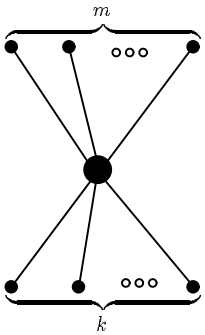

Figure 1: The partial order S_k^m.

all the ideas required to obtain the $u_0, u_1 \simeq 2.956$ and $u_0, u_1 \simeq 2.942$ factories.

We believe that further small improvements can be obtained by building more complicated factories that use even more sub-factories. It seems however that new ideas will be needed to obtain a more substantial improvement (see in particular the comments at the end of Section 9).

In the next section we describe in more detail the concept of factory production and introduce our notion of a *green* factory. We also state the properties of the improved factories that we obtain. In Section 3 we explain the way in which green factories are used to obtain efficient selection algorithms. The selection algorithm that we describe is a generalisation of the median algorithm of Schönhage *et al.* [SPP76] and is similar to the selection algorithm that we describe in [DZ95a]. The subsequent sections are then devoted to the construction of our improved green factories. We end with some concluding remarks and open problems.

2 Factory production

Denote by S_k^m a partial order composed of a *centre* element, m elements larger than the centre and k elements smaller than the centre (see Fig. 1). An S_k^m is sometimes referred to as a *spider*. Schönhage *et al.* [SPP76] show that producing l disjoint copies of S_k^m usually requires fewer comparisons than l times the number of comparisons required to produce a single S_k^m. The best way, prior to this work, of producing a single S_k^k, for example, requires about $6k$ comparisons (find the median of $2k+1$ elements using the $3n + o(n)$ median algorithm). The cost per copy can be cut by almost a half if the S_k^k's are mass produced using factories.

A *factory* for a partial order P is a comparison algorithm with continual input and output streams. The input stream of a simple factory consists of single elements. When enough elements are fed into the factory, a new disjoint copy of P is produced. A factory is characterised by the following quantities: the *initial cost* I, which is the number of comparisons needed to initialise the factory; the *unit cost* U, which is the number of comparisons needed to generate each copy of P; and finally the *production residue* R, which is the maximal number of elements that can remain in the factory when lack of inputs stops production. For every $l \geq 0$, the cost of generating l disjoint copies of P is at most $I + l \cdot U$. Schönhage *et al.* [SPP76] construct factories with the following characteristics:

Theorem 2.1 *There is a factory F_k for S_k^k with initial cost I_k, unit cost U_k and production residue R_k satisfying: $U_k \sim 3.5k$, $I_k = O(k^2)$, $R_k = O(k^2)$.*

The notation $U_k \sim 3.5k$ here means that $U_k = 3.5k + o(k)$. Schönhage *et al.* also show that if there exist factories F_k, for S_k^k's, satisfying $U_k \sim Ak$, for some $A > 0$, and $I_k, R_k = O(k^2)$, then the median of n

elements can be found using at most $An + o(n)$ comparisons. The above theorem immediately implies therefore the existence of a $3.5n + o(n)$ median algorithm.

The way factories are used by selection algorithms is described in the next section. For now, we just mention that most S_k^m's generated by a factory employed by a selection algorithm are eventually broken, with either their upper elements eliminated and their lower elements returned to the factory or vice versa. While constructing an S_k^m, a factory may have compared elements that turned out to be on the same side of the centre. If such elements are ever returned to the factory, the known relations among them may save the factory some of the comparisons it has to perform. To capture this, we extend the definition of factories and define *green* factories (factories that support the recycling of known relations). This extension is implicit in the work of Schönhage *et al.* [SPP76]. Making this notion explicit simplifies the analysis of our factories. The $3n + o(n)$ median algorithm of Schönhage *et al.* is in fact obtained by replacing the factory F_k of Theorem 2.1 by a simple green factory.

A green factory for S_k^m's is mainly characterised by the following two quantities: the *lower element cost* u_0 and the *upper element cost* u_1. Using these quantities, the *amortised* production costs of the factory can be calculated as follows: The amortised production cost of an S_k^m whose upper m elements are eventually returned (together) to the factory is $k \cdot u_0$. The amortised production cost of an S_k^m whose lower k elements are eventually returned (together) to the factory is $m \cdot u_1$. The amortised production cost of an S_k^m such that none of its elements is returned to the factory is $k \cdot u_0 + m \cdot u_1$. Note that in this accounting scheme we attribute all the production cost to elements that are *not* returned to the factory. The initial cost I and the production residue R of a green factory are defined as before. A somewhat different definition of green factory was given by us in [DZ95a]. The definition given here uses amortised costs per *element* where as our previous definition used amortised costs per partial order. A green factory does not know in advance whether the lower or upper part of a generated S_k^m will be recycled. This is set by an adversary. Though not stated explicitly, the following result is implicit in [SPP76].

Theorem 2.2 *There is a green factory G_k for S_k^k with lower and upper element costs $u_0, u_1 \sim 3$, initial cost $I_k = O(k^2)$ and production residue $R_k = O(k^2)$.*

The notation $u_0, u_1 \sim 3$ here means that $u_0, u_1 = 3 + o(1)$ where the $o(1)$ is with respect to k.

We shall see in the next section that a green factory for S_k^k with lower and upper element costs u_0 and u_1 yields a $(u_0 + u_1)/2 \cdot n + o(1)$ median algorithm. To improve the algorithm of Schönhage *et al.* it is enough therefore to construct an S_k^k factory with $(u_0 + u_1)/2 < 3$. Unfortunately, we are not able to construct such a factory.

However, we are able to reduce the upper and lower element costs if we allow variation among the partial orders generated by the factory. Let $\tilde{\mathcal{S}}_k^k = \{S_{k'}^{k''} : k \leq k' \leq 2k\ ,\ k \leq k'' \leq 2k\}$. We construct improved green factories that generate partial orders that are members of $\tilde{\mathcal{S}}_k^k$. These factories can be easily incorporated into the selection algorithm described in the next section. To obtain our $2.95n$ median algorithm we use green $\tilde{\mathcal{S}}_k^k$ factories $\mathcal{G}_k$ with the following characteristics:

Theorem 2.3 *There is a green $\tilde{\mathcal{S}}_k^k$ factory $\mathcal{G}_k$ with $u_0, u_1 \simeq 2.942$, $I_k = O(k^2)$, $R_k = O(k^2)$.*

An outline of the ideas used to construct the factories $\mathcal{G}_k$ is given in Section 5. The full details are then given in Sections 6 to 9.

3 Selection algorithms

In this section we describe our selection algorithm. This algorithm uses an $\tilde{\mathcal{S}}_k^k$ factory. The complexity of the algorithm is completely determined by the characteristics of the factory used. This algorithm is a

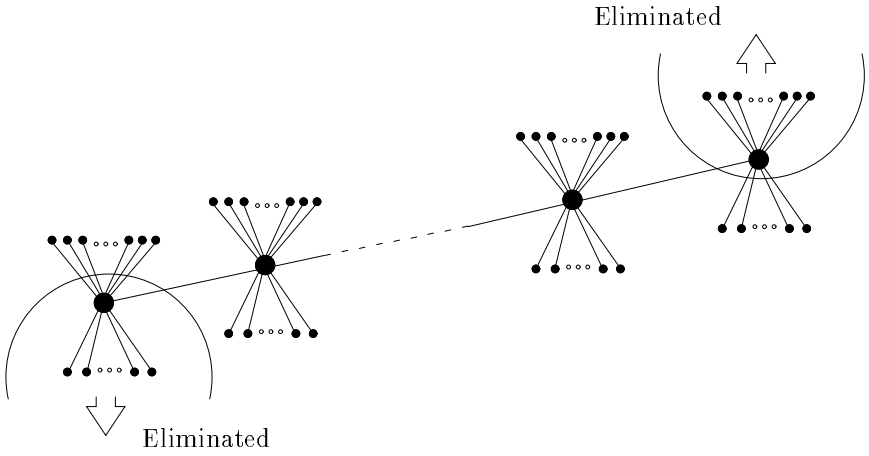

Figure 2: The ordered list of $\tilde{\mathcal{S}}_k^k$'s.

generalisation of the median algorithm of Schönhage *et al.* and a slight variation of the selection algorithm that we describe in [DZ95a].

Theorem 3.1 *Let $0 < \alpha \leq 1/2$. Let $\mathcal{F}_k$ be an $\tilde{\mathcal{S}}_k^k$ factory with lower element cost u_0, upper element cost u_1, initial cost $I_k = O(k^2)$ and production residue $R_k = O(k^2)$. Then, the αn-th smallest element, among n elements, can be selected using at most $(\alpha \cdot u_0 + (1-\alpha) \cdot u_1) \cdot n + o(n)$ comparisons.*

Proof: We refer to the αn-th smallest element among the n input elements as the percentile element. The algorithm uses the factory $\mathcal{F}_k$ where $k = \lfloor n^{1/4} \rfloor$. The n input elements are fed into this factory, as singletons, and the production of partial orders $S \in \tilde{\mathcal{S}}_k^k$ commences. The centres of the generated S's are inserted, using binary insertion, into an ordered list L, as shown in Fig. 2. When the list L is long enough we either know, as we shall soon show, that the centre of the upper (i.e., last) S in L and the elements above it are too large to be the percentile element, or that the centre of the lower (i.e., first) S and the elements below it are too small to be the percentile element. Elements too large or too small to be the percentile element are eliminated. The lower elements of the upper S, and the upper elements of the lower S are returned to the factory for recycling.

Let t be the current length of the list L and let r be the number of elements currently in the factory. The number of elements that have not yet been eliminated is therefore $N = \Theta(k) \cdot t + r$. Let i be the rank of the percentile element among the non-eliminated elements. Initially $N = n$ and $i = \lceil \alpha n \rceil$.

The number of elements in the list known to be smaller or equal to the centre of the upper S of the list is $N_0 = \Theta(k) \cdot t$. The number of elements known to be greater or equal to the centre of the lowest S of the list is $N_1 = \Theta(k) \cdot t$. Note that $N_0 + N_1 = N + t - r$ as the centres of all the S's in the list satisfy both these criteria, the r elements are currently in the factory satisfy neither, and all the other non-eliminated elements satisfy exactly one of these criteria.

The algorithm consists of the following interconnected processes:

(i) Whenever sufficiently many elements are supplied to the factory $\mathcal{F}_k$, a new partial order $S \in \tilde{\mathcal{S}}_k^k$ is produced and its centre is inserted into the list L using binary insertion.

(ii) Whenever $N_0 > i$, the centre of the upper partial order $S \in \tilde{\mathcal{S}}_k^k$ in the list and the elements above it are eliminated, as they are too big to be the percentile element. The lower elements of S are recycled.

(iii) Whenever $N_1 > N - i + 1$, the centre of the lowest partial order $S \in \tilde{\mathcal{S}}_k^k$ in the list and the elements below it are eliminated, as they are too small to be the percentile element. The upper elements of S are recycled. The value of i is updated accordingly, i.e., i is decremented by the number of elements in the lower part of S (including the centre).

If (ii) and (iii) are not applicable then $N_0 \leq i$ and $N_1 \leq N - i + 1$. Thus $N + t - r = N_0 + N_1 \leq N + 1$ and $t - 1 \leq r$. If (i) is not applicable then by the factory definition we have $r \leq R_k$. When no one of (i),(ii) and (iii) can be applied we get that $t - 1 \leq r \leq R_k = O(k^2)$. At this stage $N = O(k^3)$, which is $O(n^{3/4})$, and the i-th element among the surviving elements is found using any linear selection algorithm.

We now analyze the comparison complexity of the algorithm. Whenever (ii) is performed, the upper partial order $S \in \tilde{\mathcal{S}}_k^k$ of the list is broken. Its centre and upper elements are eliminated and its lower elements are returned to the factory. The amortised production cost of the partial order S is at most u_1 comparisons per each element above the centre.

Whenever (iii) is performed, the lowest partial order $S \in \tilde{\mathcal{S}}_k^k$ of the list is broken. Its centre and lower elements are eliminated and its upper elements are returned to the factory. The amortised production cost of the partial order S is at most u_0 comparisons per each element below the centre.

The algorithm can eliminate at most $(1-\alpha)n$ elements larger than the percentile element and at most αn elements smaller than the percentile element. The total production cost of all partial orders $S \in \tilde{\mathcal{S}}_k^k$ that are eventually broken is therefore at most $(\alpha u_0 + (1-\alpha)u_1)\cdot n + o(n)$. At most $O(k^2)$ generated partial orders $S \in \tilde{\mathcal{S}}_k^k$ are not broken. Their total production cost is $O(k^3)$. The initial production cost is $O(k^2)$. The total number of comparisons performed by the factory is therefore $(\alpha u_0 + (1-\alpha)u_1)\cdot n + o(n)$.

Let t^* be the final length of the list L (when none of (i),(ii) and (iii) is applicable). The total number of partial orders generated by $\mathcal{F}_k$ is at most $n/k + t^*$, as at least k elements are eliminated whenever a partial order is removed from L. The total cost of the binary insertions into the list L is at most $O((n/k + t^*)\cdot \log n) = O((n/k + k^2)\log n)$ which is $o(n)$. The total number of comparisons performed by the algorithm is therefore at most $(\alpha u_0 + (1-\alpha)u_1)\cdot n + o(n)$, as required. □

Using the factories of Theorem 2.3, we obtain our main result:

Theorem 3.2 *Any element, among n elements, can be selected using at most $2.942n + o(n)$ comparisons.*

4 Basic principles of factory design

In this section we give some of the basic principles used to construct efficient factories. The section is divided into three subsections. In the first subsection we remind the reader what *hyperpairs* are and what their *pruning cost* is. In the second subsection we describe the notion of *grafting*. In the third subsection we sketch the construction of the S_k^k factories of Schönhage *et al.* [SPP76]. These factories are briefly described to illustrate the basic design principles. All the results of this section, except for Theorems 4.5 and 4.6, which are new, are essentially taken from [SPP76]. Some of the proofs are therefore omitted.

Before going into details, we describe a clever accounting principle introduced by Schönhage *et al.* to simplify the complexity analysis of factories. The information we care to remember on the elements that pass through the factory can always be described using a Hasse diagram. Each comparison made by the algorithm adds an edge to the diagram and possibly deletes some edges that become redundant. At some stages we may decide to 'forget' the result of some comparisons and the edges that correspond to them are removed from the diagram. Schönhage *et al.* noticed that instead of counting the number of comparisons made, we can count the number of edges cut! To this we should add the number of edges in the eliminated parts of the partial orders as well as the edges that remain in the factory when the production stops. The

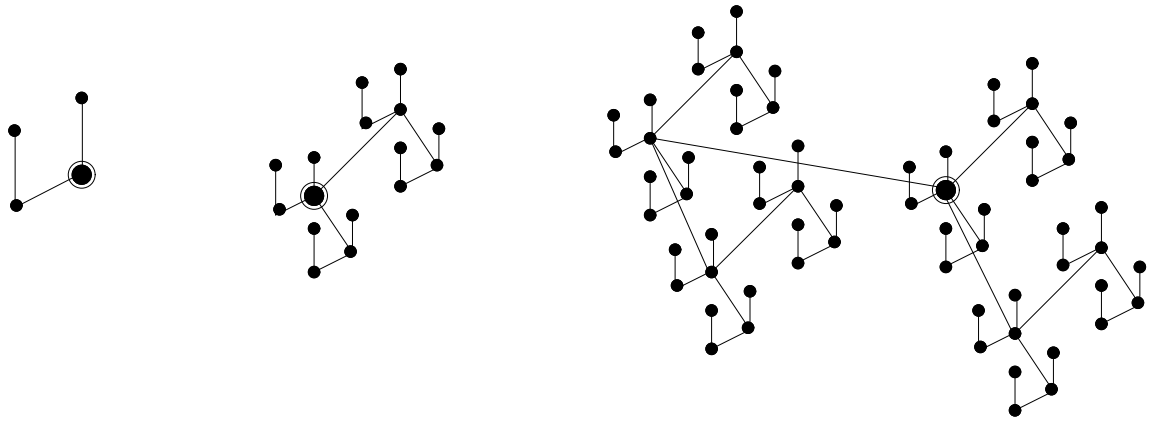

Figure 3: Some small hyperpairs ($H_2 = P_{01}$, $H_4 = P_{0110}$ and $H_6 = P_{011010}$).

second number, in our factories, is at most a constant times the production residue of the factory and it can be attributed to the initial cost.

4.1 Hyperpairs

A factory usually starts the production of a partial order from $\tilde{\mathcal{S}}_k^k$ by producing a large partial order, *a hyperpair*, that contains a partial order from $\tilde{\mathcal{S}}_k^k$.

Definition 4.1 *An hyperpair P_w, where w is a binary string, is a finite partial order with a distinguished element, the* centre, *defined recursively by (i) P_λ is a single element (λ here stands for the empty string). (ii) P_{w1} is obtained from two disjoint P_w's by comparing their centres and taking the larger as the new centre. P_{w0} is obtained in the same way but taking the smaller of the two centres as the new centre.*

The Hasse diagrams of some small hyperpairs are shown in Fig. 3 (the meaning of the notation H_r will become clear later). Some basic properties of hyperpairs are given in the following Lemma.

Lemma 4.2 *Let c be the centre of a hyperpair P_w. Let w_i be the prefix of w of length i. Let h_0 be the number of 0's in w and h_1 be the number of 1's in w. Then:*

1. *The centre c together with the elements greater than it form a $P_{0^{h_0}}$ with centre c. The elements greater than c form a disjoint set of hyperpairs P_λ, P_0, ... , $P_{0^{h_0-1}}$. The centre c together with the elements smaller than it form a $P_{1^{h_1}}$ with centre c. The elements smaller than c form a disjoint set of hyperpairs P_λ, P_1, ... , $P_{1^{h_1-1}}$.*
2. *The hyperpair P_w can be parsed into its centre c and a disjoint set $\{P_{w_i} \mid 0 \leq i < |w|\}$ of smaller hyperpairs. Moreover, the centre of P_{w_i} is above c if w_{i+1} ends with 0, and below c if w_{i+1} ends with 1.*

The Lemma can be easily proved by induction. Note, in particular, that if $m < 2^{h_0}$ and $k < 2^{h_1}$ then P_w contains an S_k^m. No edges are cut during the construction of hyperpairs. But, before outputting an S_k^m contained in a hyperpair, all the edges connecting the elements of this S_k^m with elements not contained in this S_k^m have to be cut. This rather costly operation is referred to as *pruning*.

The *downward pruning* of a hyperpair P_w is the operation of removing from the hyperpair all the elements that are not known to be smaller or equal to its centre c. The *downward pruning cost* $PR_0(w)$ of a hyperpair P_w is the cost of this operation. This cost is equal to the number of edges of the hyperpair that connect the centre of P_w and the elements below it to the other elements of P_w. The *upward pruning* and the *upward pruning cost* $PR_1(w)$ of a hyperpair P_w are defined analogously. Let w be a binary string and let h_0 and h_1 be the number of 0's and 1's in it. The pruning costs can be computed using the following recursive relations.

Lemma 4.3 (i) $PR_0(\lambda) = 0$, $PR_0(w0) = PR_0(w) + 1$, $PR_0(w1) = 2PR_0(w)$.
(ii) $PR_1(\lambda) = 0$, $PR_1(w0) = 2PR_1(w)$, $PR_1(w1) = PR_1(w) + 1$.

Proof: A P_{w0} is composed of two P_w's and an edge connecting their two centres. If c_1 and c_2 are the centres of these P_w's and $c_1 < c_2$ then c_1 is the centre of the hyperpair P_{w0}. To downward prune the hyperpair P_{w0}, all we have to do is to downward prune the hyperpair P_w whose centre is c_1 and the cut the edge between c_1 and c_2. The cost of this operation is therefore $PR_0(w)+1$. The other cases are proved similarly. □

We also define the *amortised*, per element, pruning costs of a hyperpair P_w. Let h_0 and h_1 be the number of 0's and 1's in w and let $h = h_0 + h_1$. We define $pr_0(w) = PR_0(w)/2^{h_1}$ and $pr_1(w) = PR_1(w)/2^{h_0}$ to be the *lower element pruning cost* and the *upper element pruning cost* of w. An immediate consequence of Lemma 4.3 is the following Lemma which is easily proved by induction.

Lemma 4.4 *Let $w = \gamma_1\gamma_2\ldots\gamma_h$ be a binary string, let w_i be the prefix of w of length i, and let $h_0(w_i)$ and $h_1(w_i)$ be the number of 0's and 1's, respectively, in w_i. Then,*

$$pr_0(w) = \sum_{i|\gamma_i=0} 2^{-h_1(w_i)} \quad , \quad pr_1(w) = \sum_{i|\gamma_i=1} 2^{-h_0(w_i)} \quad .$$

Note, in particular, that if w_i is a prefix of w then $pr_0(w_i) \leq pr_0(w)$ and $pr_1(w_i) \leq pr_1(w)$.

Usually, especially if grafting processes are also used, we do not want to prune all the elements above or below the centre c of a hyperpair P_w. We next estimate the cost of pruning k elements below or above the centre of a hyperpair P_w, i.e., the cost of pruning the hyperpair P_w so that all that remains of it is its centre and k elements either above or below the centre. We show that the cost of pruning k elements below the centre of P_w is at most $k{\cdot}pr_0(w)+h$ while the cost of pruning k elements above the centre of P_w is at most $k{\cdot}pr_1(w)+h$, where h is the length of w. Note that h is also the number of edges connected to the centre c of the hyperpair P_w. The h terms in the above estimates will usually be negligible compared to the other terms.

By Lemma 4.2, a hyperpair P_w with centre c can be parsed into h_0 hyperpairs $\{P_i^1 \mid 0 \leq i \leq h_0 - 1\}$ whose centres are above c and h_1 hyperpairs $\{P_i^0 \mid 0 \leq i \leq h_1 - 1\}$ whose centres are below c. It is easy to check that the number of elements in P_i^1 which are above the centre of this P_i^1, and the number of elements in P_i^0 which are below the centre of this P_i^0, are both 2^i (this follows from the fact that the string corresponding to the hyperpair P_i^1 contains exactly i 0's and that the string corresponding to the hyperpair P_i^0 contains exactly i 1's). To upward prune $k = k_0 2^0 + k_1 2^1 + \cdots + k_{h-1}2^{h-1}$ elements from P_w, we upward prune all P_i^1's for which $k_i = 1$ and cut the edges connecting c with the centres of all the other hyperpairs. As the string corresponding to P_i^1 is a prefix of w, the cost of upward pruning P_i^1 is at most $2^i{\cdot}pr_1(w)$. The cost of cutting the other edges is at most h. Thus, the cost of pruning k elements above the centre c is at most $\sum_{i|k_i=1} 2^i{\cdot}pr_1(w)+h = k{\cdot}pr_1(w)+h$. All the 'wastes' of this process are hyperpairs whose strings are prefixes of w and they can be used therefore for the construction of the next P_w. The cost of pruning k elements below the centre is estimated in the same way.

To produce partial orders from $\tilde{\mathcal{S}}_k^k$ for larger and larger values of k, we have to construct larger and larger hyperpairs. When we design a family $\{\mathcal{F}_k\}_{k=1}^{\infty}$ of factories, we usually choose a (semi) infinite binary string $\mathcal{W}$ and in each member $\mathcal{F}_k$ of this family we construct a hyperpair whose string is a long enough prefix of $\mathcal{W}$. Let w_i be the finite prefix of $\mathcal{W}$ of length i. The lower and upper element pruning costs of an infinite string $\mathcal{W}$ are defined as the limits $pr_0(\mathcal{W}) = \lim_{i\to\infty} pr_0(w_i)$ and $pr_1(\mathcal{W}) = \lim_{i\to\infty} pr_1(w_i)$. We will see in a minute that these limits do exist (they may be $+\infty$).

Let $\mathcal{W} = \gamma_1\gamma_2\ldots$ be an infinite binary string. As before, let $w_i = \gamma_1\ldots\gamma_i$ be the prefix of $\mathcal{W}$ of length i, and let $h_0(w_i)$ and $h_1(w_i)$ be the number of 0's and 1's in w_i, respectively. It is easy to check that Lemma 4.4 holds also for infinite strings.

Schönhage *et al.* base their factories on the infinite string $\mathcal{W} = 01(10)^{\omega}$ for which, as can be easily verified, $pr_0(\mathcal{W}) = pr_1(\mathcal{W}) = 1.5$. (Here and in what follows, we let x^{ω} denote the infinite string obtained by concatenating an infinite number of copies of the string x, and X^{ω} denote the set of infinite strings obtained by concatenating an infinite number of strings from the set X.) In our factories, we also need hyperpairs with cheaper lower element pruning cost and, alas, more expensive upper element pruning cost, or vice versa. For an infinite binary string $\mathcal{W}$, we let $pr(\mathcal{W}) = pr_0(\mathcal{W}) + pr_1(\mathcal{W})$. The next theorem shows that for every string $\mathcal{W}$ we have $pr(\mathcal{W}) \geq 3$. Theorem 4.6 then shows that for any real number $1 \leq a \leq 2$, there exists an infinite string $\mathcal{W}$ for which $pr_0(\mathcal{W}) = a$ and $pr_1(\mathcal{W}) = 3 - a$.

Theorem 4.5 *For any $\mathcal{W} \in \{0,1\}^{\omega}$ we have $pr(\mathcal{W}) \geq 3$ with equality holding if and only if $\mathcal{W} \in \{01,10\}^{\omega}$.*

Proof: Let $\mathcal{W}$ be an infinite string. As before, we let w_i stand for the prefix of $\mathcal{W}$ of length i. If $\mathcal{W}$ contains only a finite number of 0's or a finite number of 1's, then $pr(\mathcal{W}) = +\infty$. Assume therefore that $\mathcal{W}$ contains an infinite number of 0's and an infinite number of 1's. Using Lemma 4.4 we get that

$$pr(\mathcal{W}) = \sum_{i|\gamma_i=0} 2^{-h_1(w_i)} + \sum_{i|\gamma_i=1} 2^{-h_0(w_i)} = \sum_{i=1}^{\infty} 2^{-h_1(w_i)} + \sum_{i=1}^{\infty} 2^{-h_0(w_i)} - 2 \ ,$$

as $\sum_{i|\gamma_i=1} 2^{-h_1(w_i)} = \sum_{i|\gamma_i=0} 2^{-h_0(w_i)} = 1$. Let x_i be the index of the i-th 0 in $\mathcal{W}$ and let y_i be the index of the i-th 1 in $\mathcal{W}$. Also let $x_0 = y_0 = 1$. It is easy to check that

$$\sum_{i=1}^{\infty} 2^{-h_0(w_i)} = \sum_{i=0}^{\infty} 2^{-i}(x_{i+1} - x_i) = \sum_{i=1}^{\infty} 2^{-i} x_i - 1 \ .$$

A similar relation holds of course for $\sum_{i=1}^{\infty} 2^{-h_1(w_i)}$. As a consequence we get that

$$pr(\mathcal{W}) = \sum_{i=1}^{\infty} 2^{-i} x_i + \sum_{i=1}^{\infty} 2^{-i} y_i - 4 \ .$$

This expression is clearly minimised when for every $1 \leq i < j$ we have $x_i, y_i \leq x_j, y_j$. As all the elements in the sequences $\{x_i\}$ and $\{y_i\}$ are integral and distinct, we get that the minimum is attained when for every $i \geq 1$, either $x_i = 2i - 1$ and $y_i = 2i$ or vice versa. This corresponds to strings from $\{01,10\}^{\omega}$. It is easy to check that for a string $\mathcal{W} \in \{01,10\}^{\omega}$ we have $pr(\mathcal{W}) = 3$. □

Theorem 4.6 *For any real number $1 \leq a \leq 2$ there exists a binary sequence $\mathcal{W} \in \{01,10\}^{\omega}$ for which $pr_0(\mathcal{W}) = a$ and $pr_1(\mathcal{W}) = 3 - a$.*

Proof: Let $0.\alpha_1\alpha_2\ldots$ be the binary representation of $a - 1$. Let $\mathcal{W} = \overline{\alpha}_1\alpha_1\overline{\alpha}_2\alpha_2\ldots$, where $\overline{\alpha}$ is the complement of the bit α. Using Lemma 4.4, we get that

$$pr_0(\mathcal{W}) = \sum_{i=1}^{\infty} 2^{-(i-\alpha_i)} = \sum_{i=1}^{\infty} 2^{-i} + \sum_{i|\alpha_i=1} 2^{-i} = 1 + (a-1) = a \ .$$

As $\mathcal{W} \in \{01, 10\}^\omega$, we get using the previous theorem that $pr_1(\mathcal{W}) = 3 - a$. □

Note, as an example, that if $a = 1.5$ then $a - 1$ is either $0.1000\ldots$ or $0.0111\ldots$ and both $\mathcal{W} = 01(10)^\omega$ and $\mathcal{W} = 10(01)^\omega$ satisfy $pr_0(\mathcal{W}) = pr_1(\mathcal{W}) = 1.5$.

We are already in a position to describe a simple but complete S_k^k factory. Select a string $\mathcal{W}$. Construct a hyperpair P_w that contains the partial order S_k^k, where w is a long enough prefix of $\mathcal{W}$. Prune k elements above and k elements below the centre of this P_w. These $2k+1$ elements form a copy of S_k^k. By Lemma 4.2(ii), the remaining elements of P_w form a disjoint collection of partial orders each of the form P_{w_i}, where w_i is some prefix of w. These partial orders are used to construct a new copy of P_w that will be used to construct the next S_k^k. Before we output an S_k^k, we cut the $2k$ edges it contains. When some part of an S_k^k generated by the factory is recycled, the elements returned to the factory (as singletons) are used again for the construction of hyperpairs. It is easy to check that the lower and upper element costs of this simple factory are both $u_0, u_1 \sim pr_0(\mathcal{W}) + pr_1(\mathcal{W}) + 2 = pr(\mathcal{W}) + 2$. For any $\mathcal{W} \in \{01, 10\}^\omega$ we get that the lower and upper element costs are $u_0, u_1 \sim 5$.

The above factory can be turned into a green factory by noticing that the elements can be returned to the factory as pairs and not as singletons. The edges of a generated S_k^k are not cut when this S_k^k leaves the factory. Instead, we wait until the elements on one of the sides of this S_k^k are eliminated by the selection algorithm. We then cut the k edges connecting the eliminated elements to the centre. By cutting about $k/2$ edges we can break the k non-eliminated elements into about $k/2$ pairs which we then return to the S_k^k factory. These pairs can be used for the construction of the next large hyperpair. The lower and upper element costs of this simple green factory are $u_0, u_1 \sim pr(\mathcal{W}) + 1.5$. For any $\mathcal{W} \in \{01, 10\}^\omega$ we get $u_0, u_1 \sim 4.5$. A slightly more careful consideration shows that the elements of one of the sides can actually be recycled as quartets (P_{00}'s or P_{11}'s) thus cutting only $k/4$ edges. This gives us $u_0 \sim 4.25$ and $u_1 \sim 4.5$ or vice versa.

4.2 Grafting

The costs of the simple factories described above can be significantly improved using *grafting*. We can cheaply find elements that are smaller than the centre, or elements that are larger than the centre (but not both usually). The process of finding such elements is called *grafting*. Pruning is then used to obtain elements on the opposite side.

We demonstrate this notion using a simple example, the grafting of singletons. Take an element x, not contained in the hyperpair, and compare it to the centre c of the hyperpair. Continue is this way, comparing new elements to the centre, until either k elements above the centre, or k elements below the centre are found. Note that no edges are cut in this process. All the grafted elements are put in the output partial order. The pruning process is then used to complete the partial order into an S_k^k. Adding this process to our simple factory for S_k^k, the upper and lower element costs are reduced to: $u_1, u_0 \sim \max\{pr_0(\mathcal{W}), pr_1(\mathcal{W})\} + 2$ (note that now we have to prune elements from at most one side). Thus $u_0, u_1 \sim 3.5$ if we take $\mathcal{W} = 01(10)^\omega$ or $\mathcal{W} = 10(01)^\omega$. This supplies a proof to Theorem 2.1. Note that the obtained factory is a degenerate green factory as no relations are recycled. At least one side of each generated S_k^k is composed of singletons, and if this side is recycled, no comparisons can be reused.

4.3 The factories of Schönhage, Paterson and Pippenger

We now sketch the operation of the green factories G_k obtained by Schönhage *et al.* [SPP76]. These factories improve upon the simple factories described above by grafting and recycling pairs. They are described here using a new terminology that we also use in the next sections to describe our improved factories.

The factories of Schönhage *et al.* use two simple pair grafting processes to which we refer as the P_0 and P_1 grafting processes. The two processes are mirror images of each other.

We start with the description of the P_0 grafting process. Let $x < y$ be a pair. Compare x with c, the centre of the hyperpair. If $c < x$ then stop. Otherwise, compare y with c. The three possible outcomes of this process are (i) $c < x < y$, (ii) $x < y < c$ and (iii) $x < c < y$. These three outcomes are shown on the left of Fig. 4 and are denoted, respectively, by X_0^2, X_2^0 and X_1^1. Note that in the second case, the result $x < c$ of the first comparison becomes redundant and the edge corresponding to it, shown dashed in the figure, is cut. Similarly, in the third case, the relation $x < y$ becomes redundant and the corresponding edge is cut.

The P_1 grafting process is, as mentioned, the mirror image of the P_0 grafting process. Let $x < y$ be a pair. Compare y with c and if $c < y$ also compare x with c. The three possible outcomes of this process, denoted by Y_2^0, Y_0^2 and Y_1^1 are shown on the right of Fig. 4. Note that the outcome of Y_1^1 of the P_1 grafting process is identical to the outcome X_1^1 of the P_0 grafting process. The outcomes Y_2^0 and X_2^0, and Y_0^2 and X_0^2 have the same forms but different costs are associated with them. No edges are cut while producing X_0^2 and Y_2^0 while a single edge is cut while producing X_2^0 and Y_0^2.

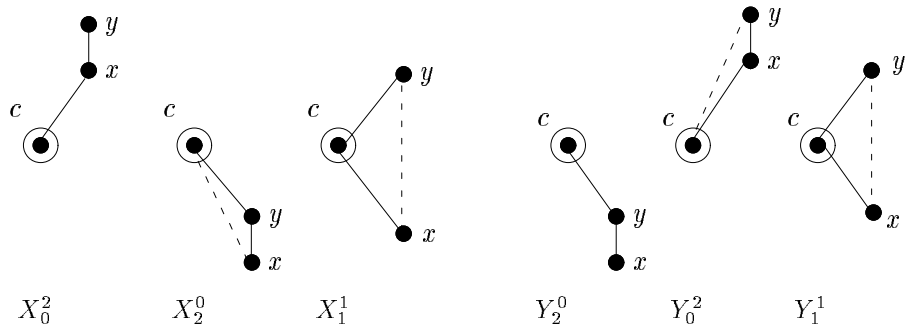

Figure 4: Possible outcomes of the simple pair grafting processes

The costs associated with each one of the outcomes obtained by these processes are summarised in Table 1. If X is an outcome of a grafting process then $gen(X)$ is the number of edges cut during the generation of X, $rec_0(X)$ is the number of edges cut when the lower part of the spider to which the outcome X belongs is recycled while the upper part is eliminated, and $rec_1(X)$ is defined analogously as the number of edges cut when the upper part of X is recycled and the lower part eliminated.

Outcome	Above	Below	gen	rec_0	rec_1
X_0^2	P_0	—	0	2	1
X_2^0	—	P_0	1	1	2
X_1^1	P_λ	P_λ	1	2	2

Outcome	Above	Below	gen	rec_0	rec_1
Y_2^0	—	P_0	0	1	2
Y_0^2	P_0	—	1	2	1
Y_1^1	P_λ	P_λ	1	2	2

Table 1: The costs of the Possible outcomes of the simple pair grafting processes.

The factory G_k starts by producing hyperpairs corresponding to prefixes of the string $\mathcal{W} = 01(10)^\omega$ (the string $\mathcal{W} = 10(01)^\omega$ could also be used). Let w_i be the prefix of $\mathcal{W}$ of length i and let $H_i = P_{w_i}$. Some small H_i's are shown in Fig. 3. By Lemma 4.2, an H_{2r}, where $r = \lceil \log(k+1) \rceil$ contains an S_k^k.

After constructing an H_{2r}, the factory initiates the pair grafting processes. The algorithm keeps two counters n_0 and n_1 of the number of grafted elements already obtained above and below the centre c of the hyperpair H_{2r}. If $n_0 > n_1$ then the P_1 grafting process is applied. If $n_0 \leq n_1$ then the P_0 grafting process is applied. The grafting continues until either $n_0 \geq k$ or $n_1 \geq k$. All the elements on at least one side of the spider are thus obtained using grafting. Missing elements on the opposite side are then obtained using pruning. The operation of the algorithm is summarised in Fig. 5.

Construct a hyperpair H_{2r}, where $r = \lceil \log(k+1) \rceil$.
Inilialize $n_0, n_1 \leftarrow 0$.

while $n_0 < k$ **and** $n_1 < k$ **do**
{
  if $n_0 > n_1$ **then**
  {
    Apply the P_1 grafting process on a pair $x < y$;
    if $x < y < c$ **then** $n_0 \leftarrow n_0 + 2$; (* Y_2^0 *)
    if $c < x < y$ **then** $n_1 \leftarrow n_1 + 2$; (* Y_0^2 *)
    if $x < c < y$ **then** $n_0 \leftarrow n_0 + 1$; $n_1 \leftarrow n_1 + 1$; (* Y_1^1 *)
  }
  else
  {
    Apply the P_0 grafting process on a pair $x < y$;
    if $c < x < y$ **then** $n_1 \leftarrow n_1 + 2$; (* X_0^2 *)
    if $x < y < c$ **then** $n_0 \leftarrow n_0 + 2$; (* X_2^0 *)
    if $x < c < y$ **then** $n_0 \leftarrow n_0 + 1$; $n_1 \leftarrow n_1 + 1$; (* X_1^1 *)
  }
}

if $n_0 < k$ **then** prune $k - n_0$ elements below the centre c of H_{2r}.
if $n_1 < k$ **then** prune $k - n_1$ elements above the centre c of H_{2r}.

Figure 5: The factory of Schönhage, Paterson and Pippenger.

The intuition behind this algorithm is simple. If we already have many grafted elements below the centre, then by comparing c with the larger element y of a pair $x < y$, we force the adversary to select one of the following two options: (1) Give us another pair below the centre; (2) Give us at least one additional element above the centre. In both cases we stand to gain. In the first case, the elements below the centre are organized in pairs, not in singletons, and their recycling cost is therefore reduced. In the second case, we get relatively cheap elements above the centre, in addition to cheap elements that were already obtained below the centre.

The elements above the centre of the generated S_k^k form a collection of disjoint P_{0^i}'s and the elements below the centre form a collection of disjoint P_{1^i}'s. When the lower or upper part of an S_k^k is returned to the factory, some of the existing relations among the elements returned are utilized. The amortised analysis of the green factory G_k encompasses a trade-off between the cost of generating an S_k^k and the utility obtained from its lower or upper parts when these parts are recycled. Although the S_k^k's generated by the factory of Schönhage *et al.* may contain P_{0^i}'s and P_{1^i}'s, where $i > 1$, their factory is only capable of

utilizing pairwise disjoint relations among the elements returned to it (as their grafting processes can only use pairs). If a P_{0^i} or a P_{1^i}, with $i > 1$, is returned to the factory, it is immediately broken into 2^{i-1} P_0's or P_1's. Note that both P_0 and P_1 simply stand for a pair of elements. It can be checked, see [SPP76], that the upper and lower element costs of this factory are $u_1, u_0 \sim 3$. This is the best factory obtained by Schönhage, Paterson and Pippenger.

5 Advanced principles of factory design

In this section, we outline the principles used to construct our improved factories. The first of these principles was already mentioned.

- Allowing variations in the produced partial orders.

  Our factories construct partial orders from $\tilde{\mathcal{S}}_k^k$. The exact proportion between the number of elements below and above the centre of a generated partial order is not fixed in advance.

- Recycling larger relations.

  The factories of Schönhage *et al.* are only capable of recycling singletons and pairs (i.e., P_0's, P_1's and P_λ's). Our factories recycle larger constructs such as quartets (P_{00}'s and P_{11}'s), octets (P_{000}'s and P_{111}'s), 16-tuples (P_{0000}'s and P_{1111}'s) as well as pairs, singletons and other structures (e.g., I_3's, sorted lists of three elements) which are not hyperpairs (see Fig. 13 in Section 8). The non-hyperpair constructs are obtained by the more sophisticated grafting processes used.

  As pairs can be used both as P_0's and P_1's, they can be used to construct a hyperpair P_w, no matter what the string w is. This is one of the reasons why recycling pairs is easier than recycling larger hyperpairs. Quartets such as P_{00}'s (or their equivalent P_{01}'s), for example, can only be used to construct a hyperpair P_w if w starts with 0. More limitations are imposed when larger hyperpairs are considered. Non-hyperpairs cannot be used directly for the construction of hyperpairs.

  To enable the recycling of larger relations, we must be able to use them for the construction of hyperpair-like relations. We should also be able to use them for grafting.

- Constructing hyper-products.

  As mentioned, our factories may receive partial orders that could not be used for the construction of hyperpairs. These partial orders are used instead for the construction of *hyper-products*. A hyper-product $P_w \circ I$, where I is some partial order with a distinguished element, which is again called a centre, is a hyperpair P_w that each of its elements is also the centre of a disjoint copy of the partial order I. Hyperpairs are of course special cases of hyper-products as $P_w \circ P_0 = P_{0w}$ and $P_w \circ P_1 = P_{1w}$.

  Some of the partial orders I used for hyper-products construction are unbalanced. To counter this, they are composed with oppositely unbalanced hyperpairs.

- Grafting larger relations and mass-grafting.

  The factories of Schönhage *et al.* use a simple pair of pair grafting processes. We use more complicated grafting processes, even for grafting pairs. For each input construct we have several different grafting processes. Some of our grafting processes use the technique of mass production.

  Our factories apply each grafting process a certain number of times and record the number of times each outcome is obtained. From time to time, they decide to place a combination of outcomes, that tend to balance each other nicely, in the output partial order. The factory algorithms make sure that for any adversary strategy, if each grafting process is applied a certain number of times, then at least one such balanced combination can be put in output partial order. The output combinations

are chosen to have low 'local' lower and upper unit costs. The maxima of all these local upper and lower element costs are the overall upper and lower element costs of the factory.

- Using sub-factories.

  The factories of Schönhage *et al.* generate only a single family of hyperpairs (corresponding to $\mathcal{W} = 01(10)^\omega$). Our factories generate several types of hyperpairs and hyper-products, as mentioned above. The construction of each one of these hyper-products is carried out in a separate sub-production unit that we refer to as a *sub-factory*. Different sub-factories also differ in the 'raw-materials' that they can process.

- Using credits in an amortised complexity analysis.

  The last principle is an accounting principle. The different constructs recycled by our factories are of different 'quality'. Some of them, like pairs and I_3's (sorted lists of three elements), can be used very efficiently for the construction of partial orders from $\tilde{\mathcal{S}}_k^k$. Others, like P_{000}'s and P_{0000}'s are not so appropriate for this process, as they are extremely unbalanced, and using them as raw materials for the construction of partial orders from $\tilde{\mathcal{S}}_k^k$ results in a much higher production cost. To equalize these costs, each construct used by our factories is assigned a *credit*, which may be either positive or negative. The credit assigned to a construct Q is denoted by *credit*(Q). When a construct Q is recycled, extra *credit*(Q) comparisons are charged to the partial order from $\tilde{\mathcal{S}}_k^k$ that is being broken. These *credit*(Q) comparisons can then be deducted from the cost of the partial order from $\tilde{\mathcal{S}}_k^k$ that will be constructed using this construct Q. High quality materials, such as I_3's, carry negative credits. The credit attach to Singletons (P_λ's) must be zero as the singletons initially fed into the factory carry no credit.

In the next section we describe the general framework of our improved factories. This framework combines the principles put forth in this and in the previous sections. We also give a general description of the complexity analysis.

6 The framework of the improved factories

A factory $\mathcal{G}$ of the type we are using is composed of several sub-factories $g_1, \ldots, g_r$. These sub-factories are activated in "parallel". Each sub-factory is either working on the construction of a partial order from $\tilde{\mathcal{S}}_k^k$, or waiting for additional raw materials. Whenever the construction of a partial order from $\tilde{\mathcal{S}}_k^k$ in one of sub-factories is finished, this partial order is output and the sub-factory begins to work on the construction of a new partial order from $\tilde{\mathcal{S}}_k^k$. While designing such a factory we have to make sure that if enough materials are fed into the whole factory, then at least one of its sub-factories can make progress in constructing the next partial order from $\tilde{\mathcal{S}}_k^k$. The production residue of the factory $\mathcal{G}$ is the sum of the production residues of the sub-factories $g_1, \ldots, g_r$.

The operation of each sub-factory is essentially independent of the operation of the other sub-factories. Each sub-factory processes a specific type (or in some cases, specific types) of input constructs. There may be, for example, a P_{00}-processing sub-factory, an I_3-processing sub-factory and so on. A construct recycled to the factory $\mathcal{G}$ is immediately fed to an appropriate sub-factory. A Q-processing sub-factory (a Q sub-factory for short) may sometimes produce, as by-products, partial orders which are not Q's. These partial orders are immediately fed into sub-factories that can consume them.

The upper and lower element costs of $\mathcal{G}$ are the maxima of the (amortised) upper and lower element costs of $g_1, \ldots, g_r$. The credits attached to the different constructs are used to equalise the costs of the different sub-factories, thereby reducing their maximum. The credits selected optimise a natural tradeoff between

the generation cost of an output partial order that contains a partial order Q and the cost of utilising Q when it is recycled.

The factory $\mathcal{G}$ can be viewed as the 'union' of the sub-factories $g_1, \ldots, g_r$. In the next subsection we describe the structure of a generic sub-factory. In Subsection 6.2 we then describe the cost analysis of a generic sub-factory.

6.1 The structure of a generic sub-factory

Each sub-factory g is composed of a hyper-product generation process, a hyper-product pruning process, a collection $A_1, A_2, \ldots, A_\ell$ of grafting processes, and a list $C_1, C_2, \ldots, C_m$ of output combinations.

The operation of each sub-factory, like the simple factories described in Section 4, is composed of three main phases. In the first phase a large hyper-product is constructed using the hyper-product generation process. In the second stage the grafting processes $A_1, A_2, \ldots, A_\ell$ are activated. Whenever a combination of outcomes from the list $C_1, C_2, \ldots, C_m$ is encountered, this combination is placed in the output partial order. The process continues until at least k grafted elements are obtained either above or below the centre c of the hyper-product. An appropriate number of elements, determined by the output combinations that were used, is then pruned using the hyper-product pruning process. The factory then outputs the partial order generated.

The hyper-product construction process of a sub-factory must be able to use all the constructs that may be fed into the sub-factory as inputs. For each possible input construct there should also be at least one grafting process that can utilise it. Typically, a sub-factory would have more than one such grafting process for each possible input construct. This gives the sub-factory the freedom to choose different grafting processes in different circumstances and helps balance the upper and lower costs of the sub-factory.

Each grafting process A_i has a list $a_{i,1}, a_{i,2}, \ldots, a_{i,l_i}$ of possible outcomes. Let $c_1(a_{i,j})$ and $c_0(a_{i,j})$ denote the upper and lower costs of $a_{i,j}$ (these costs are defined in the next subsection) and let $n_1(a_{i,j})$ and $n_0(a_{i,j})$ denote the number of elements above and below c, respectively, in the outcome $a_{i,j}$. When applying the grafting process A_i we do not know in advance which outcome will result. This is decided by the adversary.

For the accounting purposes, it is convenient to view pruning as a special grafting process. We therefore refer to pruning as the 0-th grafting process A_0 and let $a_{0,0}$ and $a_{0,1}$ stand for lower and upper pruned elements. The costs $c_0(a_{0,0})$ and $c_1(a_{0,1})$ are just and lower and upper element pruning costs. This special 'grafting' process is different from the other grafting processes in two major respects. The first is that the algorithm, and not the adversary, chooses the outcome. The second is that elements are not pruned one at a time. Instead, two counters are used to maintain the number of elements that should be pruned above and below the centre. The required numbers of elements are then pruned after all the grafting processes have finished.

An output combination $C_i = (r_{i,1} \times b_{i,1}, r_{i,2} \times b_{i,2}, \ldots, r_{i,s_i} \times b_{i,s_i})$ is a weighted list of outcomes, where $r_{i,1}, r_{i,2}, \ldots, r_{i,s_i} > 0$ are *real* numbers and each $b_{i,j}$ is an outcome $a_{i',j'}$ of one of the grafting processes A_0 and $A_1, \ldots, A_\ell$ employed by the sub-factory (the use of real numbers will be justified later). Note that A_0 is the pruning process of the sub-factory, disguised as a grafting process, and an output combination may therefore involve elements that should be obtained by pruning. In our factories most combinations involve only two outcomes.

For each possible outcome $a_{i,j}$ of the grafting processes, the sub-factory maintains a counter $\#(a_{i,j})$ of the number of outcomes of type $a_{i,j}$ which were obtained, but not yet consumed. Initially $\#(a_{i,j}) = 0$ for every $i \geq 1$ and $j \geq 1$, and $\#(a_{0,j}) = 2k$, for $j = 0, 1$. Recall that $a_{0,0}$ and $a_{0,1}$ represent pruned elements. The initial values given to $\#(a_{0,0})$ and $\#(a_{0,1})$ reflect the fact that up to $2k$ elements can be pruned on either side of the hyper-product constructed by the sub-factory. Two more counters k_0 and k_1 maintain

the number of elements below and above the centre c that were already placed in the output partial order. The counters k_0 and k_1 are initially set to 0.

An output combination $C_i = (r_{i,1} \times b_{i,1}, r_{i,2} \times b_{i,2}, \ldots, r_{i,s_i} \times b_{i,s_i})$ can be applied if $\#(b_{i,j}) \geq r_{i,j}$ for every $1 \leq j \leq s_i$. If these conditions are satisfied, then $r_{i,j}$ copies of outcome $b_{i,j}$, for $1 \leq j \leq s_i$, are 'placed' in the output partial order and all the counters are updated accordingly. The output combination C_i contains $n_0(C_i) = \sum_{j=1}^{s_i} r_{i,j} \cdot n_0(b_{i,j})$ elements below the centre c and $n_1(C_i) = \sum_{j=1}^{s_i} r_{i,j} \cdot n_1(b_{i,j})$ elements above the centre c of the hyper-product. All the combinations used in our factories satisfy the condition $\frac{1}{2} < n_0(C_i)/n_1(C_i) < 2$. The partial orders produced by our factories are therefore of the form $S_{k'}^{k''}$, where $k \leq k' \leq 2k$ and $k \leq k'' \leq 2k$, and hence members of $\tilde{\mathcal{S}}_k^k$. The factor 2 used in the definition of $\tilde{\mathcal{S}}_k^k$ is arbitrary and can be increased if necessary.

The sub-factory applies each grafting process A_i, where $1 \leq i \leq \ell$, sufficiently many times so that there is at least one outcome $a_{i,j}$ for which $\#(a_{i,j}) \geq \mathit{quota}$, where *quota* is some sufficiently large constant. A suitable choice for *quota* is the maximum $\max_{i,j} r_{i,j}$ of the coefficients that appear in the output combinations of the sub-factory. The combination list of the sub-factory should have the property that if for each $1 \leq i \leq l$ there exists at least one $1 \leq j \leq l_i$ such that $\#(a_{i,j}) \geq \mathit{quota}$, then at least one of the output combinations can be applied. This ensures that the sub-factory will never get 'stuck', no matter what the outcomes of the grafting processes will be.

In the above description, we spoke freely about 'placing' $r_{i,j}$ copies of an outcome $b_{i,j}$ in the output partial order, even though $r_{i,j}$ was not necessarily an integer. This was only done, however, for accounting purposes. The outcomes (or fractions of outcomes) 'placed' in the output partial order are the outcomes whose cost was already accounted for. At most a fixed number of outcomes of each type are unaccounted for by this analysis. Their contribution to the cost is, therefore, negligible. These unaccounted for outcomes can either be placed in the output partial order or be used to construct the next output partial order.

A pseudo-code describing the operation of a generic sub-factory is given in Fig. 6.

6.2 The analysis of a generic sub-factory

We now turn to the complexity analysis of a generic sub-factory. We start by defining the lower and upper costs of each outcome $a_{i,j}$ of the grafting processes. Let $I_1, \ldots, I_r$ be the input constructs consumed by the flow of the grafting process A_i that produces $a_{i,j}$. Let $V_1, \ldots, V_s$ be the constructs above the centre c of the hyper-product from which the outcome $a_{i,j}$ is composed and let $\Lambda_1, \ldots, \Lambda_t$ be the constructs below the centre c from which $a_{i,j}$ is composed. Let $R_1, \ldots, R_p$ be the leftovers of this process, i.e., the parts of $I_1, \ldots, I_r$ that do not become parts of either $V_1, \ldots, V_s$ or $\Lambda_1, \ldots, \Lambda_t$. The leftovers $R_1, \ldots, R_p$ are recycled to the appropriate sub-factory.

The *generation cost*, $gen(a_{i,j})$, of an outcome $a_{i,j}$ is the number of edges cut during the generation of an $a_{i,j}$. The *lower separation cost*, $sep_0(a_{i,j})$, of an $a_{i,j}$ is the number of edges that have to be cut to separate the constructs $\Lambda_1, \ldots, \Lambda_t$ from the centre c of the hyper-product. The *lower elimination cost*, $elm_0(a_{i,j})$, of an $a_{i,j}$ is the number of edges that have to be cut to turn all the elements of $\Lambda_1, \ldots, \Lambda_t$ into singletons (and disconnect them from the centre). The Hasse diagram of most outcomes is acyclic and the lower elimination cost in such a case is just the number of elements contained in the constructs $\Lambda_1, \ldots, \Lambda_t$. The *upper separation cost*, $sep_1(a_{i,j})$, and the *upper elimination cost*, $elm_1(a_{i,j})$, are defined analogously.

When the lower side of an outcome $a_{i,j}$ is recycled, the upper side is eliminated and vice versa. Hence, we define the *lower recycling cost*, $rec_0(a_{i,j})$, of the outcome $a_{i,j}$ as $sep_0(a_{i,j}) + elm_1(a_{i,j})$. The *upper recycling cost*, $rec_1(a_{i,j})$, of $a_{i,j}$ is defined analogously as $sep_1(a_{i,j}) + elm_0(a_{i,j})$.

Each input construct I_l, where $1 \leq l \leq r$, used in the flow of the grafting process that generates the outcome $a_{i,j}$ carries a credit of $\mathit{credit}(I_l)$ units. These credit units help 'finance' the generation of $a_{i,j}$. If

1. Generate a hyper-product with at least $2k$ elements on either side of its centre c.

2. Initialise the counters:
   $\#(a_{i,j}) \leftarrow 0$ for every $1 \leq i \leq l$ and $1 \leq j \leq l_i$;
   $\#(a_{0,j}) \leftarrow 2k$ for $j = 0, 1$;
   $k_0, k_1 \leftarrow 0$;

3. Perform the following steps until $k_0 \geq k$ and $k_1 \geq k$.

   (a) `for` $i \leftarrow 1$ `to` l `do` (* Activate grafting processes *)
   `while` $\max\{\#(a_{i,1}), \ldots, \#(a_{i,l_i})\} < quota$ activate grafting process A_i.

   (b) `for` $i \leftarrow 1$ `to` m `do` (* Find appropriate output combinations *)
   `if` $\#(b_{i,j}) \geq r_{i,j}$ `for` $1 \leq j \leq s_i$ `then`
   {
   $\#(b_{i,j}) \leftarrow \#(b_{i,j}) - r_{i,j}$ for $1 \leq j \leq s_i$;
   $k_0 \leftarrow k_0 + \sum_{j=1}^{s_i} r_{i,j} \cdot n_0(b_{i,j})$;
   $k_1 \leftarrow k_1 + \sum_{j=1}^{s_i} r_{i,j} \cdot n_1(b_{i,j})$
   }

4. Let $p_0 \leftarrow 2k - \#(a_{0,0})$ and $p_1 \leftarrow 2k - \#(a_{0,1})$;
   Prune p_0 and p_1 elements below and above c, respectively.

5. Output the partial order and return to 1.

Figure 6: The generic sub-factory algorithm

the lower part of the partial order generated using the outcome $a_{i,j}$ is eventually eliminated, and its upper part recycled, we have to attach $credit(V_l)$ credit units to each of the constructs $V_1, \ldots, V_s$. If, on the other hand, the upper part is eliminated and the lower part recycled, we have to attach $credit(\Lambda_l)$ credit units to each of the constructs $\Lambda_1, \ldots, \Lambda_t$. Similarly, we have to attach $credit(R_l)$ credit units to each of the leftover $R_1, \ldots, R_p$. The *lower cost* $c_0(a_{i,j})$ and the *upper cost* $c_1(a_{i,j})$ of the outcome $a_{i,j}$ are thus defined as

$$c_0(a_{i,j}) = gen(a_{i,j}) + rec_1(a_{i,j}) - \sum_{l=1}^{r} credit(I_l) + \sum_{l=1}^{s} credit(V_l) + \sum_{l=1}^{p} credit(R_l) \ ,$$

$$c_1(a_{i,j}) = gen(a_{i,j}) + rec_0(a_{i,j}) - \sum_{l=1}^{r} credit(I_l) + \sum_{l=1}^{t} credit(\Lambda_l) + \sum_{l=1}^{p} credit(R_l) \ .$$

As mentioned before, our grafting procedures may use mass production. Hence, the costs $gen(a_{i,j})$, $rec_0(a_{i,j})$ and $rec_1(a_{i,j})$ are amortised costs and $I_1, \ldots, I_r$ are the constructs used for the construction of a single copy of the outcome $a_{i,j}$.

Let $C_i = (r_{i,1} \times b_{i,1}, \ldots, r_{i,s_i} \times b_{i,s_i})$ be an output combination. The *local upper element cost*, $u_1(C_i)$, and the *local lower element cost*, $u_0(C_i)$, of the combination C_i are defined, temporarily, as:

$$u_0(C_i) = \sum_{j=1}^{s_i} r_{i,j} \cdot c_0(b_{i,j}) \Big/ \sum_{j=1}^{s_i} r_{i,j} \cdot n_0(b_{i,1}) \quad , \quad u_1(C_i) = \sum_{j=1}^{s_i} r_{i,j} \cdot c_1(b_{i,j}) \Big/ \sum_{j=1}^{s_i} r_{i,j} \cdot n_1(b_{i,1}) \ .$$

This definition, however, is not completely adequate. So far, we have attached credits to individual constructs. We would sometimes like to attach credits to combinations of constructs. It turns out that there are some constructs that can be more efficiently utilised when constructs of a different kind are also available. As a concrete example, we will see in the next sections that quartets (P_{00}'s and P_{11}'s) can be utilised much more efficiently if some pairs (P_0's and P_1's) are also available. More specifically, we will be in a situation in which $\mathit{credit}(P_0) = 0$ and $\mathit{credit}(P_{00}) > 0$, i.e., pairs carry no credit while each quartet on its own should carry some positive credit. In contrast, a combination composed of a quartet and pair, and in fact a combination composed of up to a fixed constant $\gamma > 1$ of quartets and of a single pair, could be recycled without any credit, i.e., $\mathit{credit}(P_0, \gamma \times P_{00}) = 0$. The credit that needs to be attached to a collection of constructs may therefore be smaller than the sum of the individual credits that should be attached to each member of this collection.

The above temporary definition of $u_0(C_i)$ and $u_1(C_i)$ does not take such considerations into account. It is not enough to replace the sums $\sum_{l=1}^{s} \mathit{credit}(V_l)$, $\sum_{l=1}^{t} \mathit{credit}(\Lambda_l)$ and $\sum_{l=1}^{p} \mathit{credit}(R_l)$ in the definitions of $c_0(a_{i,j})$ and $c_1(a_{i,j})$ by $\mathit{credit}(V_1, \ldots, V_s)$, $\mathit{credit}(\Lambda_1, \ldots, \Lambda_t)$ and $\mathit{credit}(R_1, \ldots, R_p)$, respectively, as good combinations may be formed using constructs obtained while constructing or recycling different outcomes that participate in the output combination. As an example, pairs are perhaps obtained when the $b_{i,1}$'s outcomes that participate in the output combination are recycled while quartets are perhaps obtained when the $b_{i,2}$'s outcomes are recycled. We therefore amend the definitions of $u_0(C_i)$ and $u_1(C_i)$ in the following way:

$$u_0(C_i) = \frac{\left[\sum_{j=1}^{s_i} r_{i,j} \cdot (\mathit{gen}(b_{i,j}) + \mathit{rec}_1(b_{i,j}))\right] - \sum_{I \in \mathcal{I}_i} \mathit{credit}(I) + \mathit{credit}(\mathcal{V}_i) + \mathit{credit}(\mathcal{R}_i)}{\sum_{j=1}^{s_i} r_{i,j} \cdot n_0(b_{i,j})} \,, \qquad (1)$$

$$u_1(C_i) = \frac{\left[\sum_{j=1}^{s_i} r_{i,j} \cdot (\mathit{gen}(b_{i,j}) + \mathit{rec}_0(b_{i,j}))\right] - \sum_{I \in \mathcal{I}_i} \mathit{credit}(I) + \mathit{credit}(\Lambda_i) + \mathit{credit}(\mathcal{R}_i)}{\sum_{j=1}^{s_i} r_{i,j} \cdot n_1(b_{i,j}))} \,, \qquad (2)$$

where $\mathcal{I}_i$, Λ_i, $\mathcal{V}_i$ and $\mathcal{R}_i$ are the (weighted) collections of all input constructs, recycled constructs and left-overs involved in the generation and the recycling of all the outcomes composing the combination C_i. The terms $\mathit{credit}(\mathcal{V}_i)$, $\mathit{credit}(\Lambda_i)$ and $\mathit{credit}(\mathcal{R}_i)$ represent the credits that should be attached to the collections $\mathcal{V}_i$, Λ_i and $\mathcal{R}_i$.

In practice, credits are attached separately to most constructs involved in the collections Λ_i,$\mathcal{V}_i$ and $\mathcal{R}_i$. In the factory described in Section 8, for example, the only exception to this is the grouping of pairs and quartets together and the formation of compound constructs of the form $(P_0, \gamma \times P_{00})$.

In general, each factory $\mathcal{G}$ has a set $\mathcal{P}$ of both simple and compound constructs to which credits are assigned. The set $\mathcal{P}$ includes all the basic constructs that can be consumed and that are recycled by the factory. Every collection Λ_i,$\mathcal{V}_i$ or $\mathcal{R}_i$ of constructs is then treated as a combination of constructs taken from $\mathcal{P}$ in a way that minimises the total credit that should be attached to the collection.

The lower and upper element costs $u_0(g)$ and $u_1(g)$ of the sub-factory g with a list $C_1, \ldots, C_m$ of output combinations are defined to be

$$u_0(g) = \max_{i=1}^{m} u_0(C_i) \quad , \quad u_1(g) = \max_{i=1}^{m} u_1(C_i) \,.$$

We are now in a position to state the following theorem.

Theorem 6.1 *If $\mathcal{G}$ is a factory that employs the sub-factories $g_1, g_2, \ldots, g_r$ then the lower and upper element costs of the factory $\mathcal{G}$ are*

$$u_0(\mathcal{G}) \sim \max_{j=1}^{r} u_0(g_j) \quad , \quad u_1(\mathcal{G}) \sim \max_{j=1}^{r} u_1(g_j) \,.$$

Proof: Let $\mathcal{P}$ be the set of basic and compound constructs recycled by the factory $\mathcal{G}$. We assume that for each construct from $\mathcal{P}$ there is at least one sub-factory among $g_1, g_2, \ldots, g_r$ that can consume it. We also assume that each sub-factory g_j, where $1 \leq j \leq r$, has an adequate list of output combinations, in the sense that if each grafting process employed by g_j is applied sufficiently many times, then at least one output combination can be used. We also assume that if enough input constructs are fed into the factory then at least one sub-factory can generate a partial order from $\tilde{\mathcal{S}}_k^k$.

Each construct from $\mathcal{P}$ has a specific amount of credit attached to it. The credit attached to singletons (P_λ's) is zero. The singletons that are initially fed into the factory thus carry the required amount of credit. Whenever constructs from $\mathcal{P}$ are recycled, we make sure that they too carry the correct credit. Each amount of credit used in the construction of a partial order from $\tilde{\mathcal{S}}_k^k$ is therefore paid for in full during the generation of other partial orders from $\tilde{\mathcal{S}}_k^k$.

The local upper and lower element costs of a combination C_i used in one of the factories were defined as $u_0(C_i) = c_0(C_i)/n_0(C_i)$ and $u_1(C_i) = c_1(C_i)/n_1(C_i)$, where $c_0(C_i)$ and $c_1(C_i)$ are the amortised lower and upper cost of the combination C_i (the numerators of equations (1) and (2)), and $n_0(C_i)$ and $n_1(C_i)$ are the number of elements in C_i above and below the centre (the denominators of equations (1) and (2)). Let $u_0 = \max_{j=1}^{r} u_0(g_j)$ and $u_1 = \max_{j=1}^{r} u_1(g_j)$. It follows that for every combination C_i used in one of the sub-factories $g_1, \ldots, g_r$ we have $u_0(C_i) \leq u_0$ and $u_1(C_i) \leq u_1$.

Suppose that an output partial order is generated in sub-factory g_j. Suppose that $C_{i_1}, C_{i_2}, \ldots, C_{i_a}$ are the output combinations that contribute to this output partial order. The total amortised cost of producing the partial order, eliminating its lower side and recycling its upper side is $\sum_{l=1}^{a} c_0(C_{i_l})$. The total number of elements below the centre of the produced partial order is $\sum_{l=1}^{a} n_0(C_{i_l})$. The amortised, per element, cost of the produced partial order is therefore $\sum_{l=1}^{a} c_0(C_{i_l}) \Big/ \sum_{l=1}^{a} n_0(C_{i_l})$. As for each $1 \leq l \leq a$ we have $c_0(C_{i_l})/n_0(C_{i_l}) \leq u_0$, we get that $\sum_{l=1}^{a} c_0(C_{i_l}) \Big/ \sum_{l=1}^{a} n_0(C_{i_l}) \leq u_0$. Similarly, we also get that the amortised, per element, upper cost of the produced partial order is at most u_1.

The above accounting did not take into account the cost of breaking the unused parts of the hyper-product. The cost of this operation is negligible, however, compared to the generation cost of the output partial order. It also did not take into account the generation and recycling costs of the outcomes of the grafting processes that were not used to construct the output partial order. There may however be only a constant number of such outcomes and the costs associated with them are therefore also negligible.

As a result, we get that $u_0(\mathcal{G}) = u_0 + o(1)$ and $u_1(\mathcal{G}) = u_1 + o(1)$ and the theorem is proved. □

Our concrete factories are described in Sections 8 and 9. For each one of our factories we specify the set $\mathcal{P}$ of basic and compound constructs used and the credit attached to each one of its elements. We then describe the sub-factories employed by the factory. For each sub-factory we specify the hyper-product generation process and grafting processes used, and finally its list of output combinations. We verify that each construct from $\mathcal{P}$ can indeed be consumed by at least one sub-factory and that the list of output combinations of each sub-factory is adequate. For each output combination we then compute its local lower and upper element costs, using the definitions given in equations (1) and (2). As implied by Theorem 6.1, the maxima of these values are then the lower and upper element costs of the whole factory. We begin the description of our factories by describing, in the next section, the grafting processes used by them.

7 Advanced grafting processes

In this section we describe the grafting processes used by our improved factories.

7.1 Recursive pair grafting

The recursive pair grafting is, as its name suggests, a recursive version of the basic pair grafting procedure described in Subsection 4.2. There are two variants of this process depending on whether the upper elements or lower elements of pairs are compared to the centre of the hyper-product. We describe the variant in which lower elements are compared. The other variant is symmetric.

The recursive pair grafting recursively builds hyperpairs which are *dominated* by the centre c of the hyper-product. A hyperpair $P = P_{1^i}$ with centre c' is *dominated* by c if every element of P, except possibly for c', is known to be smaller than c. A dominated hyperpair P_{1^i} is said to be of *level* i.

The process is composed of rounds. The i-th round receives two dominated hyperpairs P^1 and P^2 (with centres c_1 and c_2, respectively) of level i and attempts to construct a dominated hyperpair of level $i+1$. This is done by first comparing the centres c_1 and c_2 of these two dominated hyperpairs, thus obtaining a hyperpair $P = P_{1^{i+1}}$. We may assume, without loss of generality, that $c_2 > c_1$ and c_2 is therefore the centre of the new hyperpair formed. We then compare c_1 (i.e., *not* c_2, the centre of the hyperpair P) with c. The two possible outcomes are:

(1) $c_1 < c$ and P is a dominated hyperpair of level $i+1$.

(2) $c_1 > c$ and P is not dominated by c (as $c_2 > c_1 > c$).

If (2) occurs, the process is stopped. For the purposes of $\mathcal{G}_k$ and $\mathcal{G}'_k$ (the factories described in Section 8 and 9) we also stop the process when a dominated hyperpair P_{111} of level 3 is generated. We then separate the generated hyperpair into two hyperpairs: A dominated hyperpair P^1 of level 2 and a hyperpair P^2 of level 2 all whose elements are known to be below the centre. The hyperpair P^2 is an output of this grafting process. The dominated hyperpair P^1 is used to construct the next dominated hyperpair of level 3.

As described, the two hyperpairs P^1 and P^2 fed into the 0-th round of the process are two singletons c_1 and c_2. The 0-th round then starts by comparing these two singletons thus forming a pair. Instead of feeding the the 0-th round of the process with singletons, we can therefore feed it with pairs and simply skip the first comparison of this round.

The recursive pair grafting process is described schematically in Fig. 7. The dashed edges in the figure represent edges that became redundant. The four possible outcomes of this process are denoted by U_0^2, $U_{1,1}^2$, U_6^2, and U_4^0. These four outcomes, as well as the four outcomes L_2^0, $L_2^{1,1}$, L_2^6, and L_0^4 of the symmetic grafting process in which upper elements are compared to the centre are shown in Fig. 8.

We now turn to the cost analysis of this grafting process. As explained, we do not count the number of comparisons made, but rather the number edges cut. Let us analyse the number of edges cut during the i-th round of the process. Let P^1 and P^2 be the two dominated hyperairs of level i fed to the i-th round, let c_1 and c_2 be their centres and assume, without loss of generality, that the first comparison of the round established that $c_1 < c_2$. It is easy to verify, using induction, that i edges connect c with elements of P^1 and that i edges connect c with elements of P^2. Similarly, i edges connect c_1 with other elements of P^1 and i edges connect c_2 with other elements of P^2. If the i-th round results in a dominated hyperpair of level $i+1$, i.e., if $c_1 < c$ (case (1)), then the i edges connecting c with the elements of P^1 become redundant and are therefore cut. If on the other hand $c_1 > c$ (case (2)), then the $2i$ edges connecting c_1 and c_2 with elements of their hyperpairs become redundant and therefore cut. Finally, note that an additional edge is cut when a dominated hyperpair P_{111} of level 3 is separated into a dominated hyperpair of a level 2 and to an U_4^0. Using these obseravtions, it is easy to verify that the generation costs of the different outcomes shown in Fig. 8 are as shown in Table 2.

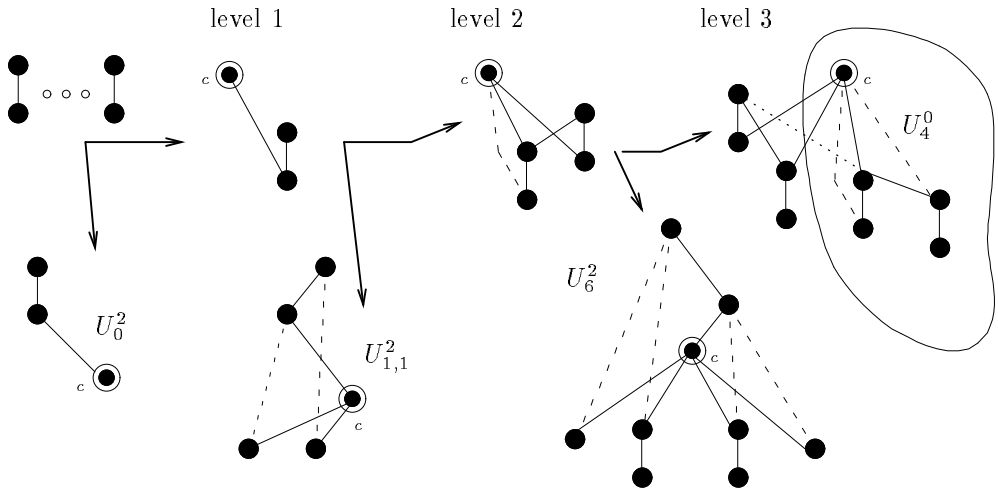

Figure 7: The recursive pair grafting process.

Outcome	Above	Below	gen	rec_0	rec_1
U_0^2	P_0	—	0	2	1
$U_{1,1}^2$	P_0	$2\times P_\lambda$	2	4	3
U_6^2	P_0	$2\times P_\lambda$, $2\times P_0$	6	6	7
U_4^0	—	P_{11}	4	1	4

Outcome	Above	Below	gen	rec_0	rec_1
L_2^0	—	P_0	0	1	2
$L_2^{1,1}$	$2\times P_\lambda$	P_0	2	3	4
L_2^6	$2\times P_\lambda$, $2\times P_0$	P_0	6	7	6
L_0^4	P_{00}	—	4	4	1

Table 2: The costs of the different outcomes of the recursive pair grafting processes.

7.2 Balanced quartet grafting

The balanced quartet grafting process is a very straightforward process. We describe the process that grafts P_{00}'s. A symmetric process can be used for grafting P_{11}'s. The grating process always consumes a single quartet and returns one of five possible outcomes.

Let u, v, w, z be the elements of a P_{00}, where $u < v$ and $u < w < z$. Start by comparing w with c. If $w > c$, cut the edge connecting w and u. The obtained outcome is denoted by Q_0^2. The remaining pair is recycled. If $w < c$ then compare v and z with c. The four different outcomes of these comparisons give rise to the outcomes $Q_2^{1,1}$, Q_3^1 and Q_4^0 depicted in Fig. 9. Dashed edges in this figure represent again edges that became redundant during the grafting process.

The elements in a $\bar{Q}_3^1$ outcome satisfy all the relations satisfied by the elements of a Q_3^1 outcome. A $\bar{Q}_3^1$ outcome is therefore a special case of a Q_3^1 outcome and we shall not consider it separately. The generation and recycling costs of the remaining four possibilities are given in Table 3.

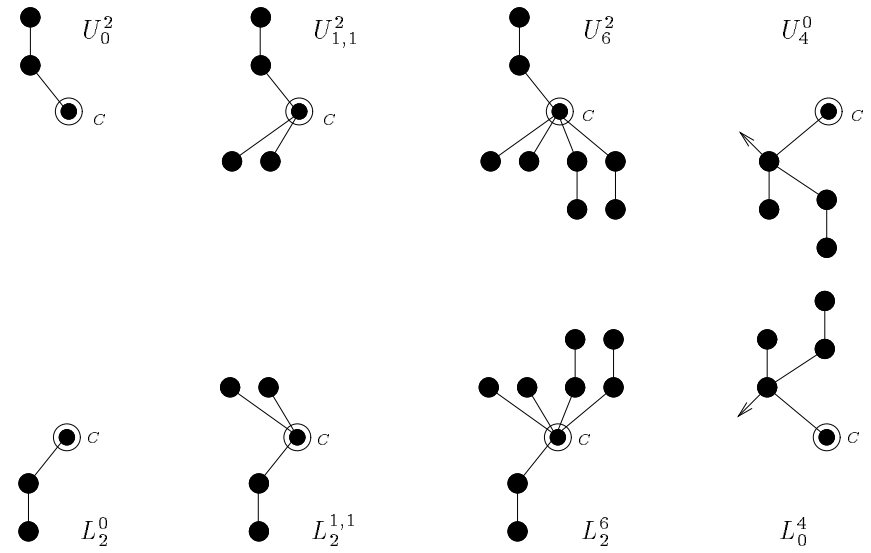

Figure 8: Possible outcomes of recursive pair grafting.

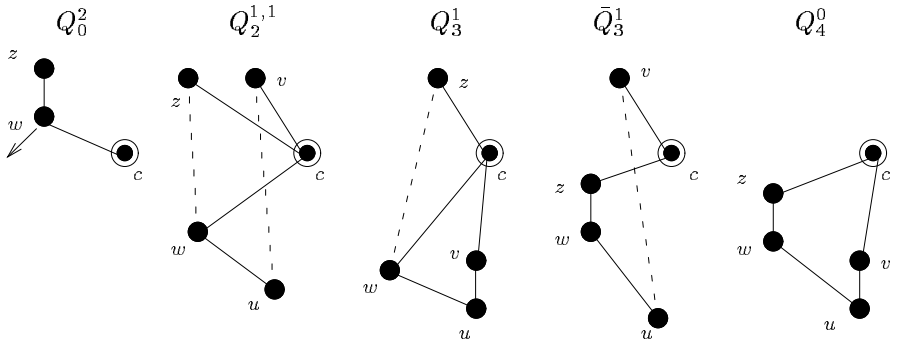

Figure 9: Possible outcomes of balanced quartet (P_{00}) grafting.

7.3 Extended balanced quartet grafting

The grafting process described in this subsection is an extension of the simple grafting process described in the previous subsection. We again describe the process that grafts P_{00}'s. A symmetric process can be used for grafting P_{11}'s.

Let u, v, w, z be the elements of a P_{00}, where $u < v$ and $u < w < z$. The process starts again by comapring w and c. If $w < c$, continue as before, obtaining one of the outcomes $Q_2^{1,1}$, Q_3^1 or Q_4^0shown in Fig. 9. If, however, $w > c$, then instead of removing the redundant pair, activate the recursive pair grafting process, as described in Subsection 7.1, on the redundant pair $u < v$. The four possible outcomes of this process are shown in Fig. 10. The outcomes Q_0^4, Q_2^6, Q_6^{10}, and Q_4^4 are obtained, respectively, from the U_0^2, $U_{1,1}^2$, U_6^2, and U_4^0outcomes of the recursive pair grafting process. The geneation and recycling costs of all the outcomes of this grafting process are given in Table 4.

Outcome	Above	Below	Leftovers	gen	rec_0	rec_1
Q_0^2	P_0	—	P_0	1	2	1
$Q_2^{1,1}$	$2\times P_\lambda$	P_0	—	2	3	4
Q_3^1	P_λ	W_3	—	1	3	5
Q_4^0	—	P_{00}	—	1	2	5

Table 3: The costs of the different outcomes of the balanced P_{00} grafting process.

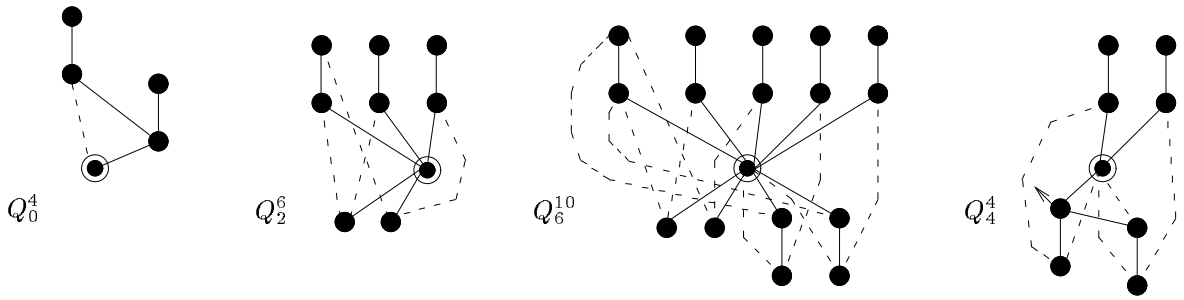

Figure 10: Possible outcomes of extended balanced grafting of P_{00}'s.

7.4 Unbalanced quartet grafting

The unbalanced quartet grafting process is another quartet grafting process used by our factories. Most of its outcomes are less balanced than the results obtained using the two previous quartet grafting processes. Again we describe the version of the process that grafts P_{00}'s. A symmetric version can be used to graft P_{11}'s.

Let u, v, w, z be the elements of a P_{00}, where $u < v$ and $u < w < z$. The two balanced quartet grafting processes started by comparing w with c, the centre of the hyper-product. The unbalanced process, on the other hand, starts by comparing v and z to c. The possible outcomes are shown at the top of Figure 11.

In the first two cases, i.e., if $v, z < c$, or if $z < c < v$, the grafting process stops. The obtained outcomes are denoted, respectively, by R_4^0 and R_3^1. The other two cases, i.e., $v < c < z$ and $c < v, z$ are more complicated.

If $v < c < z$, use the pair $w < z$, and other pairs like it, as inputs to the recursive P_0 grafting process, i.e., the mirror image of the grafting process described in Subsection 7.1. The recursive pair grafting process

Outcome	Above	Below	gen	rec_0	rec_1
$Q_2^{1,1}$	—	P_0	2	3	4
Q_3^1	P_λ	W_3	1	3	5
Q_4^0	—	P_{00}	1	2	5
Q_0^4	P_{00}	—	1	4	1
Q_2^6	$3\times P_0$	$2\times P_\lambda$	4	8	5
Q_6^{10}	$5\times P_0$	$2\times P_\lambda$, $2\times P_0$	10	14	11
Q_4^4	$2\times P_0$	P_{11}	6	5	6

Table 4: The costs of the different outcomes of the extended balanced P_{00} grafting process.

results in the creation of $L_2^{1,1}$'s, L_2^6's and L_0^4's. These $L_2^{1,1}$'s, L_2^6's and L_0^4's are composed of w and z elements of P_{00}'s. By adding to the obtained constructs the u and the v elements of the quartets from which these w and z elements were taken, we obtain outcomes of the forms R_6^2, R_{10}^6 and R_4^4 shown in Fig. 11.

Finally, if $c < v, z$, then the situation is identical to the situation after the first comparion of the second round of the recursive P_0 grafting. By continuing the recursive P_0 process from this point, possibly using another P_{00} for which $c < v, z$, we obtain an $L_2^{1,1}$, L_2^6 or an L_0^4.

The unbalanced P_{00} grafting has eight outcomes in total, R_4^0, R_3^1, R_6^2, R_{10}^6, R_4^4 and $L_2^{1,1}$, L_2^6 and L_0^4. The last three outcomes are excellent outcomes as we have been able to use quartets as if they were pairs. The costs associated with the outcomes R_4^0, R_3^1, R_6^2, R_{10}^6, R_4^4 are given in Table 5. The costs of the outcomes $L_2^{1,1}$, L_2^6 and L_0^4 are as shown in Table 2.

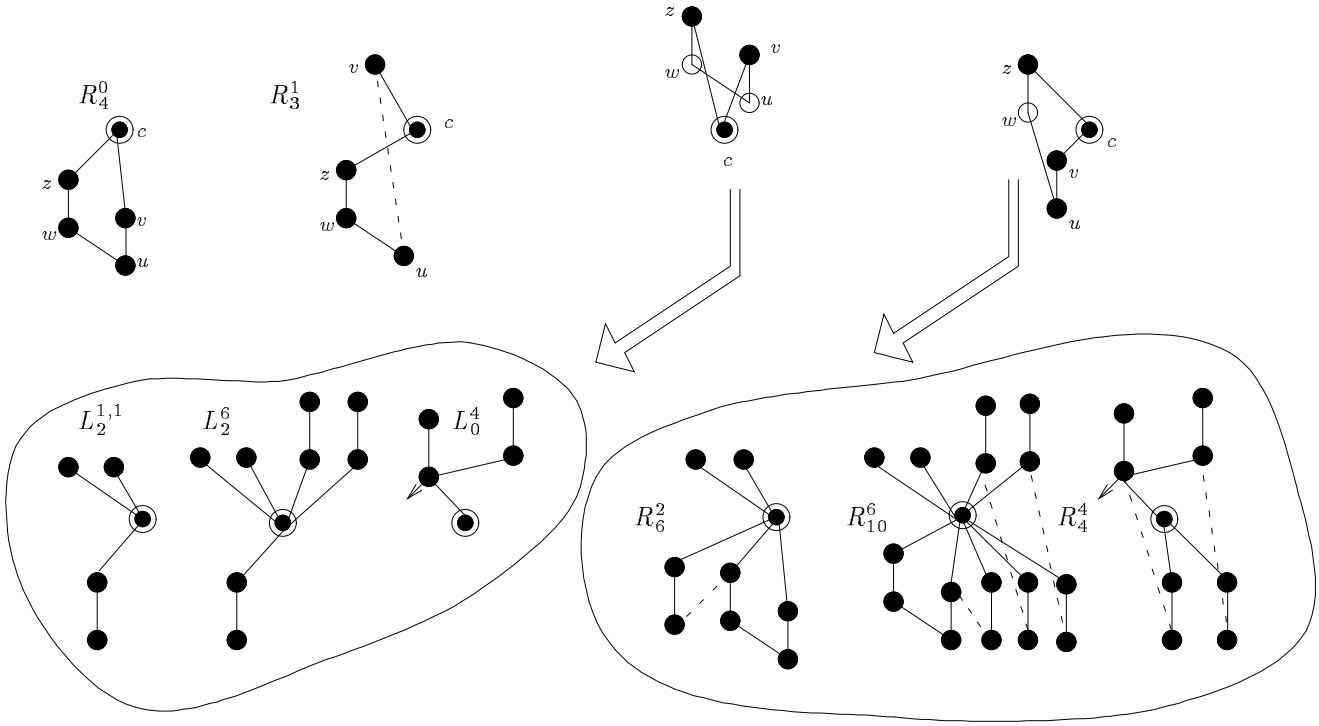

Figure 11: Possible outcomes of unbalanced P_{00} grafting.

Outcome	Above	Below	*gen*	rec_0	rec_1
R_4^0	—	P_{00}	0	2	5
R_3^1	P_λ	I_3	1	2	4
R_6^2	$2\times P_\lambda$	P_{00}, P_0	3	5	9
R_{10}^6	$2\times P_\lambda$, $2\times P_0$	$3\times P_0$, P_{00}	9	11	15
R_4^4	P_{00}	$2\times P_0$	6	6	5

Additional outcomes: $L_2^{1,1}$, L_2^6, L_0^4

Table 5: The costs of the different outcomes of the unbalanced P_{00} grafting process.

construct	credit
P_λ	0
P_0, P_1	0
P_{00}, P_{11}	0.1093
I_3, M_3, W_3	0.5000
$(P_0, \gamma \times P_{00}), (P_1, \gamma \times P_{11})$	0

Table 6: The credits attached to the constructs used by the factory $\mathcal{G}'_k$.

8 A factory with $u_0, u_1 \sim 2.955$

The construction of the factory $\mathcal{G}_k$ satisfying the conditions of Theorem 2.3 is extremely involved. In this section we give a complete description of a simplified version $\mathcal{G}'_k$ of the factory $\mathcal{G}_k$, thereby proving the the following theorem which is only slightly weaker than Theorem 2.3.

Theorem 8.1 *There is a green factory $\mathcal{G}'_k$ for $\tilde{\mathcal{S}}^k_k$ with $u_0, u_1 \sim 2.955$.*

As was the case with all the other factories we considered, the initial cost and production residues of $\mathcal{G}'_k$ are $O(k^2)$.

The constructs processed by the factory $\mathcal{G}'_k$ are shown in Fig. 12. They are singletons P_λ's, pairs P_0/P_1's, quartets P_{00}/P_{11}'s, chains of length three I_3's, and triples W_3/M_3's. The factory $\mathcal{G}'_k$ also processes compound constructs of the forms $(P_0, \gamma \times P_{00})$ and $(P_1, \gamma \times P_{11})$, where $\gamma \simeq 2.6603$. The credits attached to these basic and compound constructs are given in Table 6. Note that singletons have zero credit attached to them. Note also that quartets have a positive credit attached to them while the constructs $(P_0, \gamma \times P_{00})$ and $(P_1, \gamma \times P_{11})$ have zero credit attached to them. This reflects the fact that quartets can be more efficiently utilised if pairs are also supplied with them.

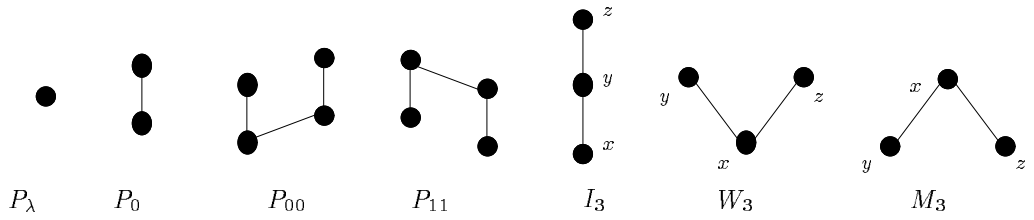

Figure 12: The basic input constructs of the factory $\mathcal{G}'_k$.

The factories $\mathcal{G}'_k$ are composed of the following four sub-factories:

1. $(P_0, \gamma \times P_{00})$ sub-factory.
2. $(P_1, \gamma \times P_{11})$ sub-factory.
3. P_{00} sub-factory.
4. P_{11} sub-factory.

The $(P_1, \gamma \times P_{11})$ sub-factory is the mirror image of the $(P_0, \gamma \times P_{00})$ sub-factory and the P_{11} sub-factory is the mirror image of the P_{00} sub-factory. The description of the $(P_0, \gamma \times P_{00})$ and the P_{00} sub-factories are given in the next two subsections.

As singletons (P_λ's) and pairs (P_0's) can be converted to quartets (P_{00}'s) at no cost, the $(P_0, \gamma \times P_{11})$ sub-factory may also receive singletons and pairs.

The constructs W_3, M_3 and I_3 cannot be utilised directly by the four sub-factories mentioned above. We therefore use the following simple process (carried out in a workshop?) to transform two W_3's into a P_0 and a P_{00}:

> Let $x_1, y_1 > z_1$ and $x_2, y_2 > z_2$ be two W_3's. Compare x_1 with x_2. Assume, without loss of generality, that $x_1 < x_2$. By removing the edge $x_2 > z_2$ we obtain the P_{00} x_1, y_1, z_1, x_2, where $z_2 < y_1$ and $z_2 < x_1 < x_2$, and the pair $z_2 < y_2$.

A single edge is cut in this process. As $\gamma > 1$, a combination of a P_0 and a P_{00} can be recycled without attaching any credit to it. Formally, this combination may be treated as $\frac{1}{\gamma} \times (P_0, \gamma \times P_{00})$ plus $(1 - \frac{1}{\gamma}) \times P_0$ and both these construct carry zero credit. The credits attached to two W_3's should therefore pay for cutting a single edge. This is exactly the case as $\mathit{credit}(W_3) = 0.5$.

Similarly, we can transform two M_3's into a P_1 and a P_{11}. An I_3 may be treated as either a W_3 or a M_3.

The factory $\mathcal{G}'_k$ described in this section can be improved if instead of using the above process for converting W_3's, M_3's and I_3's into P_0's and P_{00}'s or P_{11}'s, we used sub-factories that could directly consume these constructs. The improved factory $\mathcal{G}_k$ (whose construction is sketched in the next section) includes, along with other additions and changes, a separate I_3 sub-factory. The constructs I_3's turn out to be excellent raw materials for the construction of $\tilde{S}_k^k$'s. Each I_3 carries, in $\mathcal{G}_k$, a credit of -0.4213 units and the presence of I_3's among the recycled elements becomes a boon, and not a burden as in $\mathcal{G}'_k$.

We now describe the $(P_0, \gamma \times P_{00})$ sub-factory and the P_{00} sub-factory used by the $\mathcal{G}'_k$ factory.

8.1 The $(P_0, \gamma \times P_{00})$ sub-factory

The input to this sub-factory consists of costructs of the form $(P_0, \gamma \times P_{00})$, where $\gamma \simeq 2.6603$, with zero credit attached to each one of them. As singletons (P_λ's) and pairs (P_0's) can be converted to quartets (P_{00}'s) at no cost, the sub-factory may also receive singletons and pairs with no credit attached to them. Equivalently, we can say that the input to this sub-factory consists of singletons (P_λ's), pairs (P_0's) and quartets (P_{00}'s) with no credit attached to them, subject to the condition that each quartet is accompanied by at least $2/\gamma$ elements in singletons and/or pairs. The parameter γ is chosen to optimise the performence of the factory.

The $(P_0, \gamma \times P_{00})$ sub-factory can also receive quartets with no accompanying singletons or pairs, provided that each such quartet carries a credit of $1/(1 + 2\gamma)$. Such a credit can pay for breaking a quartet P_{00} into $\frac{2}{2\gamma+1} \times (P_0, \gamma \times P_{00})$ which can be accepted with no credit. In the $\mathcal{G}'_k$ factory, $0.1093 \simeq \mathit{credit}(P_{00}) < 1/(1 + 2\gamma) \simeq 0.1582$. It is therefore cheaper to feed quartets without accompanying singletons or pairs into the P_{00} sub-factory described in the next subsection.

Quartets fed into the sub-factory are used for the construction of a hyper-product which, in this case, is simply a hyperpair. Quartets are also grafted using the balanced quartet grafting process described in Subsection 7.2. Pairs are grafted using the recursive pair grafting process described in Subsection 7.1.

8.1.1 The hyper-product generation process of the $(P_0, \gamma \times P_{00})$ sub-factory

The hyper-products used in the $(P_0, \gamma \times P_{00})$ sub-factory are simply the balanced hyperpairs constructed according to the string $\mathcal{W} = 01(10)^\omega$. As P_{01}'s and P_{00}'s are in fact the same partial order (just the centre

is different), P_{00}'s can be used for the construction of such hyperpairs. The upper and lower element pruning costs of these hyperpairs are $pr_1(\mathcal{W}) = 1.5$ and $pr_0(\mathcal{W}) = 1.5$.

When r elements below the centre of a hyperpair H_i are prunned, the hyperpair H_i is broken into a collection of smaller hyperpairs. This collection includes r singletons (P_λ's). All the other remaining hyperpairs are of size at least eight. A similar thing happens when r elements above the centre of a hyperpair H_i are pruned. This time the collection of smaller hyperpairs obtained includes $\lfloor r/2 \rfloor$ pairs (P_0's). All the other hyperpairs are of size at least four. In both cases, when r elements are pruned, about r elements are obtained in singletons or pairs while all the other elements are conatined in hyperpairs of size at least four. The hyperpairs of size at least four are immediately used for the construction of the next H_i hyperpairs. The singletons and pairs are too valuable to be used for this purpose. They are recycled. Their recycling enables the recycling of quartetes, fed to the sub-factory as parts of $(P_0, \gamma \times P_{00})$'s constructs, without the necessity of attaching credits to them.

If $r = \sum_{i=0}^{\ell} r_i$, where the r_i's are distinct powers of two, then pruning r elements above the centre c of a hyperpair H_i results in a collection of hyperpairs $P_{0^{r_1}}, P_{0^{r_2}}, \ldots, P_{0^{r_\ell}}$ all whose centres are above c. Similarly, pruning r elements below the centre c of H_i results in a collection of hyperpairs $P_{1^{r_1}}, P_{1^{r_2}}, \ldots, P_{1^{r_\ell}}$ all whose centres are below c. All but at most three of such r pruned elements are contained in quartets (we can also assume, if convenient, that r is divisible by four in which case all the pruned elements are contained in quartets). All but at most seven of the pruned elements are contained in octets. The sub-factories of the factory $\mathcal{G}'_k$ are not capable of utilising octets. When pruned elements are recycled in the sub-factories of the factory $\mathcal{G}'_k$, we have to break them, therefore, into a collection of quartets at a cost of 1/4-th of a comparison per element. Some of the sub-factories of the factory $\mathcal{G}_k$, described in the next section, are capable of utilising 16-tuples (P_{0000}'s and P_{1111}'s). In the sub-factories of $\mathcal{G}_k$, it is therefore enough to break recycled pruned elements into P_{0000}'s or P_{1111}'s at a cost of 1/16-th of a comparison per element. In both cases, when these quartets or 16-tuples are recycled, an appropriate amount of credit should be attached to them.

As was already explained, it is convenient, for accounting purposes, to consider pruning as a special grafting process. It follows from the discussion above that we can consider the pruning process of the $(P_0, \gamma \times P_{00})$ sub-factory as a grafting process with the characteristics described in Table 7. The pruning processes used in other sub-factories will have other characteristics. We denote a downward pruned element by PR_0 and an upwards pruned element by PR_1. This notation is used in all sub-factories.

Outcome	Above	Below	Leftovers	gen	rec_0	rec_1
PR_0	—	$\frac{1}{4} \times P_{11}$	P_λ	1.5	0.25	1
PR_1	$\frac{1}{4} \times P_{00}$	—	$\frac{1}{2} \times P_0$	1.5	1	0.25

Table 7: The costs of the pruning process of the $(P_0, \gamma \times P_{00})$ sub-factory, viewed as a grafting process.

8.1.2 The output combinations of the $(P_0, \gamma \times P_{00})$ sub-factory

The list of the output combinations used by the $(P_0, \gamma \times P_{00})$ sub-factory is given in Table 8. Each combination involves just two outcomes. The exact proportion of the two outcomes in each combination is chosen so that the lower and upper element costs of the combination become equal, thereby minimising their maximum.

Before analysing the costs of these combinations, we verify that if enough outcomes of each of the three grafting processes used by the sub-factory are available, then at least one output combination can indeed be used. Suppose that at least one outcome of each of the two recursive pair grafting processes (i.e., at least

	Combination	Lower and Upper Element Cost
1	$(\ 1.\times Q_2^{1,1} \ , \ \ 0.4639\times PR_0 \ )$	2.9059
2	$(\ 1.\times U_0^2 \ \ , \ \ 2.1995\times PR_0 \ )$	<u>2.9546</u>
3	$(\ 1.\times L_2^0 \ \ , \ \ 2.1995\times PR_1 \ )$	<u>2.9546</u>
4	$(\ 1.\times L_2^{1,1} \ , \ \ 0.2269\times Q_3^1 \ \ )$	2.9538
5	$(\ 1.\times L_2^6 \ \ , \ \ 2.4546\times Q_3^1 \ \ )$	2.9499
6	$(\ 1.\times L_0^4 \ \ , \ \ 2.0254\times Q_3^1 \ \ )$	2.9001
7	$(\ 1.\times L_2^{1,1} \ , \ \ 0.1134\times Q_4^0 \ \ )$	<u>2.9546</u>
8	$(\ 1.\times L_2^6 \ \ , \ \ 1.2271\times Q_4^0 \ \ )$	2.9517
9	$(\ 1.\times L_0^4 \ \ , \ \ 1.0000\times Q_4^0 \ \ )$	2.8954
10	$(\ 1.\times U_{1,1}^2 \ , \ \ 0.2038\times Q_0^2 \ \ )$	2.9538
11	$(\ 1.\times U_6^2 \ \ , \ \ 2.2042\times Q_0^2 \ \ )$	2.9483
12	$(\ 1.\times U_4^0 \ \ , \ \ 1.8150\times Q_0^2 \ \ )$	2.9075

Table 8: The output combinations of the $(P_0, \gamma\times P_{00})$ sub-factory.

one outcome of $U_0^2, U_{1,1}^2, U_6^2$ and U_4^0 and at least one outcome of $L_2^0, L_2^{1,1}, L_2^6$ and L_0^4) is available and that at least nine outcomes of the P_{00}-grafting process (i.e., at least nine outcomes out of $Q_2^{1,1}, Q_3^1, Q_4^0$ and Q_0^2) are available. Each of the outcomes U_0^2, L_2^0 and $Q_2^{1,1}$ can immediately be used in conjunction with prunned elements (combinations 1,2 or 3), as prunned elements are always available. If none of these outcomes is available then we have at least three outcomes of either Q_3^1, Q_4^0 or Q_0^2, at least one of $U_{1,1}^2, U_6^2$ and U_4^0, and at least one of $L_2^{1,1}, L_2^6$ and L_0^4. If three Q_3^1's are available we can therefore activate one of combinations 4,5 or 6. If three Q_4^0's are available (actually two are enough here), we can activate one of combinations 7,8 or 9. Finally, if three Q_4^0's are available, we can activate one of combinations 10,11 or 12.

We now turn to the cost analysis of these combinations. We perform explicitly the computations for combinations 2 and 7. The other computations are similar.

- $(1.\times U_0^2, 2.1995\times PR_0)$:

The number $r \simeq 2.1995$ of pruned elements used in conjunction with U_0^2 is chosen so that the lower and upper element costs of the combination would become equal. We demostrate the computation that leads to this optimal choice of r.

Remembering that the input to the $(P_0, \gamma\times P_{00})$ sub-factory consists of $(P_0, \gamma\times P_{00})$'s, we get that the generation of an U_0^2 consumes one pair P_0 and leaves $\gamma\times P_{00}$ as leftovers. Similarly, the generation of $r\times PR_0$ consumes $\frac{r}{2}\times P_{00}$ and leaves $r\times P_\lambda$ (consult Table 7) and $\frac{r}{2\gamma}\times P_0$ as leftovers. The leftover singletons are immediately joined into pairs. The leftovers from the two processes are therefore $\frac{r}{2}(1+\frac{1}{\gamma})\times P_0$ and $\gamma\times P_{00}$. Provided that $\frac{r}{2}(1+\frac{1}{\gamma}) \geq 1$, these leftovers can be recycled without having to attach any credit to them.

The combination $(1.\times U_0^2, r\times PR_0)$ is composed of a P_0 above the centre and $\frac{r}{4}\times P_{11}$ below the centre. As

no credit is attached to the inputs, nor paid for the leftovers, the local lower and upper element costs of this combinarions are

$$u_0(C_2) = \frac{1.\times(gen(U_0^2)+rec_1(U_0^2))+r\times(gen(PR_0)+rec_1(PR_0))+credit(P_0)}{1.\times n_0(U_0^2)+r\times n_0(PR_0)},$$

$$u_1(C_2) = \frac{1.\times(gen(U_0^2)+rec_0(U_0^2))+r\times(gen(PR_0)+rec_0(PR_0))+\frac{r}{4}\times credit(P_{11})}{1.\times n_1(U_0^2)+r\times n_1(PR_0)}.$$

Consulting Table 2 we get that $gen(U_0^2)=0, rec_0(U_0^2)=2, rec_1(U_0^2)=1$ and of course that $n_0(U_0^2)=0$ and $n_1(U_0^2)=2$. Consulting Table 7 we get that $gen(PR_0)=1.5, rec_0(PR_0)=0.25, rec_1(PR_0)=1$ and of course that $n_0(PR_0)=1$ and $n_1(PR_0)=0$. We also have $credit(P_{11})=0.1093$ and $credit(P_0)=0$. Substituting these values into the above expressions and equating the two costs we obtain the equation

$$\frac{1.\times(0+1)+r\times(1.5+1)}{1.\times 0+r\times 1}=\frac{1.\times(0+2)+r\times(1.5+0.25)+\frac{r}{4}\times 0.1093}{1.\times 2+r\times 0}.$$

It is easy to check that the solution of this equation is $r\simeq 2.1995$ and the values of both the lower and upper element costs in this case are $u_0(C_2)=u_1(C_2)\simeq 2.9546$.

- $(1.\times L_2^{1,1}, 0.1134\times Q_4^0)$:

We again compute the lower and upper element costs of a combination $(1.\times L_2^{1,1}, r\times Q_4^0)$ and choose r so that both these costs would become equal. The generation of an $L_2^{1,1}$ consumes $2\times P_0$ and leaves $2\gamma\times P_{00}$ as leftovers. The generation of $r\times Q_4^0$ consumes $r\times P_{00}$ and leaves $\frac{r}{\gamma}\times P_0$ as leftovers. The proportion of quartets among the leftovers is much higher than in the previous case and they cannot all be recycled by pairing them with pairs. The $\frac{r}{\gamma}\times P_0$ can be paired with $r\times P_{00}$ of the leftovers to form $\frac{r}{\gamma}\times(P_0,\gamma\times P_{00})$ which can be recycled without credit. Credit however should be attached to the remaining $(2\gamma-r)\times P_{00}$ (assuming, as will be the case, that $2\gamma-r>0$).

The combination $(1.\times L_2^{1,1}, r\times Q_4^0)$ is composed of $2\times P_\lambda$ above the centre and a P_0 and a $r\times P_{00}$ below the centre. The lower part is recycled to the $(P_0,\gamma\times P_{00})$ sub-factory with no attached credit, as $r<\gamma$. The local lower and upper element costs of this combination are therefore

$$u_0(C_7) = \frac{1.\times(gen(L_2^{1,1})+rec_1(L_2^{1,1}))+r\times(gen(Q_4^0)+rec_1(Q_4^0))+(2\gamma-r)\times credit(P_{00})}{1.\times n_0(L_2^{1,1})+r\times n_0(Q_4^0)},$$

$$u_1(C_7) = \frac{1.\times(gen(L_2^{1,1})+rec_0(L_2^{1,1}))+r\times(gen(Q_4^0)+rec_0(Q_4^0))+(2\gamma-r)\times credit(P_{00})}{1.\times n_1(L_2^{1,1})+r\times n_1(Q_4^0)}.$$

Substituting the values $gen(L_2^{1,1})=2, rec_0(L_2^{1,1})=3, rec_1(L_2^{1,1})=4$ and $n_0(L_2^{1,1})=n_1(L_2^{1,1})=2$, found in Table 2, and the values $gen(Q_4^0)=1, rec_0(Q_4^0)=2, rec_1(Q_4^0)=5$ and $n_0(Q_4^0)=4, n_1(Q_4^0)=0$, found in Table 3, together with the value $credit(P_{00})=0.1093$ and equating these costs, we obtain the following equation

$$\frac{1.\times(2+4)+r\times(1+5)+(2\cdot 2.6603-r)\cdot 0.1093}{1.\times 2+r\times 4}=\frac{1.\times(2+3)+r\times(1+2)+(2\cdot 2.6603-r)\cdot 0.1093}{r\times 0+1.\times 2}.$$

It is easy to check that the solution of this equation is $r\simeq 0.1134$ and the values of both the lower and upper element costs in this case are $u_0(C_7)=u_1(C_7)\simeq 2.9546$.

The analysis of the other ten cases is similar. Combinations 2,3 and 7 turn out to be the worst combinations of this sub-factory.

8.2 The P_{00} sub-factory

The input to the P_{00} sub-factory consists of P_{00}'s. Each P_{00} fed to the sub-factory carries a credit of $credit(P_{00}) \simeq 0.1093$. The sub-factory constructs hyper-products using the process described below The sub-facroty uses the extended balanced and the unbalanced P_{00} grafting processes described in Subsections 7.3 and 7.4.

8.2.1 The hyper-product generation process of the P_{00} sub-factory

The hyper-products used by the P_{00} sub-factory are generated according to an infinite string $\mathcal{W}' = 011\mathcal{W}''$ where $\mathcal{W}''$ is an infinite string with $pr_0(\mathcal{W}'') \simeq 1.7465$ and $pr_1(\mathcal{W}'') \simeq 1.2535$. Such a string exists according to Theorem 4.6. According to Lemma 4.4 we get that $pr_0(\mathcal{W}') = \frac{1}{4}pr_0(\mathcal{W}'') + 1 \simeq 1.4366$ and that $pr_1(\mathcal{W}') = \frac{1}{2}pr_1(\mathcal{W}'') + 1 \simeq 1.6268$. As the string $\mathcal{W}'$ begins with 01, the hyperpairs constructed according to it are indeed hyper-products of P_{00}'s. We let H'_i denote the hyper-product generated according the prefix of length i of $\mathcal{W}'$.

The generation of the hyper-products H'_i consumes P_{00}'s. Each such P_{00} carries a credit of $credit(P_{00}) = 0.1093$. As we have seen in the $(P_0, \gamma \times P_{00})$ sub-factory, the pruning process may be viewed as a pruning process that receives $\frac{1}{2} \times P_{00}$ and returns either $\frac{1}{4} \times P_{11}$ below the centre, or $\frac{1}{4} \times P_{00}$ above the centre, as well as a leftover of $\frac{1}{2} \times P_0$ (or a P_λ which can be converted into $\frac{1}{2} \times P_0$ at no cost). The leftover $\frac{1}{2} \times P_0$ can be joined with $\frac{\gamma}{2} \times P_{00}$ from the input stream to form $\frac{1}{2} \times (P_0, \gamma \times P_{00})$. Such constructs can be recycled without any credit attached to them. The cost of pruning an element can therefore be 'subsidised' by the credit attached to $\frac{1}{2}(1+\gamma) \times P_{00}$.

Outcome	Above	Below	gen	rec_0	rec_1
PR_0	—	$\frac{1}{4} \times P_{11}$	1.4366	0.25	1
PR_1	$\frac{1}{4} \times P_{00}$	—	1.6268	1	0.25

Table 9: The costs of the pruning process of the P_{00} sub-factory, viewed as a grafting process.

8.2.2 The output combinations of the P_{00} sub-factory

The list of the output combinations used by the P_{00} sub-factory is given in Table 10. This list includes all the combinations that involve the outcomes Q_0^4, Q_2^6 and Q_6^{10} of the extended balanced P_{00} grafting and the outcomes R_4^0, R_3^1, R_6^2, R_{10}^6 and R_4^4 of the unbalanced P_{00} grafting. As can be seen, each of these outcomes can be used in conjunction with pruned elements to obtain low local lower and upper element costs. The balanced grafting process may also produce outcomes $Q_2^{1,1}$, Q_3^1 and Q_4^0, and the unbalaced grafting process may also produce outcomes $L_2^{1,1}$, L_2^6 and L_0^4. These outcomes are combined using the combinations of the $(P_0, \gamma \times P_{00})$ sub-factory given in Table 8. It is easy to verify that when each of the two grafting processes is applied a sufficient number of times, at least one output combination is applicable.

The costs of all the combinations involving $Q_2^{1,1}$, Q_3^1 and Q_4^0 and $L_2^{1,1}$, L_2^6 and L_0^4 are strictly smaller than their costs in the $(P_0, \gamma \times P_{00})$ sub-factory. This is due to the fact that the input constructs to the P_{00} sub-factory carry (positive) credits while the inputs to the $(P_0, \gamma \times P_{00})$ sub-factory do not, and due to the fact that the effective pruning costs $pr'_0(\mathcal{W}') \simeq 1.4366 - \frac{1+\gamma}{2}\, credit(P_{00}) \simeq 1.2366$ and $pr'_1(\mathcal{W}') \simeq 1.6268 - \frac{1+\gamma}{2}\, credit(P_{00}) \simeq 1.4267$ in the P_{00} sub-factory are lower than the pruning costs $pr_0(\mathcal{W}) = pr_1(\mathcal{W}) = 1.5$ of the $(P_0, \gamma \times P_{00})$ sub-factory.

It is therefore enough to analyse the costs of the combinations listed in Table 10. We present, as examples, the analyses of combinations 3 and 7 which, together with combinations 2,3 and 7 of the $(P_0, \gamma \times P_{00})$

	Combination	Lower and Upper Element Cost
1	$(\ 1.\times R_4^0 \ , \ 4.0000\times PR_1 \ )$	2.9267
2	$(\ 1.\times R_3^1 \ , \ 2.0000\times PR_1 \ )$	2.7481
3	$(\ 1.\times R_6^2 \ , \ 3.5464\times PR_1 \ )$	<u>2.9546</u>
4	$(\ 1.\times R_{10}^6 \ , \ 3.5456\times PR_1 \ )$	2.9508
5	$(\ 1.\times R_4^4 \ , \ 0.4096\times PR_1 \ )$	2.8972
6	$(\ 1.\times Q_0^4 \ , \ 4.0000\times PR_0 \ )$	2.7366
7	$(\ 1.\times Q_2^6 \ , \ 4.0000\times PR_0 \ )$	<u>2.9546</u>
8	$(\ 1.\times Q_6^{10} \ , \ 4.0000\times PR_0 \ )$	2.9509
9	$(\ 1.\times Q_4^4 \ , \ 0.4131\times PR_0 \ )$	2.8790

Additional combinations involve $Q_2^{1,1}$, Q_3^1, Q_4^0, $L_2^{1,1}$, L_2^6, L_0^4 and pruning

Table 10: The output combinations of the P_{00} sub-factory.

sub-factory, determine the worst-case behaviour of the whole factory. The analysis of all the other cases is similar.

- $(1.\times R_6^2, 3.5464\times PR_1)$:

The local lower and upper element costs of the combination $(1.\times R_6^2, r\times PR_1)$ are

$$u_0(C_3) = \frac{1.\times(gen(R_6^2)+rec_1(R_6^2))+r\times(gen(PR_1)+rec_1(PR_1))-(2+\frac{r}{2}(1+\gamma))\,credit(P_{00})}{1.\times n_0(R_6^2)+r\times n_0(PR_1)},$$

$$u_1(C_3) = \frac{1.\times(gen(R_6^2)+rec_0(R_6^2))+r\times(gen(PR_1)+rec_0(PR_1))-(2+\frac{r}{2}(1+\gamma))\,credit(P_{00})}{1.\times n_1(R_6^2)+r\times n_0(PR_1)},$$

No credits should be attached to recycled elements, assuming that $\frac{r}{4} \leq \gamma$, as the proportion of quartets among the recycled elements is low enough. Solving the equation

$$\frac{1.\times(3+9)+r\times(1.6268+0.25)-(2+\frac{r}{2}(1+2.6603))\cdot 0.1093}{1.\times 6+r\times 0} = \frac{1.\times(3+5)+r\times(1.6268+1)-(2+\frac{r}{2}(1+2.6603))\cdot 0.1093}{1.\times 2+r\times 1}$$

we get thet $r \simeq 3.5464$ and $u_0(C_3) = u_1(C_3) \simeq 2.9546$.

- $(1.\times Q_2^6, 4.0000\times PR_0)$:

The local lower and upper element costs of the combination $(1.\times Q_2^6, r\times PR_0)$ are

$$u_0(C_7) = \frac{1.\times(gen(Q_2^6)+rec_1(Q_2^6))+r\times(gen(PR_0)+rec_1(PR_0))-(2+\frac{r}{2}(1+\gamma))\,credit(P_{00})}{1.\times n_0(Q_2^6)+r\times n_0(PR_0)},$$

$$u_1(C_7) = \frac{1.\times(gen(Q_2^6)+rec_0(Q_2^6))+r\times(gen(PR_0)+rec_0(PR_0))-(2+\frac{r}{2}(1+\gamma))\,credit(P_{00})}{1.\times n_1(Q_2^6)+r\times n_0(PR_0)}.$$

Again, no credits should be attached to recycled elements, assuming that $\frac{r}{4} \leq \gamma$. Solving the equation

$$\frac{1.\times(4+5)+r\times(1.4366+1)-(2+\frac{r}{2}(1+2.6603))\cdot 0.1093}{1.\times 2+r\times 1} = \frac{1.\times(4+8)+r\times(1.4366+0.25)-(2+\frac{r}{2}(1+2.6603))\cdot 0.1093}{1.\times 6+r\times 0}$$

we get thet $r \simeq 4.0000$ and $u_0(C_7) = u_1(C_7) \simeq 2.9546$.

This completes the description and the analysis of the factory $\mathcal{G}'_k$.

9 A factory with $u_0, u_1 \sim 2.942$

In this section we sketch the construction of factories $\mathcal{G}_k$ with lower and upper element costs $u_0, u_1 \sim 2.942$ whose existence was claimed in Theorem 2.3. A complete description of the factories $\mathcal{G}_k$ could be found in [Dor95].

The general structure of the factories $\mathcal{G}_k$ is similar to the structure of the factories $\mathcal{G}'_k$ described in the previous section. The $\mathcal{G}_k$ factories recycle however many more constructs. The structures recycled by the factories G_k are the structures shown in Fig. 13 and their mirror images (compare this to the much smaller set of constructs recycled by the factories $\mathcal{G}'_k$, shown in Fig. 12).

The factory $\mathcal{G}_k$ is composed of thirteen sub-factories: a $(P_0, \gamma\times P_{00})$ sub-factory, a $(P_1, \gamma\times P_{11})$ sub-factory, a P_{00} sub-factory, a P_{11} sub-factory, a P_{000} sub-factory, a P_{111} sub-factory, a P_{0000} sub-factory, a P_{1111} sub-factory, a $(P_\lambda, P_0, \beta \times P_{000})$ sub-factory, a $(P_\lambda, P_1, \beta \times P_{111})$ sub-factory, a I_3 sub-factory, a I_4 sub-factory and a I_5 sub-factory, where $\beta \simeq 1.6500$ and $\gamma \simeq 2.0500$.

The credits attached to the basic and compound constracts are $credit(P_\lambda) = 0$, $credit(P_0) = 0$, $credit(P_0, \gamma\times P_{00}) = 0$, $credit(P_{00}) = 0.1330$, $credit(P_{000}) = 0.8630$, $credit(P_{0000}) = 2.4847$, $credit(I_3) = -0.4213$, $credit(I_4) = -0.9230$, $credit(I_5) = -1.2000$, $credit(W_3) = 0.5000$ and $credit(P_\lambda, P_0, \beta\times P_{000}) = 1.2259$. The credit attached to a construct is equal to the credit attached to its mirror image.

The first four sub-factories employed by $\mathcal{G}_k$ are essentially identical to the corresponding sub-factories used by $\mathcal{G}'_k$. The main difference is that different parameters are used and that pruned elements are recycled as 16-tuples (i.e., P_{0000}'s or P_{1111}'s) and not as quartets. The value of the parameter γ is decreased to $\gamma \simeq 2.0500$. The credit attached to a quartet is increased to $credit(P_{00}) \simeq 0.1330$. These changes change the costs of these sub-factories but the analysis is very similar to the one carried out in the previous section. The other nine sub-factories are new.

As can be seen, there is no W_3 sub-factory. As before, a workshop is used to convert two W_3's into a P_0 and a P_{00} at the price of cutting a single edge.

The second output combination of the $(P_0, \gamma \times P_{00})$ sub-factory is again one of the worst cases of the factories $\mathcal{G}_k$. We describe the analysis of this case and compare it to the analysis of the same case in the $\mathcal{G}'_k$ factories.

- $(1.\times U_0^2, 2.2613\times PR_0)$:

The local lower and upper element costs of the combination $(1.\times U_0^2, r\times PR_0)$ are

$$u_0(C_2) = \frac{1.\times(gen(U_0^2)+rec_1(U_0^2))+r\times(gen(PR_0)+rec_1(PR_0))+credit(P_0)}{1.\times n_0(U_0^2)+r\times n_0(PR_0)},$$

$$u_1(C_2) = \frac{1.\times(gen(U_0^2)+rec_0(U_0^2))+r\times(gen(PR_0)+rec_0(PR_0))+\frac{r}{16}\times credit(P_{0000})}{1.\times n_1(U_0^2)+r\times n_1(PR_0)}.$$

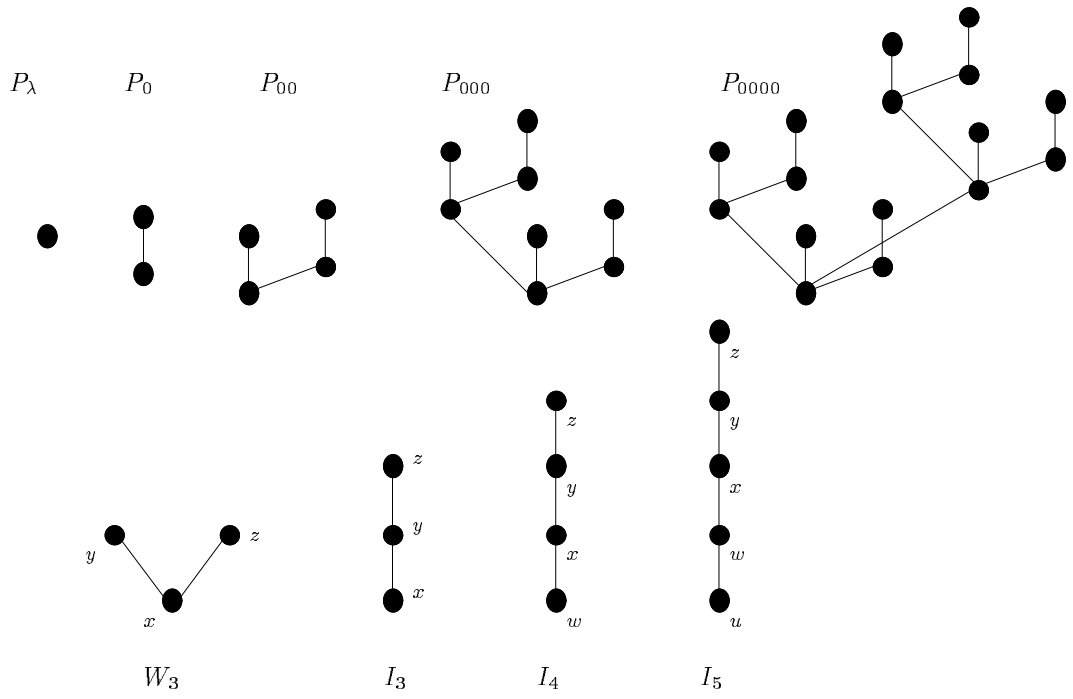

Figure 13: The basic input constructs of the factory $\mathcal{G}_k$.

The expression for $u_0(C_2)$ is identical to the expression for $u_0(C_2)$ in $\mathcal{G}'_k$. The expression for $u_1(C_2)$ differs from the corresponding expression in $\mathcal{G}'_k$ as prunned elements are now recycled as P_{1111}'s. The recycling costs of prunned element are now $rec_0(PR_0) = 0.0625$ and $rec_1(PR_0) = 1$. Each recycled P_{1111} should carry a credit of $credit(P_{0000}) \simeq 2.4847$. Substituting these updated values into the above expressions and equating the two costs we obtain the equation

$$\frac{1.\times(0+1)+r\times(1.5+1)}{1.\times 0+r\times 1} = \frac{1.\times(0+2)+r\times(1.5+0.0625)+\frac{r}{16}\times 2.4847}{1.\times 2+r\times 0}\,.$$

It is easy to check that the solution of this equation is $r \simeq 2.2613$ and the values of both the lower and upper element costs in this case are $u_0(C_2) = u_1(C_2) \simeq 2.9422$.

We believe that it is possible to obtain further *small* improvements by recycling more and yet larger constructs and by designing a new sub-factory for each such construct or combination of constructs. The $(P_0, \gamma\times P_{00})$ sub-factory serves as a keystone in all our factories and in all such possible extensions. Recall that one of the worst cases of the $(P_0, \gamma\times P_{00})$ sub-factory is the combination of U_0^2 and pruning. The lower part of an output partial order generated using this combination is essentially a hyperpair of the form $P_{1^{k'}}$, where $k \le k' \le 2k$. It is not hard to verify that even if we could recycle such a large hyperpair without attaching any credit to it, the lower and upper element costs of this combination would still be about $u_0(C_2) = u_1(C_2) \simeq 2.895$. It seems, therefore, that some new ideas are required to obtain a major improvement to our median selection algorithm.

10 Concluding remarks and open problems

We have improved the result of Schönhage, Paterson and Pippenger [SPP76] and obtained an algorithm for the selection of the median that uses slightly less than $3n$ comparisons. Our algorithm is much more

complicated than the algorithm of Schönhage *et al.* and it is perhaps a bit disappointing that the improvement obtained is so small. As mentioned at the end of the previous section, further small improvements are possible but it seems that new ideas are required to obtain a more substantial improvement.

Further narrowing the gap between the known upper and lower bounds on the number of comparisons needed to select the median remains a challenging open problem. Mike Paterson (personal communication) conjectures that the number of comparisons required for selecting the median, in the worst case, is about $\log_{4/3} 2 \cdot n \approx 2.41n$.

Schönhage, Paterson and Pippenger [SPP76] show that a conjecture of Yao [Yao74] implies the existence of a median finding algorithm that uses at most $2.5n + o(n)$ comparisons. Proving or disproving Yao's conjecture is also a challenging open problem.

The factories constructed in this paper are $\tilde{\mathcal{S}}_k^k$ factories and not S_k^k factories, as were the factories of Schönhage, Paterson and Pippenger. Is it possible to improve the S_k^k factories of Schönhage *et al.* and obtain S_k^k factories whose lower and upper element costs are below 3?

Schönhage *et al.* [SPP76] describe non-green S_k^k factories with unit cost $U_k \sim 3.5k$. Is it possible to improve this result?

Acknowledgement

The authors would like to thank Mike Paterson for some helpful discussions and for his comments on an earlier version of this paper.

References

[Aig82] M. Aigner. Selecting the top three elements. *Discrete Applied Mathematics*, 4:247–267, 1982.

[BFP+73] M. Blum, R.W. Floyd, V. Pratt, R.L. Rivest, and R.E. Tarjan. Time bounds for selection. *Journal of Computer and System Sciences*, 7:448–461, 1973.

[BJ85] S.W. Bent and J.W. John. Finding the median requires $2n$ comparisons. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, Providence, Rhode Island*, pages 213–216, 1985.

[CM89] W. Cunto and J.I. Munro. Average case selection. *Journal of the ACM*, 36(2):270–279, April 1989.

[Dor95] D. Dor. *Selection algorithms.* PhD thesis, Department of Computer Science, Tel Aviv University, 1995.

[DZ95a] D. Dor and U. Zwick. Finding percentile elements. In *Proceedings of the 3rd Israel Symposium on Theory and Computing systems, Tel Aviv, Israel*, pages 88–97, 1995. Journal version to appear in *Combinatorica*.

[DZ95b] D. Dor and U. Zwick. Selecting the median. In *Proceedings of the 6rd Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, California*, pages 28–37, 1995.

[Eus93] J. Eusterbrock. Errata to "Selecting the top three elements" by M. Aigner: A result of a computer-assisted proof search. *Discrete Applied Mathematics*, 41:131–137, 1993.

[FG78] F. Fussenegger and H.N. Gabow. A counting approach to lower bounds for selection problems. *Journal of the ACM*, 26(2):227–238, April 1978.

[FJ59] L.R. Ford and S.M. Johnson. A tournament problem. *American Mathematical Monthly*, 66:387–389, 1959.

[FR75] R.W. Floyd and R.L. Rivest. Expected time bounds for selection. *Communication of the ACM*, 18:165–173, 1975.

[HS69] A. Hadian and M. Sobel. Selecting the t-th largest using binary errorless comparisons. *Colloquia Mathematica Societatis János Bolyai*, 4:585–599, 1969.

[Hya76] L. Hyafil. Bounds for selection. *SIAM Journal on Computing*, 5:109–114, 1976.

[Joh88] J.W. John. A new lower bound for the set-partition problem. *SIAM Journal on Computing*, 17(4):640–647, August 1988.

[Kir81] D.G. Kirkpatrick. A unified lower bound for selection and set partitioning problems. *Journal of the ACM*, 28:150–165, 1981.

[Kis64] S.S. Kislitsyn. On the selection of the k-th element of an ordered set by pairwise comparisons. *Sibirsk. Mat. Zh.*, 5:557–564, 1964.

[MP82] I. Munro and P.V. Poblete. A lower bound for determining the median. Technical Report Research Report CS-82-21, University of Waterloo, 1982.

[Poh72] I. Pohl. A sorting problem and its complexity. *Communication of the ACM*, 15:462–464, 1972.

[RH84] P.V. Ramanan and L. Hyafil. New algorithms for selection. *Journal of Algorithms*, 5:557–578, 1984.

[Sch32] J. Schreier. On tournament elimination systems. *Mathesis Polska*, 7:154–160, 1932. (in Polish).

[SPP76] A. Schönhage, M. Paterson, and N. Pippenger. Finding the median. *Journal of Computer and System Sciences*, 13:184–199, 1976.

[SY80] P. Stockmeyer and F.F. Yao. On the optimality of linear merge. *SIAM Journal on Computing*, 9:85–90, 1980.

[Yao74] F. Yao. On lower bounds for selection problems. Technical Report MAC TR-121, Mass. Inst. of Technology, 1974.

[Yap76] C.K. Yap. New upper bounds for selection. *Communication of the ACM*, 19(9):501–508, September 1976.