GENERATING SUBSCRIPTIONS AND PUBLICATIONS

1. Executive Summary

Ideally we'd like to be able to chose any subset of subscribers and design a publication so that every subscriber in that set receives the publication and no other subscribers receive the publication. Unfortunately, Theorem 12 shows that this is only possible for a relatively small set of subscriptions. However, Section 7 outlines an algorithm that allows excellent diversity in selection of hosts receiving a subscription when the desired match rate is relatively low (around 10 or so). This procedure may not be optimal, but the theorems in Section 4 give us good reason to believe it is at least close to optimal. The outlines procedure is likely good enough for use in TA3 and the subscription and publication generation algorithms should be relatively easy to translate into code.

2. Assumptions

From https://wiki.llan.ll.mit.edu/display/GROUP68/TA3.1+Performer+Subscription+Support it appears that all performers support DNF queries. ACS and Argon support this directly, while BBN supports this by having multiple active subscriptions each of which is a conjunction of equalities. Thus, this document assumes all performers support DNF queries.

3. Definitions

We assume the meta-data consists of n attributes. An attribute is the description of a field like "name", "date of birth", or "address".

Definition 1. An *attribute equality* is a predicate that asserts that an attribute in the meta-data has a specified value. For example, "fname = 'Oliver"' is an attribute equality asserting that the value of the first name field is "Oliver". The attribute equality evaluates to true on a publication if and only if the specified field in the publication's meta-data has the specified value.

Definition 2. A *subscription* consists of a conjunction of attribute equalities. While some performers can support disjunctions or DNF formulas, in this document we will treat those as a subscriber that has multiple active subscriptions, each of which is the conjunction of attribute equalities.

Definition 3. Two subscriptions, s_i and s_j, are said to be *mutually independent* if it is possible to generate a publication that matches s_i but not s_j and it is possible to generate a publication that matches s_j but not s_i.

Definition 4. Two subscriptions, s_i and s_j, are said to be *mutually compatible* if it is possible to generate a publication, p, that matches both s_i and s_j.

Definition 5. A set of subscriptions, S, is set to be *perfectly compatible* if, for every subset $M \subseteq S$, it is possible to generate a publication, p, such that all the subscriptions in M match p, and none of the subscriptions in $S - M$ match p.

Definition 6. A *disjoint group of perfectly compatible subsets* is a set of sets $D = \{S_1, S_2, \ldots, S_{|D|}\}$ such that S_i is a perfectly compatible subset for all $S_i \in D$ and $s_{i,p} \in S_i$ and $s_{j,q} \in S_j$ guarantees that $s_{i,p}$ and $s_{j,q}$ are mutually independent if $i \neq j$.

Definition 7. The *target set* of a publication, p, is the set of subscriptions that match p.

Definition 8. If S is a set of subscriptions, $M \subseteq S, |M| = z$, and there exists a publication p such that M is the target set of p (i.e. all the subscriptions in M match p and none of the subscriptions in $S - M$ match p) then M is said to be a *viable target for z*.

4. Facts

Theorem 9. *If two subscriptions, s_i and s_j are members of a perfectly compatible set, S, then any attribute that appears in an attribute equality of both s_i and s_j must be assigned the same value in the attribute equality.*

Proof. If not, then s_i and s_j are not mutually compatible. □

Theorem 10. *If S is a perfectly compatible set of subscriptions, each of which is the conjunction of m attribute equality conditions, then no two subscriptions, s_i and s_j, in S can have equalities on more than $m - 1$ common attributes.*

Proof. Suppose not. Then there exists s_i and s_j in S such that s_i and s_j are equalities over the same set of m attributes (since we're assuming each subscription is a conjunction of m attribute equalities). Thus, by theorem 9, the values in each shared attribute equality must be the same. In other words, s_i and s_j must be the same subscription. However, this means that any publication that matches s_i will also match s_j violating the mutual independence requirement. □

Theorem 11. *If S is a perfectly compatible set of subscriptions, $M \subset S$ is any subset of S, and $\bar{s} \in (S - M)$ is any subscription in S but not in M, then $\bar{s}$ must have at least one equality condition that is not in any other subscription in M.*

Proof. Suppose not. Then there exits $\bar{s} \in (S - M)$ such that all of the attributes in $\bar{s}$ appear in at least one of the $s_i \in M$. By Theorem 9 all of the attributes in $\bar{s}$ and M must then have the same values. But this means that if a publication, p, matches all of the $s_i \in M$ then p must also match $\bar{s}$. However this contradicts the independence requirement of perfectly compatible sets so we have a contradiction. □

Theorem 12. *If all subscriptions in a set A are a conjunction of m equality conditions, then no* perfectly compatible subset *can contain more than $n - m + 1$ subscriptions.*

Proof. Let s_1, s_2, …, s_k denote the subscriptions in S. By supposition, s_1 contains m attributes. As per Theorem 10, any pair of queries in S can have at most $m - 1$ attributes in common, so s_2 must contain an attribute that is not present in s_1. Thus, the set $\{s_1, s_2\}$ contains at least $m + 1$ different attribute equalities. Similarly, by Theorem 11,s_3 must contain an attribute that is not present in s_1 or s_2, so the set $\{s_1, s_2, s_3\}$ must contain at least $m + 2$ different attribute equalities, and so on. Thus, if $|S| = k$, there must be at least $m + k - 1$ different attributes represented in the k subscriptions. But there are at most n possible attributes so $m + k - 1 \leq n$ which means $k \leq n - m + 1$. □

Theorem 13. *If S is a set of perfectly compatible subscriptions but each subscription can have a different number of attribute equalities, the maximum size of S is $n - m + 1$ where m is the number of attribute conjunctions in the subscription with the most attribute equalities.*

Proof. Virtually identical to the proof of 12. □

5. Subscription Generation Procedures

5.1. **Diversity Metrics.** If we want to generate publications that match $z \leq n$ subscribers then the primary challenge is figuring out to maximize "*diversity*". Here diversity has at least two meanings:

(1) Diversity of subscriptions: we'd like subscriptions to vary in the number of predicates in each conjunction and we'd like them to vary in the fields in each equality predicate.
(2) Diversity is matching subscribers: we'd like to be able to spread the matches out to subscribers in any way we want. Sometimes we'll want to send a long string of publications which match the same subscribers, sometimes we'll want to each publication to match a totally random subset of subscribers, and sometimes we'll want something in between. Each type of matching stresses different aspects of the tested system (e.g. repeated matching by a single subscriber stresses that subscriber and may cause it to become the bottleneck for the entire system, while matching different subscribers with each publication forces the broker and/or data store to manage the maximum number of concurrent connections). In any event, it seems desirable that we have maximal control here.

Definition (1) of diversity is, roughly speaking, almost free if we try to generate perfectly compatible subsets, so we turn our attention to maximizing definition (2) of diversity. Assume we have generated a set of subscriptions, S, according to some procedure. We would like to quantify the diversity afforded by this set. One way to quantify the diversity of by looking at the number of distinct subsets of subscriptions we could choose if our goal was to generate a publication that matches z subscriptions. If V_z is the set of all viable targets for z (i.e. $v_{z,i} \in V_z$ means that $v_{z,i}$ is a viable target for z). Then we define $\alpha(z) = |V_z|$. Thus $\alpha(z)$, which is a function of z, indicates how many unique subsets of subscriptions we could choose to be the target of a subscription of our choosing.

While $\alpha(z)$ is a decent metric, it does not distinguish between solutions that allow for multiple choices of fully disjoint subsets of subscriptions and those that don't. For example, we might find a subscription generation procedure that has a large $\alpha(z)$ but for which there exists some subscriptions that are common to all viable targets for z. To quantify the diversity in disjoint sets we define $U_{z,i}$ as a subset of V_z containing only disjoint viable target sets. Let U_z be the largest such subset. Then $\beta(z) = |U_z|$ has the desired properties.

6. Generating Disjoint Perfectly Compatible Subsets

A perfectly compatible subset offers maximal control but, as demonstrated above, we can only generate these for a small subset of subscribers. However, we have a few options:

- A single subscriber can be in more than one perfectly compatible subset. Thus, by carefully generating our subsets we can randomly choose our first host and still have a wide variety of subset to choose for the 2nd.
- We could generate a group of disjoint perfectly compatible subsets. To send a publication we'd randomly choose a perfectly compatible subset first and then choose the subscribers from that set.

In this section we explore the second option. The first option is explored later.

Algorithm 1 Procedure to generate the number of predicates in each subscription in a perfectly compatible subset.

Function: GetSubsetSubscriptionSizes(k, n, $G()$)
Inputs:

- k: the maximum number of subscriptions in the subset.
- n: The number of fields available for use in this subset.
- $G()$: A function which returns m_i, the number of predicates desired for the next subscription. $G()$ might be random or deterministic depending on user requirements.

Outputs: An array of integers indicating the size of each subscription. The number of integers is the maximum possible size for a the subset and is a function of the largest value returned by $G()$.

```
(1) $m_{next} \leftarrow G()$
(2) $m \leftarrow m_{next}$
(3) $R \leftarrow$the empty array
(4) while $n - m + 1 > |R|$ and $|R| < k$
    (a) Append $m_{next}$ to $R$
    (b) $m_{next} = G()$
    (c) $m = \max\{m, m_{next}\}$
(5) Sort $R$ in descending order
(6) Return $R$
```

6.1. **Subscription Generation Procedure.** Our goal is to generate h subscriptions. We would like to be able to group these subscriptions into disjoint perfectly compatible subsets. Additionally, we would like to be able to generate the size of each subscription as we go. This last requirement is a consequence of BAA rules (or at least a recent interpretation of them) that the number of clauses in each subscription have a Poisson distribution.

Let m_i be the number of clauses in subscription i. m_i might be deterministic, or it might be the output of a random number generator. A naive algorithm might start generate subscriptions in a subgroup by determining the next m_i, and then building a subscription with m_i predicates, ensuring that at least one attribute equality is on an attribute not yet present in any other subscription in the subgroup. However, this can fail. For example, suppose $m_1 = 1$ and the subscription s_1 =“fname = ’Oliver”’ is chosen for the 1st subscription. Then, for the next subscription we get $m_2 = 2$. We must ensure that at least one attribute for the 2nd subscription isn’t present in any other subscription in the subset so we choose “city = ’Eugene”’ for one attribute equality. In order the maximize the size of the subset it behooves us to choose the other attribute equality from the set that is in use so we choose “fname” and, since we’re still building a perfectly compatible subset we must assign that attribute the value ’Oliver’. This yields s_2 =”fname = ’Oliver” and city = ’Eugene”’. However, this violates the independence requirement as it is impossible to generate a publication that targets s_2 but not s_1.

Luckily the above issue is easily overcome if we generate all the subscription sizes in a single subset, sort them in descending size order, and then follow the general algorithm sketch above. Algorithm 1 outlines a procedure for generating the number of predicates in each subscription in a subset and maximizing the size of the subset. There is one flaw with Algorithm 1; the last value returned by $G()$ is discarded. If $G()$ is returning numbers according to some distribution this would skew the results in favor of smaller values. The solution is to return the last value generated by $G()$ and use that as the 1st m_{next} on the next call to the function (with minor adjustments should m_{next} be impossible). We also define a function, TrimN, which looks at all the subscriptions generated thus far and determines how many fields are available for future subscriptions. If a field can obtain γ unique values and $\gamma - 1$ of these have been used in a subscription, then that field is no longer available; we must reserve at least one value for publications that should not target any of the subscriptions generated thus far. The implementation of TrimN is not shown here but should be obvious enough.

We can use GetSubsetSubscriptionSizes, as defined in Algorithm 1, in our general subscription generation algorithm. This yields Algorithm 2.

6.2. **Publication Generation Procedure.** Decide on z, the number of subscribers that should match. Select a perfectly compatible subset $S_j \in C$ such that $|S_j| \geq z$. Select a random subset, M, from the subgroup. For each attribute/value pair represented in M choose those values for the meta-data. For each attribute not in M choose a value not in A.

6.3. **Analysis.** The subscription and publication generation procedures are relatively straight forward. How little or how much diversity does they provide?

When we go to generate a publication we first pick z. Then we must pick a $S_j \in C$, $|S_j| \geq z$. Assuming all S_j have the same m, there are $r = \text{floor}\left(h/\left(n - m + 1\right)\right)$ or $r = \text{ceil}\left(h/\left(n - m + 1\right)\right)$of these (depending on the size of the last subgroup generated). In each subgroup we have $t = n - m + 1$ subscriptions, except perhaps the last generated subgroup which may have fewer. Since each subgroup is perfectly compatible we can generate $\binom{z}{t}$ unique combinations of subscriptions. Thus, $\alpha\left(z\right) = r\binom{z}{t}$.

The maximum number of disjoint subgroups in any perfectly compatible subgroup is simply the number of times z divides t. Thus, the number of disjoint subgroups is given by $\beta\left(z\right) = r\text{floor}\left(t/z\right)$.

Algorithm 2 Subscription generation procedure.

```
Inputs:
    • $G()$: some function that, when called, returns the number of predicates that should be in the next subscription generated.
    • $h$: the total number of subscriptions to Generate
Outputs: $C$, a set of perfectly compatible subsets. Each subset in $C$ is independent of the others.
  (1) Let $A \leftarrow \{\}$ be the empty set. This will hold all the attribute/value pairs in any subscription generated in any subset.
  (2) Let $C \leftarrow \{\}$ be the empty set. This will be the set of perfectly compatible subsets.
  (3) Let $n \leftarrow$TrimN() be the number of fields available for subscriptions.
  (4) Let $j \leftarrow 0$ be the current subset index
  (5) Let $i \leftarrow 0$ be the current subscription index
  (6) While $|A| < h$
      (a) $R =$GetSubsetSubscriptionSizes($h - |A|$, $n$, $G()$)
      (b) $A_j \leftarrow \{\}$. This will hold the attribute/value pairs of all subscriptions in subset $j$
      (c) $S_j \leftarrow \{\}$. This will be perfectly compatible subset $j$.
      (d) for $m_{next}$ in $R$
          (i) $s_i \leftarrow \{\}$
          (ii) Select $\min(|A_j|, m_{next} - 1)$ attribute/value pairs from $A_j$ and add them to $s_i$.
          (iii) Randomly select $m_{next} - \min(|A_j|, m_{next} - 1)$ attributes that aren't yet in $A_j$ and assign them a value that isn't in $A$. Add these new attribute/value pairs to $s_i$ and $A_j$.
          (iv) Add $s_i$ to $S_j$.
          (v) $i \leftarrow i + 1$
      (e) Append $S_j$ to $C$
      (f) $j \leftarrow j + 1$
      (g) $A \leftarrow A \cup A_j$
      (h) $n \leftarrow$TrimN()
```

z	$\alpha(z)$	Optimal $\alpha(z)$	$\beta(z)$	Optimal $\beta(z)$
1	96	100	96	100
2	336	4950	48	50
3	672	161700	24	33
4	840	>3 million	24	25
5	672	>75 million	12	20
6	336	>1 billion	12	16
7	96	>15 billion	12	14
8	12	>186 billion	12	12

Table 1. $\alpha(z)$, and $\beta(z)$ for the disjoint perfectly compatible subsets solution. The "Optimal $\alpha(z)$" and "Optimal $\beta(z)$" columns indicate values that would be achievable if it were possible to generate a single large perfectly compatible subgroup consisting of 100 elements.

6.3.1. *Example.* Let us assume some nearly worst case conditions: 100 subscribers, 11 attributes, and subscriptions containing up to 4 equalities in each perfectly compatible subset. Table 1 shows $\alpha(z)$ and $\beta(z)$ as a function of z. Also shown is the impossible, but optimal values that would be obtainable were it possible to have a perfectly compatible subgroup of size 100.

This table clearly demonstrates that over the course of a test, which likely involves thousands of publications, the difference between the simple solution and the "impossible but optimal" one will be noticeable, at least as far as $\alpha(z)$ is concerned. The difference for $\beta(z)$ is not as great.

If the difference between the "impossible but optimal" solution and the simple one provided above is too troubling we can generate subsets with varying degrees of overlap. More analysis of the overlap case is in Section

7. Generating Overlapping Perfectly Compatible Subsets

It is possible to generate perfectly compatible subgroups that are not disjoint. For example, let $e_{i,j}$ denote the predicate that asserts that field i is equal to value j. We can then represent a conjunction of such equalities by a tuple like $(e_{1,5}, e_{3,2})$ which means that the subscription matches if field 1 equals value 5 of filed 1 and field 3 equals the 2nd value for that field. Using this notation we see that we can easily generate perfectly compatible subscription groups with overlap. For example, let $S_1 = \{(e_{1,1}, e_{2,1}), (e_{2,1}, e_{3,1}), (e_{3,1}, e_{4,1})\}$ and let $S_2 = \{(e_{1,1}, e_{2,1}), (e_{2,1}, e_{3,2}), (e_{3,2}, e_{4,2})\}$. Clearly both are perfectly compatible subgroups and the 1st subscription is a member of both. Furthermore, we can then generate another perfectly compatible subset that overlaps with the last subscription in S_2 but does not overlap with S_1. In this way we

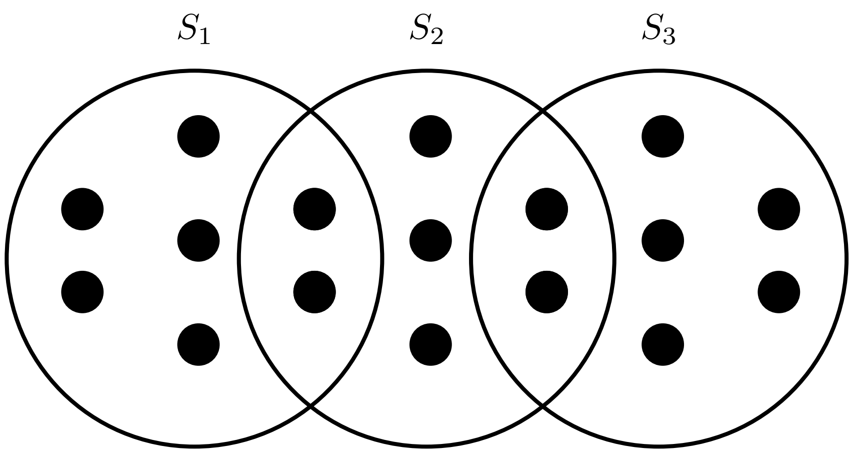

FIGURE 7.1. Graphical depiction of 3 perfectly compatible sets, S_1, S_2, and S_3, each of which contains $t = 7$ subscriptions, with an overlap of $v = 2$.

z	$\alpha(z)$, $v = 0$	$\alpha(z)$, $v = 1$	$\alpha(z)$, $v = 2$	$\alpha(z)$, $v = 3$	$\alpha(z)$, $v = 4$
1	96	99	98	98	100
2	336	392	433	478	534
3	672	784	896	1046	1252
4	840	980	1120	1330	1657
5	672	784	896	1064	1344
6	336	392	448	532	672
7	96	112	128	152	192
8	12	14	16	19	24

TABLE 2. $\alpha(z)$ for various values of v.

can generate an arbitrary number of perfectly compatible subgroups with "overlap 1" (meaning that each subgroup shares exactly 1 element with exactly 1 other subgroup). This same general procedure can be extended to generate set of perfectly compatible subgroups of size t with overlap v where $v \leq t/2$. Figure 7.1 depicts the case with $r = 3$, $t = 7$, and $v = 2$.

To compute $\alpha(z)$ for overlapping compatible groups we must consider two cases, $v < z$ or $v \geq z$. We consider the $v < z$ case first. If we intend to generate h subscriptions then we can generate $r = \text{floor}((h-t)/(t-v)) + 1$, assuming $h \geq t$. To see why, we note that the first perfectly compatible subgroup contains t elements by definition. The second one adds $t - v$ new elements since v of them overlap with the first. Thus every subgroup except the first generates $t - v$ new elements, and the first generates the full t elements. Note that if $v < z$ then generating a publication that matches some combination of z subscriptions in subgroup S_i will never be the same combination as we'd get if we matched a combination of subscriptions from a different subgroup, S_j. Thus, the total number of unique distinct subscription groups we could form with a match rate of z is given by $\alpha(z) = \text{floor}((h-t)/(t-v))\binom{t}{z}$.

If $v \geq z$ things are a bit more complex. The calculation of r does not change, but now we might generate a combination of z subscriptions in one subset, S_i, that also appears in another subset S_j. For example, if $v = 3$ and $z = 2$ then there are $\binom{3}{2}$ combinations of subscriptions that can be made if we search for subscriptions in S_1 or S_2, since $v = 3$ hosts are common to these subsets. So, while there are $\binom{t}{z}$ unique combinations possible if we choose subscriptions from S_1, and $\binom{t}{z}$ combinations if we choose subscriptions from S_2, there are only $2\binom{t}{z} - \binom{v}{z}$ *unique* combinations of subscriptions of we choose subscriptions from $S_1 \cup S_2$. In general, if there are r perfectly compatible subgroups, each of which has v members which overlap with only one other group, then there are $\alpha(z) = r\binom{t}{z} - (r-1)\binom{v}{z}$ unique combinations of z subscriptions that can be made to match the next publication.

Table 2 compares $\alpha(z)$ for several different values of v, including $v = 0$ (disjoint subgroups) using the same assumptions as were used to build Table 1. Clearly increasing the amount of overlap increases the diversity. Indeed, compared to disjoint subsets, an overlap of $v = 4$ roughly doubles the diversity.

In general we note that $\beta(z)$ should improve compared to the the disjoint subsets case. Figure 7.2 shows the extra flexibility gained by allowing subset to overlap. A single figure is clearly not a formula or a proof, but I feel relatively confident that having overlapping subgroups only increases $\beta(z)$. Additionally, since the gap between actual and theoretically optimal (but impossible) $\beta(z)$ is relatively small, it is probably less important that $\alpha(z)$.

7.1. Subscription and Publication Generation Procedure.

These are very similar to their counterparts in Section 6. To generate subscriptions, each new perfectly compatible subset is started with v subscriptions from the most recently

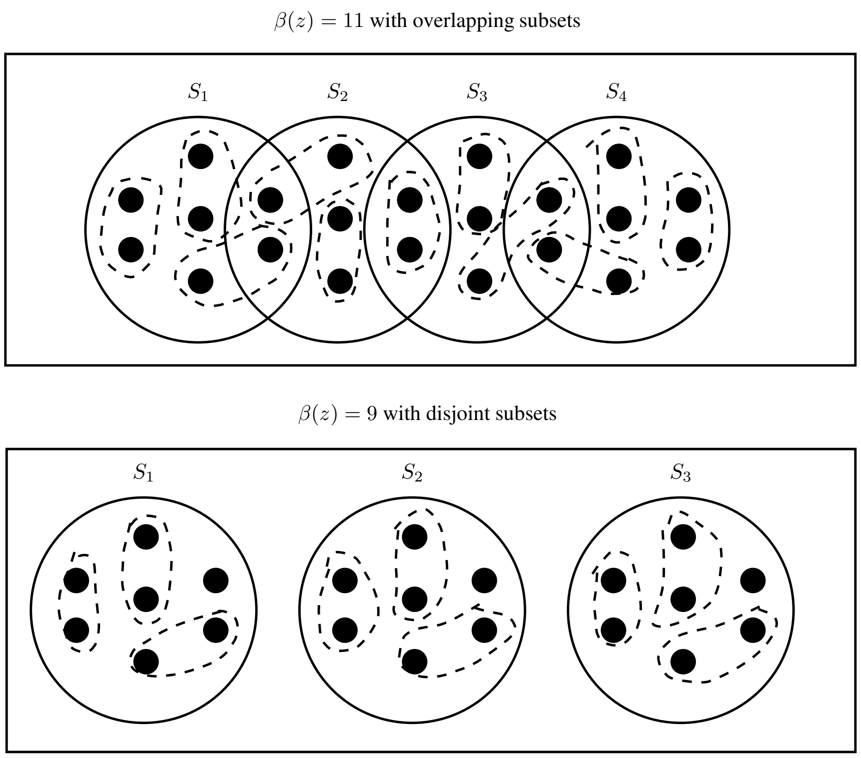
$\beta(z) = 11$ with overlapping subsets
S_1
S_2
S_3
S_4
$\beta(z) = 9$ with disjoint subsets
S_1
S_2
S_3

FIGURE 7.2. The top figure demonstrates that 22 subscriptions can yield a $\beta(z) = 11$ with overlapping subgroups while 21 subscriptions yields a $\beta(z) = 9$ with disjoint subgroups.

generated perfectly compatible subset. The only difficulty is that we can no longer rely on the sorted list of predicate sizes provided by Algorithm 1 to prevent Algorithm 2 from generating a new subscription that has a dependency on an subscription generated earlier. This is because some of the subscriptions in the subset being generated came from a previous subset and have already been assigned values. Thus, care must be taken to ensure all newly generated subscriptions are independent of the overlapping one.

The publication generation procedure is unchanged.

Unfortunately time did not permit us to code up the overlapping variant though it should not be too difficult to do so.