Universiteit Leiden

Opleiding Informatica

Portfolio Selection from Large Molecular Databases using Genetic Algorithms

Name:	Alice de Vries
Studentnr:	0980013
Date:	July 2, 2014
1st supervisor:	Dr. Michael Emmerich
2nd supervisor:	Dr. André Deutz

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

1 Introduction

High-Throughput Screening (HTS) is used by drug researchers to assess whether molecular compounds can be used in drug design. HTS uses advanced technologies to test thousands of compounds a day. HTS is expensive and inefficient, therefore Virtual Screening (VS) methods have been developed. Virtual screening [10] uses computational techniques to select a subset of molecules from virtual libraries. Molecular compounds are selected based on their chemical properties. A problem with existing VS methods is that they do not take into account the risk related to selecting chemically similar compounds. Selecting compounds with similar chemical structures is risky. If the structure proves to be unusable, then all similar compounds will most likely be unusable.

In an unpublished technical report Yevseyeva et al. [11] have suggested that VS with risk of similarity can be seen as a Portfolio Selection Problem (PSP). The PSP model was proposed by Markowitz [7] to solve the problem of asset selection in the financial world. In finance you wish to select a portfolio of assets that maximizes expected return while minimizing risk.

To find a suitable portfolio in the set of optimal portfolios (that is, in the efficient set of portfolios, we first have to find the efficient set of portfolios. The graphical representation of the efficient set is often referred to as non-dominated front or Pareto front (PF). A portfolio is element of the efficient set if it is not dominated by any other portfolio. In the case of PSP a portfolio i is dominated by portfolio j if $i.return \leq j.return$ `and` $i.risk \geq j.risk$ given $i \neq j$.

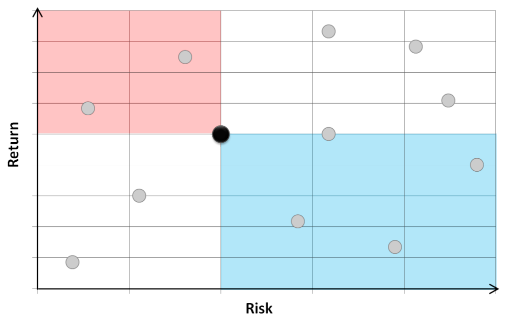


Figure 1: The given portfolio is dominated by all portfolios in the upper left quadrant but dominates all portfolios in the lower right quadrant. The portfolios in the upper right and lower left quadrants are imcomparable.

This is demonstrated in Figure 1. The given portfolio is more efficient than any portfolios in the lower right quadrant. Thus none of the portfolios in this quadrant belong to the PF. The given portfolio is less efficient than any portfolio in the upper left quadrant. When there are no portfolios in this quadrant the given portfolio is part of the PF. By checking domination for all portfolios we can find the PF. Figure 2 shows an example of a PF.

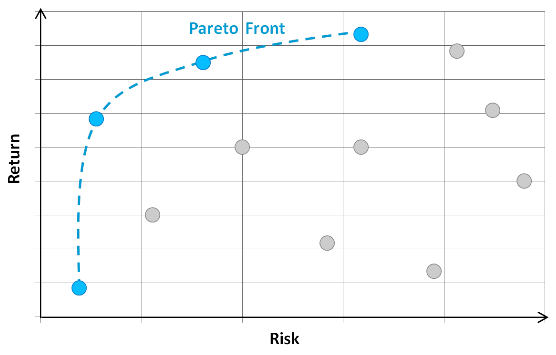


Figure 2: Pareto front for the given portfolios

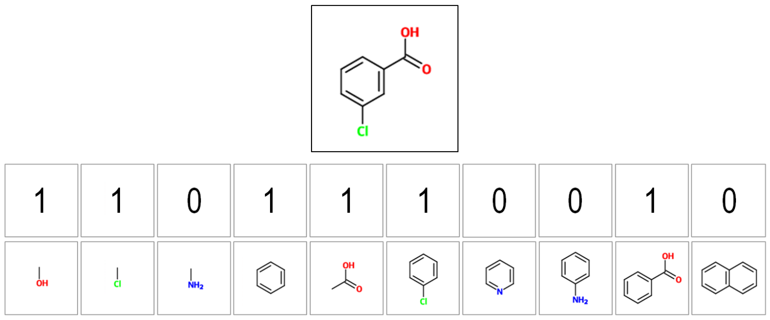
In virtual screening we wish to select a subset of compounds for chemical testing. We want a subset that maximizes expected return and minimizes risk of similarity. Yevseyeva et al. have proposed a model formulation for subset selection of compounds. To determine the correlation between compounds they used Tanimoto similarity of molecular fingerprints combined with the correlation function from the Solow Polasky diversity measure.

Tanimoto similarity T_s is a measure of similarity between two bit vectors A and B. It can be calculated as:

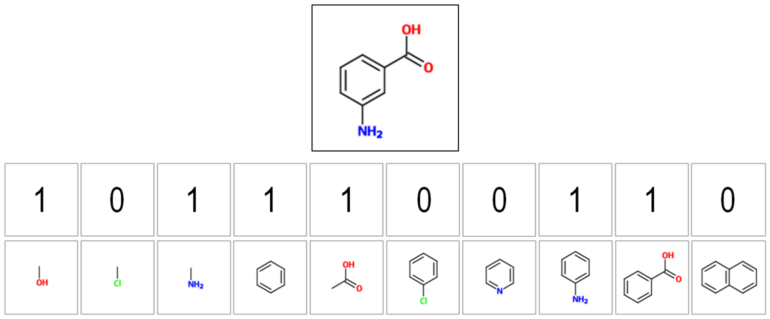
$$T_s(A,B) = \frac{\sum_i A_i \wedge B_i}{\sum_i A_i \vee B_i} \tag{1}$$

For molecular compounds we calculate the Tanimoto similarity using their molecular fingerprint. A molecular fingerprint is a bit vector where each bit represents whether a chemical substructure is part of the molecular compound ('1') or not ('0'). A simplified example of molecular fingerprints is shown in Figure 3. For every pair of compounds in this example we can calculate the Tanimoto similarity:

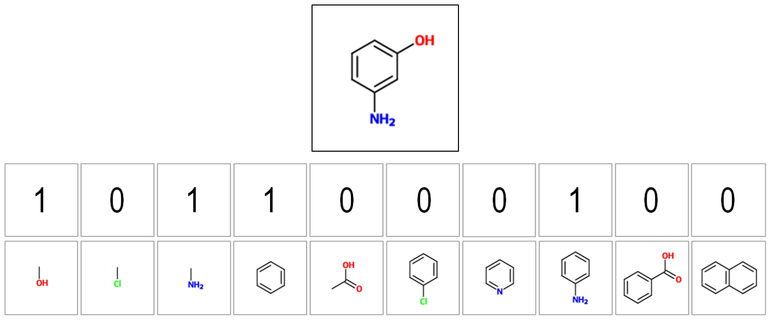
$$T_s(A,B) = \frac{4}{8} = 0.5, \quad T_s(A,C) = \frac{2}{8} = 0.25, \quad T_s(B,C) = \frac{4}{6} = 0.667$$



(a) compound A



(b) compound B



(c) compound C

Figure 3: Simplified example of molecular fingerprints

In [9] Solow and Polasky discuss different measures of diversity. Their aim was to propose a diversity measure, that can be used as an objective function when the aim is to preserve a set of species that maximize biological diversity. Solow and Polasky took a utilitarian approach, suggesting species should be preserved for their possible future medical benefit. They have suggested a diversity function:

$$D(P) = vF^{-1}v^T \tag{2}$$

where F is the correlation matrix and v is a vector of ones. They also suggested a correlation function $f(d_{ij}) = e^{-\theta d_{ij}}$ that computes the correlation between two individuals, i and j, given their distance. The distance between two compounds is calculated using the Tanimoto similarity: $d_{ij} = 1 - T_s$. The value of θ has been computed so that a Tanimoto similarity of 0.9 equals a correlation of 0.95 and a Tanimoto similarity of 0.5 equals a correlation of 0.1. Solving for θ gives us $\theta = 5$.

Yevseyeva et al. defined a quadratic programming (QP) formulation and used the Gurobi QP solver [6] to find an exact solution to the problem. In this thesis we describe how we used genetic algorithms to approximate a solution. Meinl et al. [8] have suggested a method of virtual screening that maximizes diversity and maximizes (bio-)activity. In this thesis we propose a similar model. We use a different diversity measure, namely the Solow Polasky diversity measure. Instead of maximizing activity we maximize expected return which is based on activity.

In section 2 we define risk, diversity and return in the scope of virtual screening and elaborate on the model used by Yevseyeva et al. and the diversity maximizing model we have designed. Section 3 describes the genetic algorithms used. We discuss the results and compare them to the results of the quadratic programming used by Yevseyeva et al. in section 4. In section 5 we discuss a problem with the probability formulation in the original model and suggest an improved probability formulation.

2 Portfolio selection for virtual screening

Markowitz's portfolio selection theory [7] uses the expected return $E(R)$ and the variance of expected returns $V(R)$, also referred to as risk, to find the Pareto efficient set of portfolios. An efficient portfolio maximizes return while minimizing risk. This is formally:

$$\begin{aligned}
&\mathrm{E(R)} = \sum_{i=1}^{n} x_i * r_i && \rightarrow \quad \max \\
&\mathrm{V(R)} = \sum_{i=1}^{n}\sum_{j=1}^{n} x_i * x_j * \sigma_{ij} && \rightarrow \quad \min \\
&\text{s.t.} \quad x_i \in [0,1]; \quad i,j \in [1,n]
\end{aligned} \tag{3}$$

where n is the number of assets, x_i is the proportion of money invested in asset i, r_i is the return on investment of asset i and σ_{ij} is the covariance of returns of assets i and j.

In the case of virtual screening we take molecular compounds as assets that can either be selected ($x_i = 1$) or not selected ($x_i = 0$). We define the return on investment of compound i as the probability of success times profit per unit investment minus the probability of no success. This is formally:

$$r_i = p_i * \frac{g_i - c_i}{c_i} + (1 - p_i) * \frac{-c_i}{c_i} \tag{4}$$

where p_i is the probability of success for compound i, g_i is the gain given compound i is successful and c_i is the price of compound i. In drug research the gain for all successful compounds is equal:

$$g_i = \mathrm{G} \tag{5}$$

In this paper we consider drug discovery done within funded research projects. Therefore we assume that drug researchers have to stay within a research budget. Thus investment is constrained by budget B:

$$\sum_{i=1}^{n} c_i \leq \mathrm{B} \tag{6}$$

Budget assigned to buying compounds that is not spent has to be returned and from the perspective of the research project can be considered as a loss. We can therefore say that the size of individual investments is irrelevant as long as the constraint (6) holds. Combining this fact with (4) and (5) Yevseyeva et al. come to a new return formulation:

$$r_i = p_i * G \tag{7}$$

Probability of success for a given compound is proportional to the (bio-)activity of that compound:

$$p_i = \mathrm{k} * a_i \tag{8}$$

where a_i is the activity of compound i and k is a proportionality constant. Typically the activity data is available as the logarithm of a_i, say ℓ_i. In this case we use $a_i = e^{\ell_i}$. To solve for k we need a fixed point. As in Yevseyeva et al. we use the average activity and the average probability of success that can be defined as:

$$\overline{a} = \frac{1}{n} * \sum_{i=1}^{n} a_i \ , \quad \overline{p} = \frac{1}{n} * \sum_{i=1}^{n} p_i \tag{9}$$

If we combine (8) and (9) we can derive k:

$$\mathrm{k} = \overline{p} * \frac{1}{\overline{a}} \tag{10}$$

By substituting k in (8) we obtain an expression for the success probability that is inversely proportional to the average activity:

$$p_i = a_i * \overline{p} * \frac{1}{\overline{a}} \tag{11}$$

Given the rule of thumb used by drug researchers from the Leiden Academic Center for Drug Research (LACDR) which assumes that 1 in 100 compounds will be successful, the probability of success becomes:

$$p_i = \frac{1}{100} * \frac{a_i}{\overline{a}} \tag{12}$$

As previously stated the risk in virtual screening is related to the similarity between the compounds selected. The more similar two compounds are the more similar their expected returns will be. As correlation and covariance differ only by a constant we can use the correlation as the covariance and obtain the same ranking of portfolios. We use the correlation term from the Solow-Polasky diversity measure [9] to calculate the correlation between two compounds:

$$\sigma_{ij} = e^{-\theta d_{ij}} \tag{13}$$

where d_{ij} is the distance between compounds i and j and θ is the correlation factor. A higher θ will decrease the correlation for a given distance, whereas a smaller θ will increase the correlation for a given distance. We calculate the distance between compounds i and j as:

$$d_{ij} = 1 - s_{ij} \tag{14}$$

where s_{ij} is the Tanimoto similarity between compounds i and j.

Combining (3), (6), (7), (11) and (13) we get the portfolio selection model for virtual screening:

$$\begin{aligned}
&\mathrm{E(R)} = \sum_{i=1}^{n} x_i * p_i * G \qquad \rightarrow \quad \max \\
&\mathrm{V(R)} = \sum_{i=1}^{n}\sum_{j=1}^{n} x_i * x_j * \sigma_{ij} \qquad \rightarrow \quad \min \\
&\text{where} \quad p_i = \frac{1}{100} * \frac{a_i}{\bar{a}}; \qquad \sigma_{ij} = e^{-\theta(1-s_{ij})} \\
&\text{s.t.} \ \ x_i \in \{0,1\}; \ \ i,j \in [1,n]; \ \ \textstyle\sum_{i=1}^{n} c_i \leq \mathrm{B}
\end{aligned} \tag{15}$$

In practice the testing equipment is limited to process only a fixed amount of compounds per unit of time. We will therefore also implement the model as in (15) with the added cardinality constraint:

$$\sum_{i=1}^{n} x_i = \mathrm{N} \tag{16}$$

Another way to look at the virtual screening is to maximize diversity instead of minimizing risk of similarity. We use the diversity as proposed by Solow and Polasky, which can be calculated as the sum of the entries of the inverse of the correlation matrix for selected compounds:

$$\mathrm{D} = \sum_{i=1}^{m} \sum_{j=1}^{m} (F^{-1})_{ij} \rightarrow \max \tag{17}$$

where m is the number of compounds selected and F^{-1} is the inverse of the correlation matrix for all selected compounds. This gives us the model:

$$\begin{aligned} &\mathrm{E(R)} = \sum_{i=1}^{m} p_i * G && \rightarrow \quad \max \\ &\mathrm{D} = \sum_{i=1}^{m} \sum_{j=1}^{m} (F^{-1})_{ij} && \rightarrow \quad \max \\ &\text{where} \quad p_i = \frac{1}{100} * \frac{a_i}{\overline{a}}; && F_{ij} := e^{-\theta(1-s_{ij})} \\ &\text{s.t.} \;\; x_i = 1; \quad i, j \in [1, m]; \;\; \sum_{i}^{m} c_i \leq \mathrm{B} \end{aligned} \tag{18}$$

While both models use the Solow Polasky correlation function, (15) minimizes the sum of the entries if the correlation matrix while (18) maximizes the sum of the entries of the inverse of the correlation matrix. This entails that the first favours smaller portfolios while the latter favours bigger portfolios, since all entries of the matrices are positive.

3 Implementation

We have implemented two well-known evolutionary multi-objective optimization algorithms (EMOAs), namely NSGA-II [3] and SMS-EMOA [4], using Python 2.7. In this section we will describe these evolutionary algorithms. As we instantiate them on binary search spaces we will refer to them as genetic algorithms (GAs). We will discuss the similarity and differences between the two. We elaborate on the variation operators (selection, crossover and mutation) chosen. Moreover, we will show the correctness of the implementation by applying it to a different multi-objective optimization problem, for which an exact solution has been obtained. From this section onwards we will use the generic terms solution and population, where a solution is a portfolio of molecules and a population is a set of portfolios.

3.1 NSGA-II

The Non-Dominated Sorting Genetic Algorithm II (NSGA-II) [3] is a well-known genetic algorithm that is specifically designed to solve multi-objective problems. Algorithm 1 shows the NSGA-II algorithm.

Algorithm 1 NSGA-II

```
initialize $P_0$
fast-nondominated-sort($P_0$)
$Q_0 \leftarrow$ make-new-population($P_0$)
for $t = 0$ to NUM_ITER do
    $R_t \leftarrow P_t \cup Q_t$
    $F \leftarrow$ fast-nondominated-sort($R_t$)
    while $|P_{t+1}| + |F_i| <$ POP_SIZE do
        $P_{t+1} \leftarrow P_{t+1} \cup F_i$
        $i \leftarrow i + 1$
    $F_i \leftarrow$ crowding-distance-assignment($F_i$)
    $P_{t+1} \leftarrow P_{t+1} \cup F_i$
    $P_{t+1} \leftarrow P_{t+1}[0 :$ POP_SIZE$]$
    $Q_{t+1} \leftarrow$ make-new-population($P_{t+1}$)
```

It starts by initialising the parent population P_0. Then it applies `fast-nondominated-sort`, which sorts the solutions into layers of non-dominated fronts. The next step is to create an offspring population Q_0 from the parent population with `make-new-population`, which uses the variation operators to create a new population.

Now for each iteration it combines the parent (P_t) and offspring (Q_t) population in R_t. It applies `fast-nondominated-sort` on the combined population. Then it creates the parent population P_{t+1} from which the next generation of offspring is created. NSGA-II does this by adding the non-domination fronts while $|F_i|$ is smaller than $\text{POP\_SIZE} - |P_{t+1}|$. Then it sorts the next non-domination front using `crowding-distance-assignment` and adds this front to P_{t+1}. Then it trims the solutions from this front that have the smallest crowding distance so that P_{t+1} has exactly POP_SIZE solutions. Finally using `make-new-population` the new offspring population Q_{t+1} is created again.

As mentioned earlier a non-dominated front is a representation of the efficient set of solutions in the objective space. It can be found by checking domination for all available solutions. The solutions that are not dominated by any other solutions make up a non-dominated front. By identifying the solutions that are only dominated by solutions on the non-dominated front(s) we found before, we find the next front. We do this until all solutions are part of a non-dominated front. We name the non-dominated fronts $F_1, F_2, ...F_{last}$, where front F_1 is the Pareto front for the given set of solutions and front F_i is better than front F_j for all $j > i$. When a solution breaks a constraint it will not dominate any other solutions and be part of the lowest ranked front F_{last}. Figure 4 shows the non-dominated fronts for the given set of solutions.

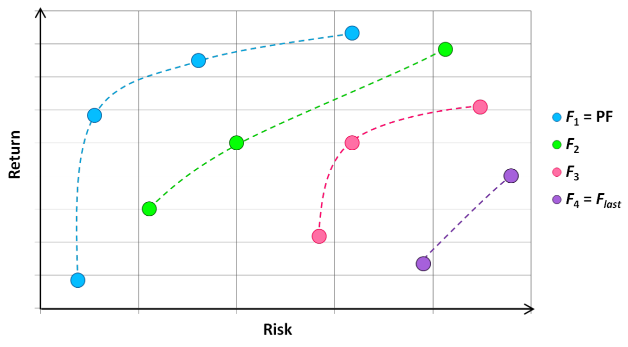


Figure 4: An example of non-dominated fronts

NSGA-II uses `fast-nondominated-sort`, which is a fast way of sorting solutions into non-dominated fronts. Algorithm 2 describes how it works.

Algorithm 2 `fast-nondominated-sort`(P)

```
for all $p \in P$ do
  $S_p \leftarrow \varnothing$
  $n_p \leftarrow 0$
  for all $q \in P$ do
    if $p$ dominates $q$ then
      $S_p \leftarrow S_p \cup \{q\}$
    else if $q$ dominates $p$ then
      $n_p \leftarrow n_p + 1$
  if $n_p = 0$ then
    $F_1 \leftarrow F_1 \cup \{p\}$
$i \leftarrow 1$
while $F_i \neq \varnothing$ do
  for all $p \in F_i$ do
    for all $q \in S_p$ do
      $n_q \leftarrow n_q - 1$
      if $n_q = 0$ then
        $F_{i+1} \leftarrow F_{i+1} \cup \{q\}$
  $i \leftarrow i + 1$
return $F$
```

For every pair of solutions p, q in the population it checks whether or not p dominates q. If so, q is added to the set of solutions dominated by p (S_p). If instead q dominates p, the number of solutions that dominate p is increased ($n_p + 1$). If $n_p = 0$ after domination is checked for all $q \in P$, we can say solution p is non-dominated and p is added to the first non-dominated front F_1. Now for every solution p in F_1 it loops through all solutions $q \in S_p$ and decreases n_q. If $n_q = 0$, solution q is part of the next non-dominated front F_2. This is repeated until all solutions are part of a non-dominated front.

An optimal non-dominated front consist of a uniformly distributed set of solutions. This makes solutions which are crowded together less desirable than solutions which are spread out. To determine which solutions of an non-dominated front are more desirable, NSGA-II uses `crowding-distance-assignment`. As can be seen in Algorithm 3, it measures the crowding distance for all solutions in a given set of solutions L. Crowding distance is the space between the neighbouring solutions of a given solution in each dimensions. For each solution the distance is set to zero. Then for each objective $m \in \textsc{obj}$ we sort the solutions. The first and last solutions, which are the boundary points of the non-dominated front, get a distance of ∞ because they should always be part of a Pareto front approximation. For the other solutions their distance is increased with the difference in m of their neighbouring solutions. Finally it sorts the solutions in L according to their crowding distance and returns the sorted set.

Algorithm 3 `crowding-distance-assignment`(L)

```
$l \leftarrow |L|$
**for all** $i \in L$ **do**
  $i.distance \leftarrow 0$
**for all** $m \in \textsc{obj}$ **do**
  $L \leftarrow \text{sort}(L, m)$
  $L[1].distance, L[l].distance \leftarrow \infty$
  **for** $i = 2$ **to** $l - 1$ **do**
    $L[i].distance \leftarrow L[i].distance + (L[i+1].m - L[i-1].m)$
$L \leftarrow \text{sort}(L, distance, \text{order=descending})$
**return** $L$
```

3.2 SMS-EMOA

The S-Metric Selection Evolutionary Multi-objective Optimization Algorithm (SMS-EMOA) [4] is another well-known genetic algorithm that is partially derived from NSGA-II. It is similar to NSGA-II for bi-objective problems. Algorithm 4 shows the SMS-EMOA algorithm.

It starts by initialising the parent population P_0. Then it applies `fast-nondominated-sort`, which is the same as in NSGA-II (see Algorithm 2). The next step is to create one offspring q_0 from the parent population with `make-new-individual`, which uses the variation operators to create a new, single solution.

Algorithm 4 SMS-EMOA

```
initialize $P_0$
fast-nondominated-sort($P_0$)
$q_0 \leftarrow$ make-new-individual($P_0$)
for $t = 0$ to NUM_ITER do
    $R_t \leftarrow P_t \cup \{q_t\}$
    $F \leftarrow$ fast-nondominated-sort($R_t$)
    $r \leftarrow$ hypervolume-assignment($F_{last}$)
    $P_{t+1} \leftarrow R_t - \{r\}$
    $q_{t+1} \leftarrow$ make-new-individual($P_{t+1}$)
```

Now for each iteration SMS-EMOA adds the parent population P_t and offspring solution q_t to R_t. It then sorts the solutions using `fast-nondominated-sort`. To the last non-dominated front F_{last} it then applies `hypervolume-assignment`, which returns the solution r with the smallest contribution to the hypervolume indicator. SMS-EMOA then creates a new parent population P_{t+1} using all solutions in R_t except r. From this new parent population a new offspring q_{t+1} is created.

In SMS-EMOA the distribution of solutions over the set is driven by `hypervolume-assignment`. As we are dealing with a bi-objective problem, the hypervolume contribution of a given solution can be seen as the area of exclusive domination for that given solution. The higher the contribution of a given solution, the more important that solution is. Algorithm 5 shows how it can be calculated in the case of two objectives.

First the algorithm sets the contribution for all points to 1. Then for both objectives it sorts the solutions. The boundary points get a contribution of ∞ because they should always be part of a Pareto front approximation. For the other solutions their contribution is multiplied with the difference of objective $m \in \textsc{obj}$ between a given solution i and the solution next to it. In case of an objective to be minimized, the solution $i+1$ with a higher value of m is considered. In case of maximization, the solution $i-1$ with a lower m is considered. When this is done for all objectives, it sorts the solutions by their contribution in descending order. It then returns the solutions with the smallest contribution to the hypervolume indicator, which will be discarded.

Algorithm 5 `hypervolume-assignment`(L); $|\textsc{obj}| = 2$

```
$l \leftarrow |L|$
for all $i \in L$ do
  $i.contribution \leftarrow 1$
$L \leftarrow \mathrm{sort}(L, \textsc{obj}[0])$
$L[1].contribution, L[l].contribution \leftarrow \infty$
for all $m \in \textsc{obj}$ do
  for $i = 2$ to $l - 1$ do
    if $m \rightarrow \min$ then
      $L[i].contribution \leftarrow L[i].contribution * (L[i+1].m - L[i].m)$
    else if $m \rightarrow \max$ then
      $L[i].contribution \leftarrow L[i].contribution * (L[i].m - L[i-1].m)$
$L \leftarrow \mathrm{sort}(L, contribution, \mathrm{order{=}descending})$
return $L[l]$
```

3.3 Variation Operators

Genetic algorithms use selection, crossover and mutation operators to create offspring. According to Bäck [1] we can distinguish two types of selection: mating selection and environmental selection. We have used the same mating selection, crossover and mutation operators for both algorithms so we can better compare their performance.

The choice of operators depends on the representation of solutions and vice versa. We define a solution as a sorted list of indices of the compounds selected. For example, if we select every second compound out of n compounds our solution would be defined as $[2, 4, 6, \ldots, n]$. If we selected only the first 10 out of n compounds, the solution would be $[1, 2, 3, \ldots, 10]$. By representing a solution as a list of selected compounds we can easily constrain the number of molecules chosen.

3.3.1 Environmental Selection

Environmental selection determines which solutions are kept and which solutions are discarded after each iteration. For NSGA-II and SMS-EMOA the environmental selection is done within the main algorithm. As can be seen in Algorithm 1, NSGA-II uses $(\mu + \lambda)$-selection. This means that from μ parents λ offspring are created. From the combined $\mu + \lambda$ solutions the best μ are kept. SMS-EMOA on the other hand uses a $(\mu + 1)$-selection, as can be seen in Algorithm 4. This means that from μ parents only one offspring is created and the μ best of the $\mu + 1$ solutions are kept.

3.3.2 Mating Selection

With mating selection we can choose which of the surviving solutions are used to create offspring. Although not part of the original SMS-EMOA, we have chosen to use binary tournament selection on both NSGA-II and SMS-EMOA. Binary tournament selection randomly selects two solutions from parent population P_{t+1} and compares them. The best solution is added to the mating pool. Because of this the mating pool will often consist of multiple copies of the better solutions and a few not so good solutions.

For NSGA-II solution i is better than solution j when $i.rank < j.rank$ or $(i.rank = j.rank$ `and` $i.crowding\_distance > j.crowding\_distance)$.
For SMS-EMOA solution i is better than solution j when $i.rank < j.rank$ or $(i.rank = j.rank$ `and` $i.hypervolume > j.hypervolume)$.

3.3.3 Crossover

The crossover operator combines two parent solutions from the mating pool to create two offspring solutions. We apply crossover with a probability of p_{CO}. We have implemented n-point crossover. This means we randomly select n points and after each point we change the parent we copy onto the offspring. So we start with copying the first parent p_1 onto the first offspring c_1 and the second parent p_2 onto the second offspring c_2 until the first crossover point. After this point we copy p_2 onto c_1 and p_1 onto c_2 until the next crossover point where we switch again. An example of 2-point crossover is shown in Figure 5.

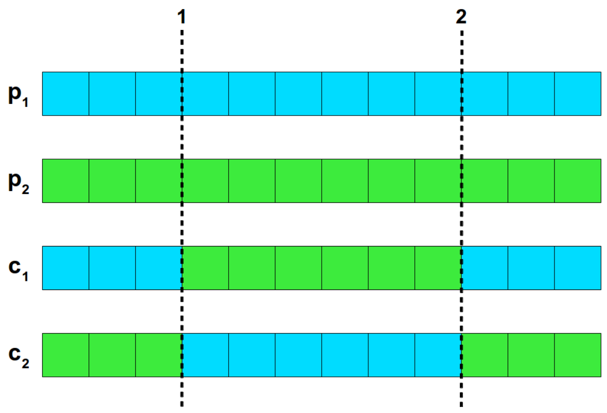
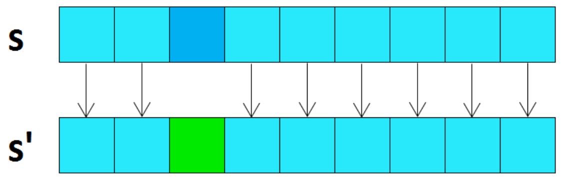


Figure 5: 2-point crossover

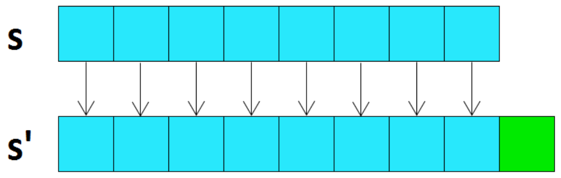
3.3.4 Mutation

We use the mutation operator to introduce new compounds to the population. We have chosen a simplistic mutation operator. For every compound in the solution we replace it with a random new compound with probability p_{MR}. We also expand our list of solutions by adding a compound with probability p_{MA}. And we delete a compound from a given solution with probability p_{MD}. Examples of these mutations are shown in Figure 6.

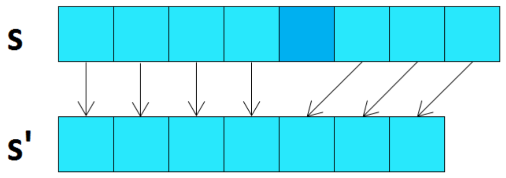
We can constrain the number of compounds selected by changing p_{MA} and p_{MD}. In case of a constant number of compounds we take $p_{\mathrm{MA}} = 0$ and $p_{\mathrm{MD}} = 0$. For if we do not add or delete compounds, the number of compounds will stay equal to the size of the initial solution.



(a) replace



(b) add



(c) deleting

Figure 6: Mutation

3.4 Correctness of code

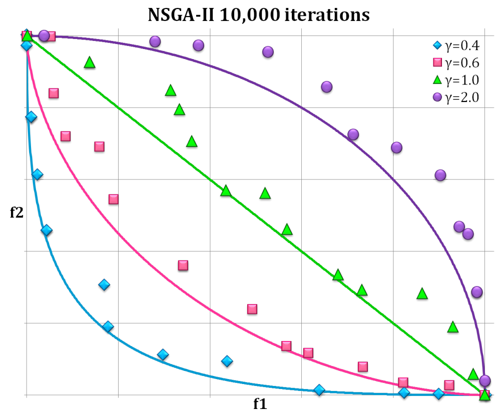
To test the implementation we applied the algorithm to a different optimization problem for which an exact solution exists. We used the generalized Schaffer's problem proposed by Emmerich and Deutz [5]:

$$\begin{aligned} f_1(x) &= \tfrac{1}{n^\alpha}\left(\sum_i^n x_i^2\right)^\alpha \rightarrow \min \\ f_2(x) &= \tfrac{1}{n^\alpha}\left(\sum_i^n (1-x_i)^2\right)^\alpha \rightarrow \min \end{aligned} \tag{19}$$

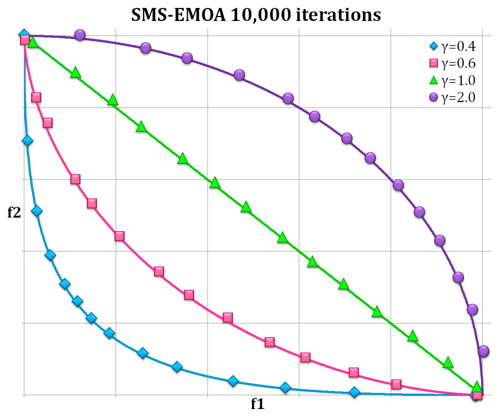
Emmerich and Deutz have proved that the Pareto front can be computed with:

$$f_2 = (1 - f_1^\gamma)^{\frac{1}{\gamma}}, \qquad \text{where } \gamma = \tfrac{1}{2\alpha} \tag{20}$$

We applied both NSGA-II and SMS-EMOA for 10,000 iterations with a population size of 15. We used $n = 5$ and $\gamma \in \{0.4, 0.6, 1.0, 2.0\}$. The results are shown in Figure 7. For NSGA-II we can see how the solutions approach the Pareto front, but only a few are actually on the PF. For SMS-EMOA we can see that most of the solutions are on the PF. From this we can conclude that both implementations are approximating the PF but that SMS-EMOA delivers more accurate results. It is noticeable that SMS-EMOA performs better than NSGA-II even though it is computationally more expensive as it does 15 times more function evaluations.



(a) NSGA-II



(b) SMS-EMOA

Figure 7: Optimization of Schaffer's Problem

4 Results

To compare our proposed model to the existing models we have experimented with all three models. We compare the NSGA-II algorithm with the SMS-EMOA algorithm. We also compare the results obtained with the genetic algorithms to the exact Gurobi Quadratic Programming solver used by Yevseyeva et al.

4.1 Experiments

We use three datasets of chemical compounds provided in Yevseyeva et al. Each dataset contains a subset of compounds from the ZINC database, which has been sorted by activity. The first dataset contains the 1000 *most active* compounds, the second contains the 2500 *most active* compounds and the third contains the 5000 *most active* compounds. Each dataset consists of the cost and logarithmized activity for each compound and the Tanimoto similarity for every pair of compounds in the dataset. In Figure 8 we have plotted the activity and cost of the compounds in each of the datasets.

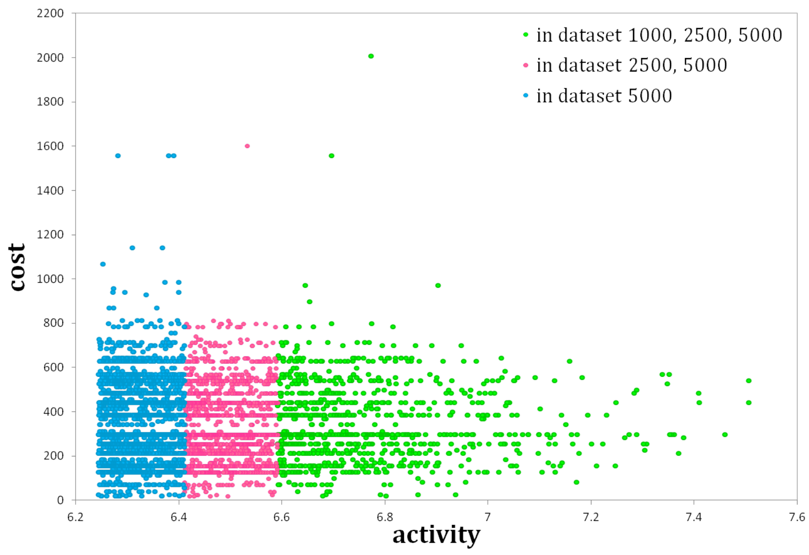


Figure 8: Cost and activity for the compounds in the datasets of 1000 (green), 2500 (green and pink) and 5000 (green, pink and blue) compounds

We have appplied all three models to each of the datasets. First we applied the model in (15) which minimizes risk. Then we used the cardinality constraint model, which uses the constraint in (16) with the model in (15). For this model we compare the results that we obtained with NSGA-II and SMS-EMOA to the results obtained by Yevseyeva et al. using quadratic programming. After that we applied our new model that maximizes diversity as described in (18).

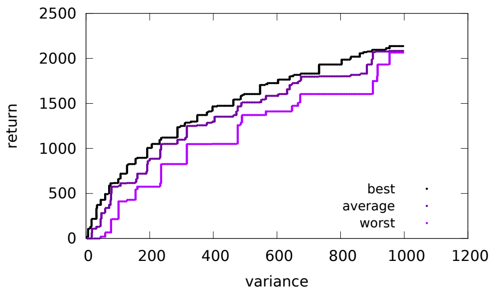
As in Yevseyeva et al., we set the fixed amount of molecules to one hundred ($N = 100$). The budget is set to a hundred times the average cost of the compounds in the dataset: $B = 100 * \overline{c}$. For the dataset of 1000 compounds this yields $B = 34502$, for the dataset of 2500 compounds this yields $B = 34400$ and for the dataset of 5000 compounds this results in $B = 34622$.

We used a small population size: POP_SIZE $= 10$. Besides, we used 1-point crossover with probability $p_{\mathrm{CO}} = 0.2$. In other words, for every 10 offspring there are 2 that have been created using 2 parents while the other 8 offspring are copies of some parent. We replaced a compound in the solution with $p_{\mathrm{MR}} = 0.01$, which on average is one compound per offspring. We added a compound to half the offspring on average: $p_{\mathrm{MA}} = 0.5$. We removed a compound from a solution on average once per 10 offspring: $p_{\mathrm{MD}} = 0.1$.

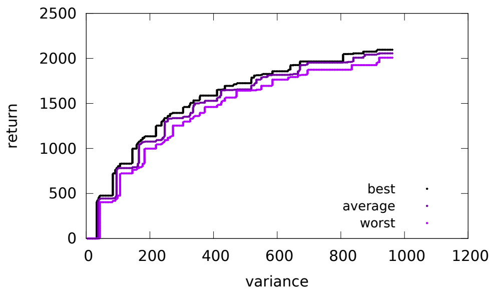
For the dataset of 1000 compounds we run NSGA-II for 10,000 iterations. Since SMS-EMOA creates 1 offspring each iteration whereas NSGA-II creates 10 offspring each iteration, we run SMS-EMOA for 100,000 iterations. For the bigger datasets we increase the number of iterations with the same factor as the size of the dataset. Thus for the dataset containing 2500 compounds we run NSGA-II for 25,000 iterations and SMS-EMOA for 250,000 iterations. For the dataset of 5000 compounds we run NSGAII for 50,000 iterations and SMS-EMOA for 500,000 iterations. This way the number of evaluations of the objective functions is the same for both algorithms.

Because of the randomness of the initialization, crossover and mutation we should not compare only one run of NSGA-II and SMS-EMOA. Instead we should compare some average of several runs. Comparisons using a convergence measure are not possible since there is no exact formulation of the Pareto front. We therefore compute the average using attainment curves [2] which allow for a visual comparison. This method uses a generalization of the median as an average and is robust against outliers.

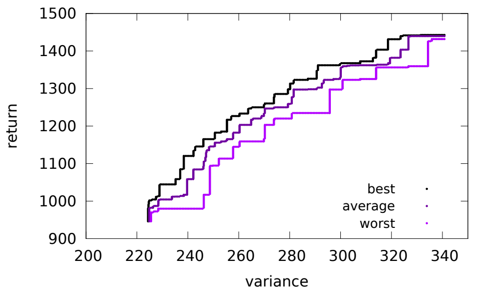
In Figure 9 we can see attainment curves of the best, average and worst out of 10 runs for all three models. We can see that SMS-EMOA is very robust, especially in the models minimizing risk. NSGA-II is less robust, but good enough to let us compare the average runs.



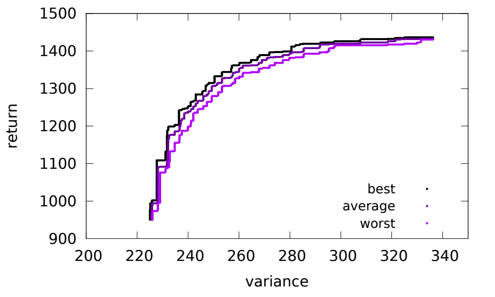
(a) NSGA-II minimizing risk



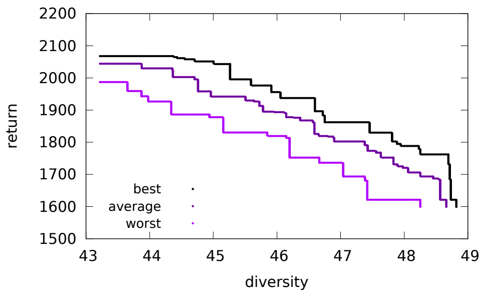
(b) SMS-EMOA minimizing risk



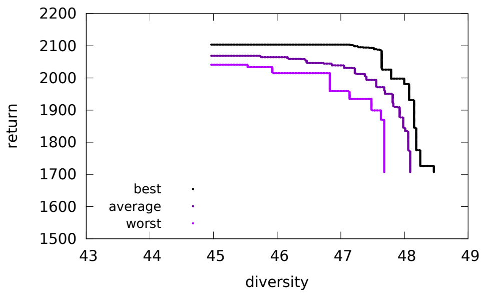
(c) NSGA-II cardinality constraint



(d) SMS-EMOA cardinality constraint



(e) NSGA-II maximizing diversity

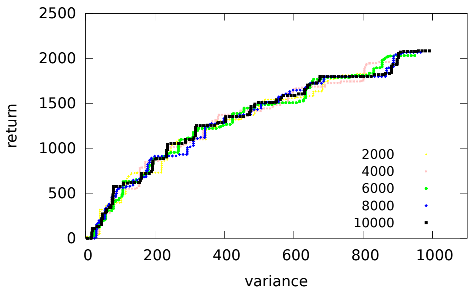


(f) SMS-EMOA maximizing diversity

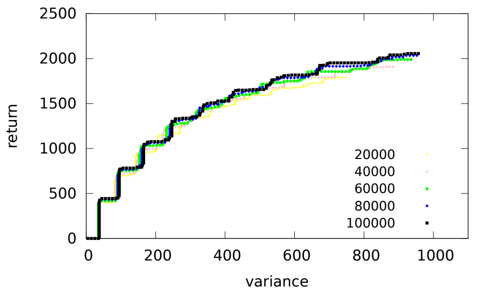
Figure 9: Robustness of NSGA-II and SMS-EMOA based on attainment curves

4.2 Minimizing risk without cardinality constraint

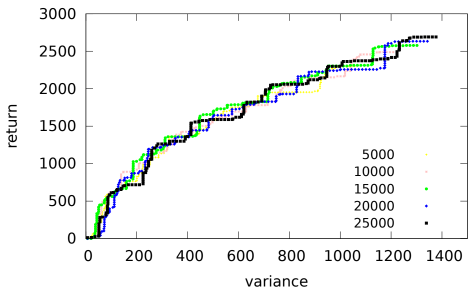
In this section we will show and discuss the results for the risk minimizing model as described in (15). Figure 10 shows how the average front changes over the iterations. Both NSGA-II and SMS-EMOA seem to find solutions of the efficient set in a few iterations. However these solutions are found for low variance only. It seems that more iterations result in more efficient solutions for higher variance.



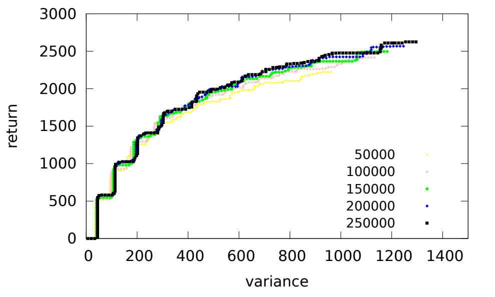
(a) NSGA-II dataset 1000



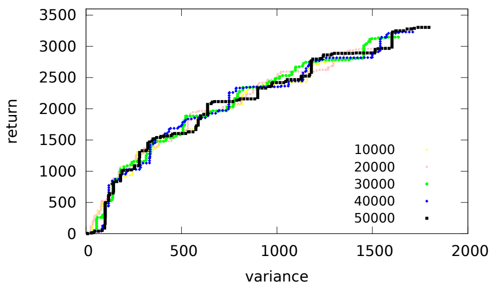
(b) SMS-EMOA dataset 1000



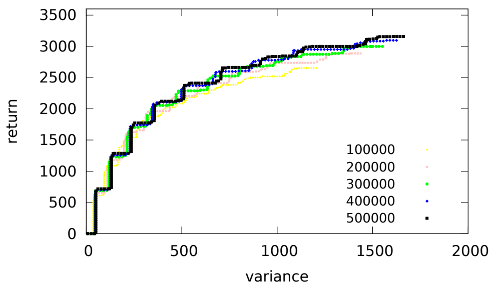
(c) NSGA-II dataset 2500



(d) SMS-EMOA dataset 2500



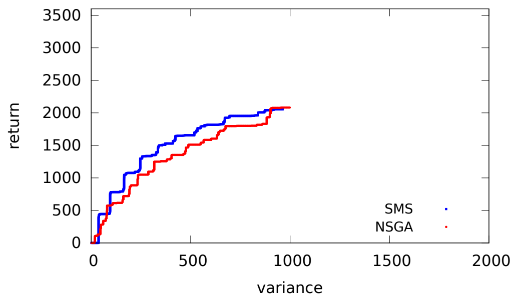
(e) NSGA-II dataset 5000



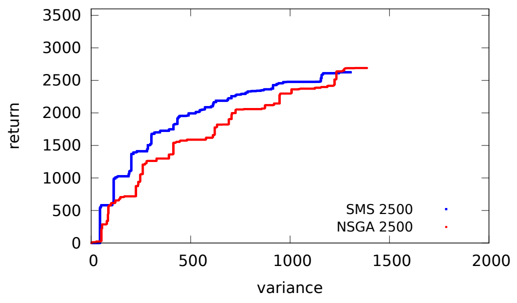
(f) SMS-EMOA dataset 5000

Figure 10: Comparison of number of iterations for minimization of risk

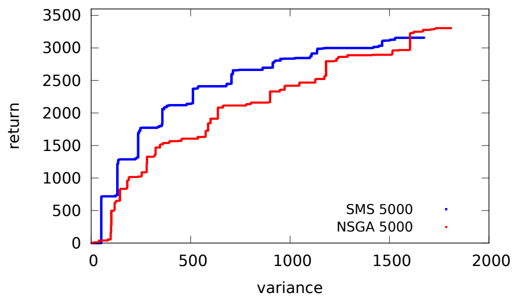
Figure 11 shows a direct comparison of NSGA-II and SMS-EMOA on each of the datasets. We can clearly see SMS-EMOA outperforms NSGA-II in finding more concave solutions. However, NSGA-II is better in finding solutions with higher variance.



(a) dataset 1000



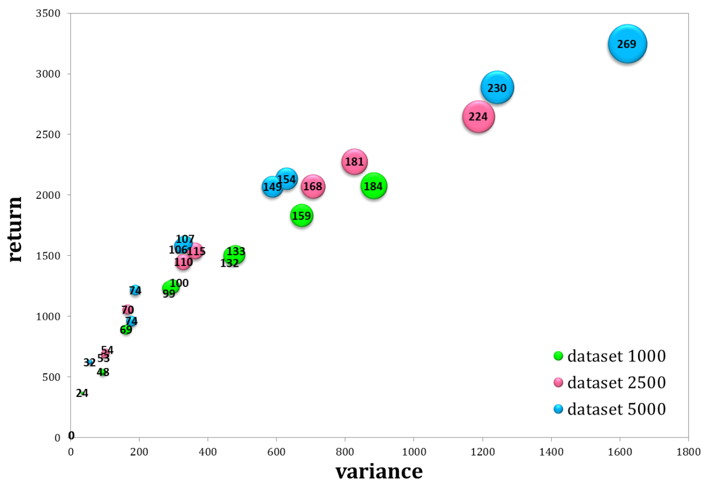
(b) dataset 2500



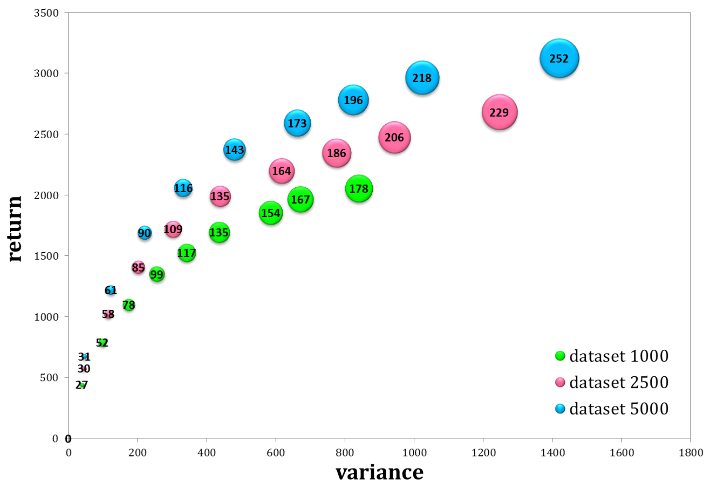
(c) dataset 5000

Figure 11: Comparison of NSGA-II and SMS-EMOA

Figure 12 gives an insight in the cardinality of the solutions in the population obtained after a typical run of both algorithms. We can see a problem with a model that minimizes risk without a cardinality constraint. It selects very small subsets, as well as the empty set, as part of the optimal front. Given the model this makes sense as there is no subset with a higher return given a variance of 0. However, in practice this is undesirable.



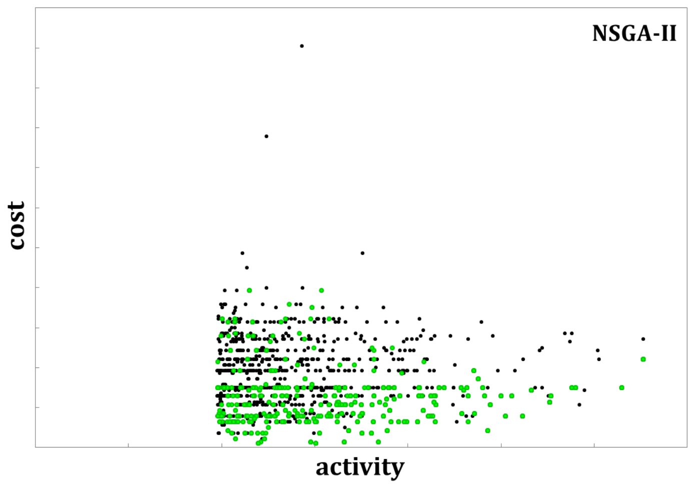
(a) NSGA-II



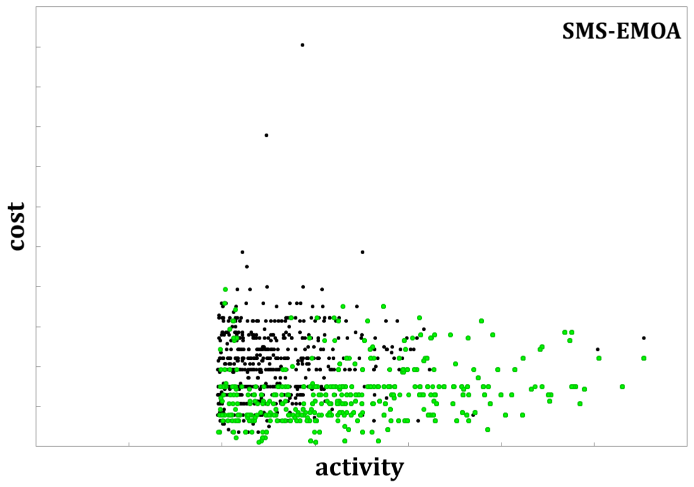
(b) SMS-EMOA

Figure 12: Solutions in the population after 1 run. Size of the point is representative of the size of the subset

In Figure 13 we can see the cost and activity for all compounds in the dataset of 1000 compounds. Highlighted are the compounds selected in any of the solutions of the population after a typical run. We can see a difference between NSGA-II and SMS-EMOA. NSGA-II seems to prefer cheap compounds while SMS-EMOA selects more expensive and more active compounds. This explains the difference in performance we saw in Figure 11.



(a) NSGA-II

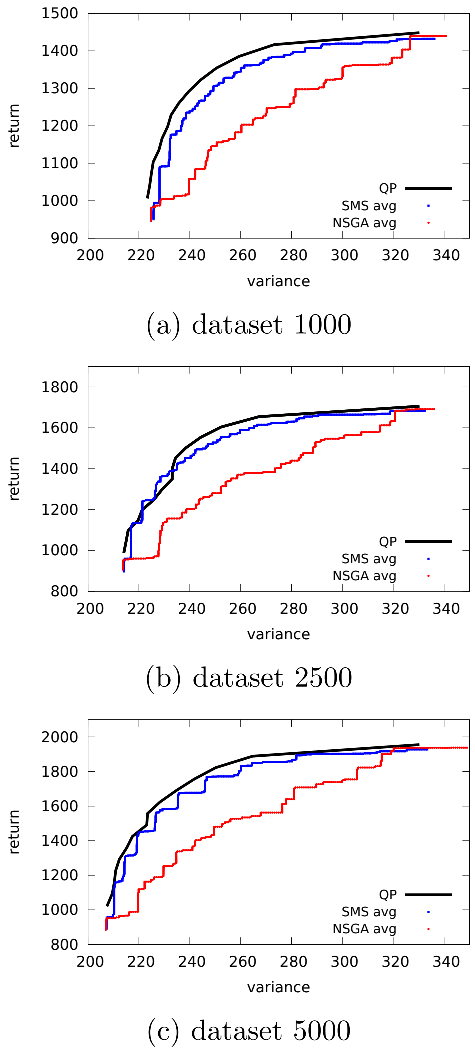


(b) SMS-EMOA

Figure 13: Cost and activity for all compounds in dataset 1000. In green are the compounds selected in the population after a typical run.

4.3 Cardinality constraint

In this section we show and discuss the results we obtained for the risk minimizing model as in (15) with the added cardinality constraint as in (16). In Figure 14 we can compare the performance of NSGA-II, SMS-EMOA and the Gurobi QP solver used by Yevseyeva et al. We can clearly see that SMS-EMOA outperforms NSGA-II and that SMS-EMOA approaches the Pareto front found by Gurobi.



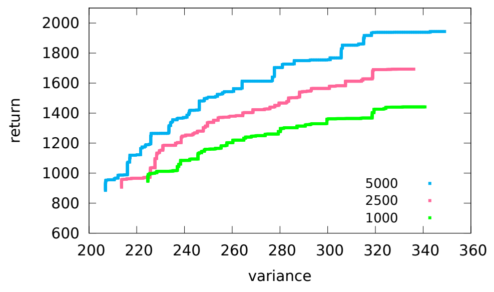
(a) dataset 1000

(b) dataset 2500

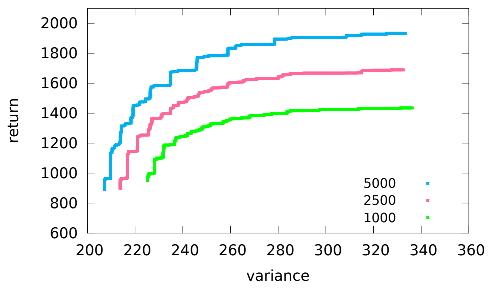
(c) dataset 5000

Figure 14: Comparison of NSGA-II, SMS-EMOA and Gurobi QP solver

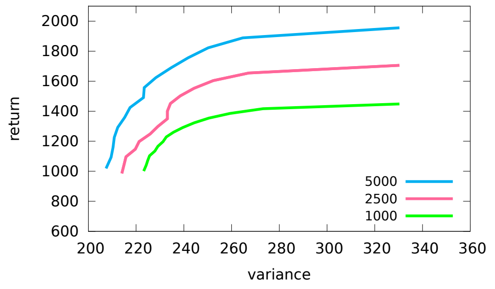
In Figure 15 we compare the datasets for each of the methods. It is interesting to see that the bigger datasets perform better, since from all datasets a constant number of 100 compounds are selected and the 2500 and 5000 dataset do not contain more active compounds, which have a higher success probability and expected return. One could argue that this may be the result of applying more iterations in the bigger datasets. However, as can be seen in Figure 16 the 1000 dataset converges after 10,000/100,000 iterations, which means that running the algorithms for more iterations will not be effective. The real reason for this behaviour is the probability calculation. When we calculate probability we normalize the activity with the formula (11). As the bigger datasets contain more compounds with low activity, the average activity is lower in the bigger dataset. Therefore the success probability of a given compound is higher in the bigger dataset than that of the same compound in the smaller dataset. We will discuss this further in section 5.



(a) NSGA-II

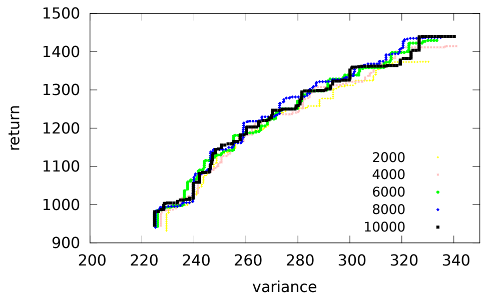


(b) SMS-EMOA

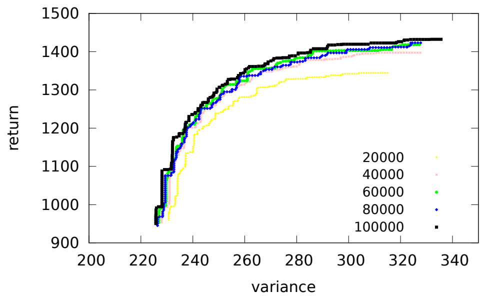


(c) Gurobi QP solver

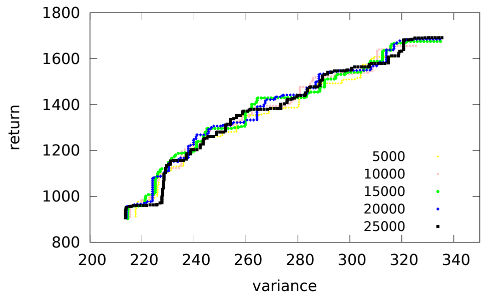
Figure 15: Comparison of datasets



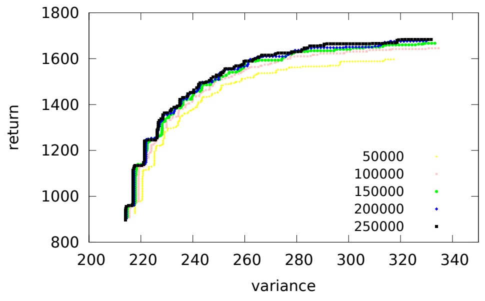
(a) NSGA-II dataset 1000



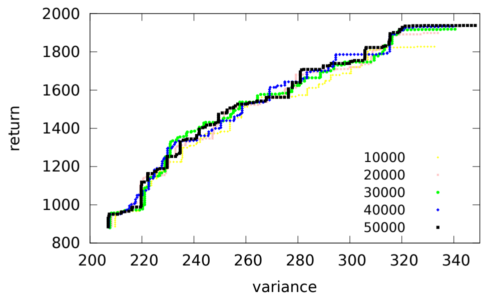
(b) SMS-EMOA dataset 1000



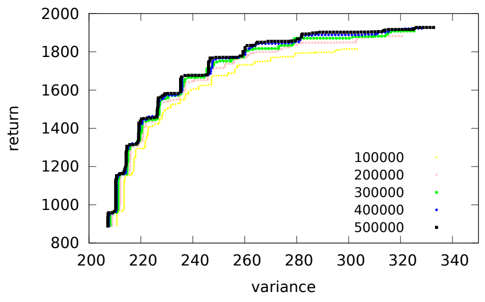
(c) NSGA-II dataset 2500



(d) SMS-EMOA dataset 2500



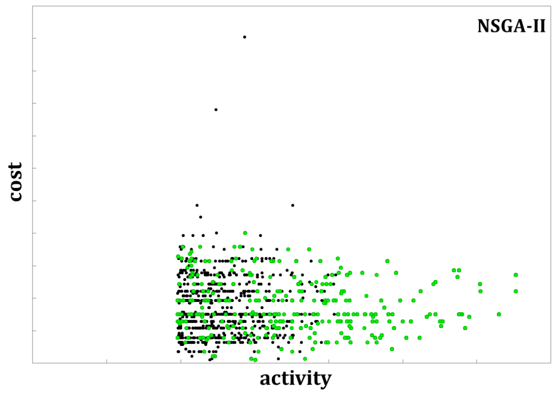
(e) NSGA-II dataset 5000



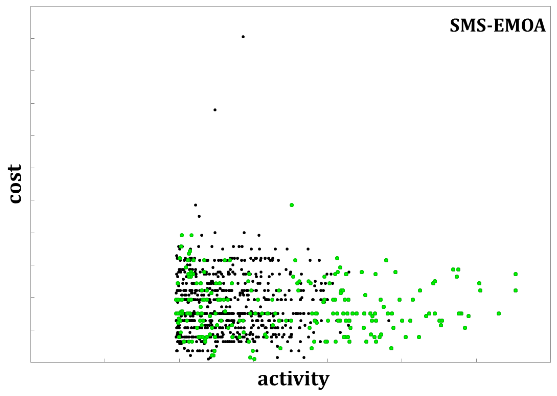
(f) SMS-EMOA dataset 5000

Figure 16: Comparison of number of iterations

In Figure 17 we can see the cost and activity for all compounds in the dataset of 1000 compounds. Highlighted are the compounds selected in all solutions of the population after one run. These graphs clearly show us that not only the most active compounds are selected, though 90 percent of the 100 most active molecules have been selected at least once. We can see that not only cheap molecules are selected. It is noticeable that more expensive compounds have been chosen over less expensive compounds with equal or higher activity. This shows how the genetic algorithms select compounds based on both activity and similarity.



(a) NSGA-II

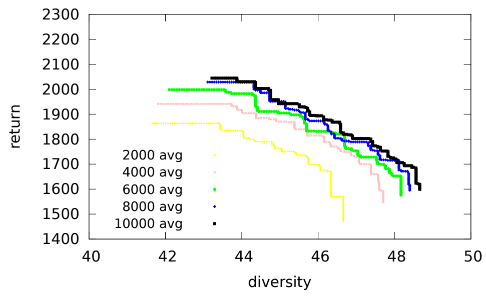


(b) SMS-EMOA

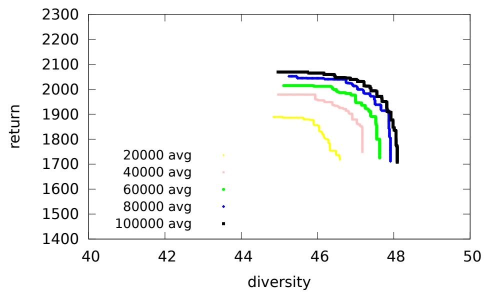
Figure 17: Cost and activity for all compounds in dataset 1000. In green are the compounds selected in the population after 1 run.

4.4 Maximizing diversity

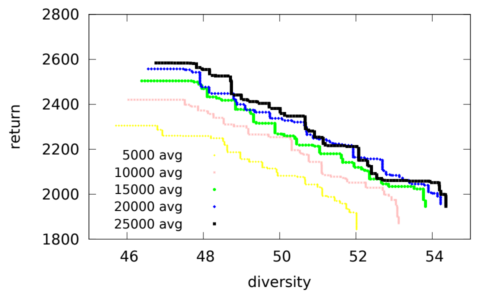
In this section we show and discuss the results obtained for the model in (18). In Figure 18 we can see the average front at 5 intervals. We can see that running SMS-EMOA for 10 times more iterations is justified, as NSGA-II converges after only 10,000 iterations while SMS-EMOA needs 100,000 to converge. Figure 18 also shows justification for increasing the number of iterations as the size of the database increases.



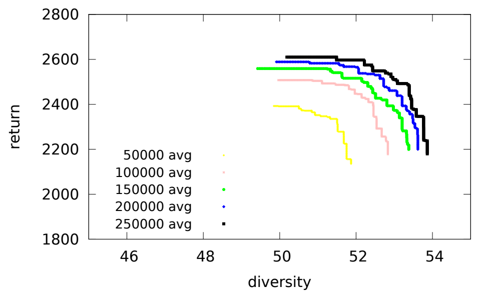
(a) NSGA-II dataset 1000



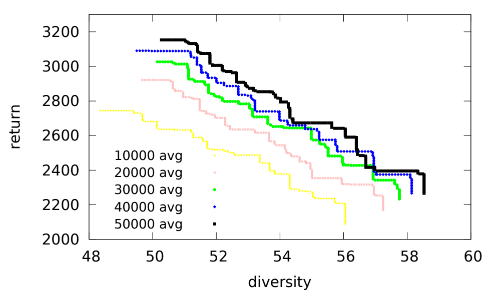
(b) SMS-EMOA dataset 1000



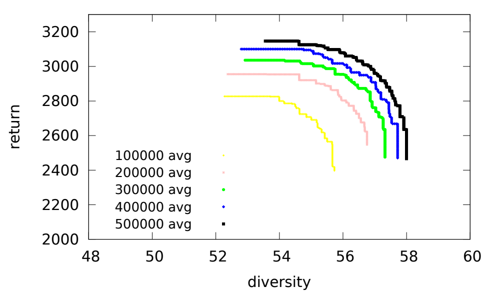
(c) NSGA-II dataset 2500



(d) SMS-EMOA dataset 2500



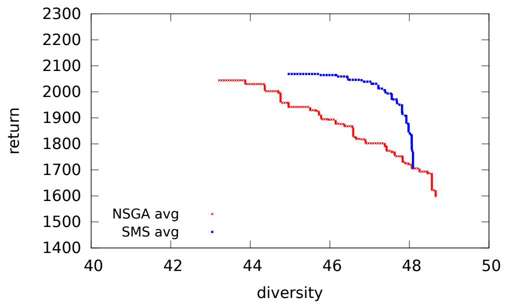
(e) NSGA-II dataset 5000



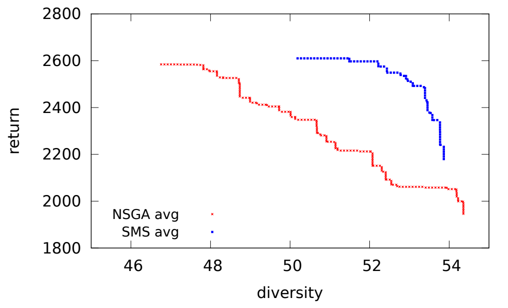
(f) SMS-EMOA dataset 5000

Figure 18: Comparison of number of iterations for maximization of diversity

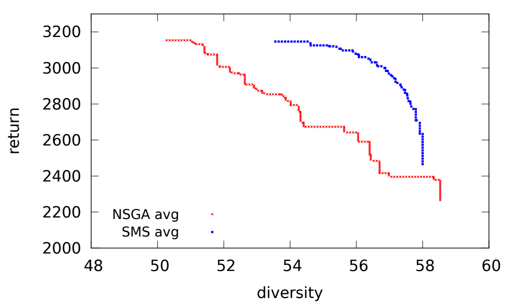
Figure 19 shows a direct comparison of NSGA-II and SMS-EMOA on each of the datasets. We can clearly see SMS-EMOA outperforms NSGA-II in finding more concave solutions. However, NSGA-II is better in finding solutions with higher diversity.



(a) dataset 1000



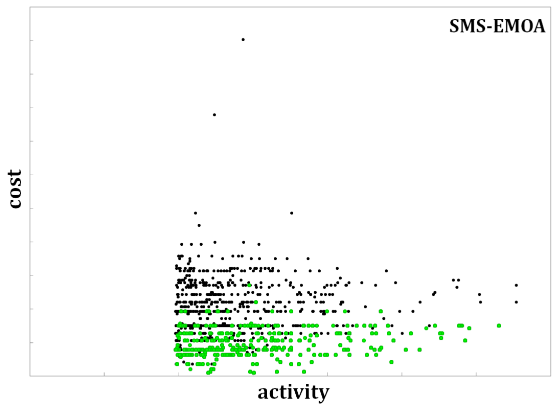
(b) dataset 2500



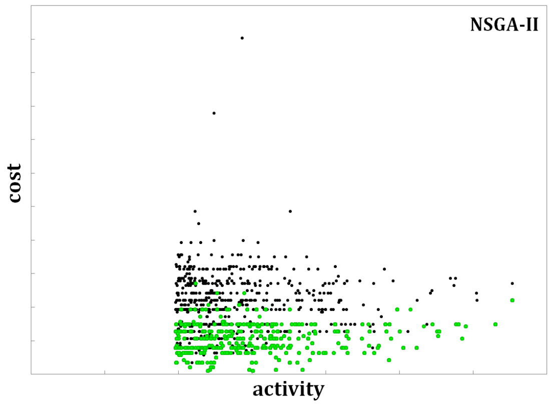
(c) dataset 5000

Figure 19: Comparison of NSGA-II and SMS-EMOA

In Figure 20 we see the cost and activity for all compounds in the dataset of 1000 compounds. Highlighted are the compounds selected in all solutions of the population after one run. What we notice is a preference for cheap molecules. This can be explained by looking at the diversity calculation. Diversity will always increase as the cardinality of the solution increases. We want to maximize diversity. By maximizing the cardinality we also maximize the diversity. As we are bound by a budget, we maximize the number of compounds while staying under budget. Thus we select subsets with cheap compounds.



(a) NSGA-II



(b) SMS-EMOA

Figure 20: Cost and activity for all compounds in dataset 1000. Highlighted are the compounds selected in the population after 1 run.

5 Correction for probability calculation

As we have seen in Figure 15, there is a problem with the success probability calculation. A problem arises because the success probability of a compound is inversely proportional to the average activity of the dataset it is in. If the datasets would be uniformly selected from the ZINC database this would be a correct approach. However, as we have seen in Figure 8, the datasets consist of the most active compounds in the database. Thus the bigger datasets contain more compounds with a lower activity than the small dataset. This means that the average activity of the dataset of 1000 compounds is higher than average activity of the datasets containing 2500 and 5000 compounds respectively. As the success probability of a compound is inversely proportional to the average activity of the dataset it is in, the same compound will have a higher probability of success when it is considered as part of the dataset with 5000 compounds than when if it were considered as part of the dataset containing 1000 compounds.

We propose a new formulation for the success probability better suited to datasets without a uniform distribution of activities. As before we assume a proportionality to the activity:

$$p_i = \mathrm{k} * a_i$$

But instead of having k proportional to the average probability in the dataset we use a constant derived from the average probability and activity of the dataset with 1000 compounds:

$$\overline{p}_{1000} = \mathrm{k} * \overline{a}_{1000} \tag{21}$$

where $\overline{a}_{1000}$ is the average activity and $\overline{p}_{1000}$ the average success probability of the compounds in the dataset with 1000 compounds. This gives us:

$$\mathrm{k} = \overline{p}_{1000} * \frac{1}{\overline{a}_{1000}} \tag{22}$$

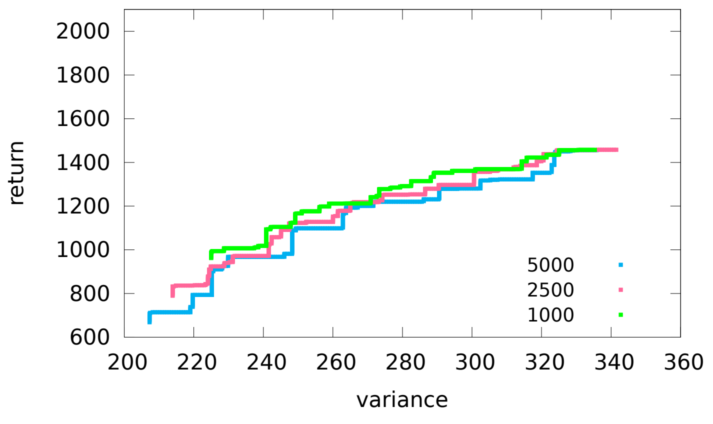
Thus the probability formulation becomes:

$$p_i = \overline{p}_{1000} * \frac{a_i}{\overline{a}_{1000}} \tag{23}$$

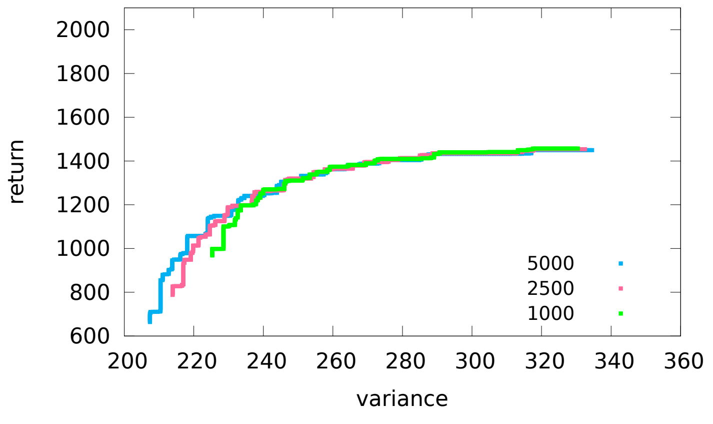
Given the LACDR's drug researchers rule of thumb that the average probability of success is 1/100, we get a new success probability formulation:

$$p_i = \frac{1}{100} * \frac{a_i}{\overline{a}_{1000}} \tag{24}$$

In Figure 21 we can see the average run for the cardinality constraint model using the new probability formulation. We can see how the expected returns of all three datasets are similar now. For the dataset of 1000 compounds the average is the same as in Figure 15 since the proportionality constant k was obtained from this dataset. We can see that with the SMS-EMOA we find the same optimal solutions with higher risk for all datasets. However, for the bigger dataset SMS-EMOA finds better solutions with lower risk. For the NSGA-II we see that solutions selected for the bigger datasets are dominated by the solutions of the small dataset, with the exception of the solutions with lower risk.



(a) NSGA-II



(b) SMS-EMOA

Figure 21: Comparison of datasets with improved probability formulation

6 Conclusions

We have seen three models for solving the portfolio selection problem for drug discovery. The first model maximizes expected return while minimizing variance of returns. The second model is the same as the first model with an added cardinality constraint for which the number of the compounds to be selected is constant. The third model maximizes both expected return and diversity. Yevseyeva et al. solved the second model with the Gurobi Quadratic Programming solver. We have solved all three models using two genetic algorithms, namely NSGA-II and SMS-EMOA. In all three models we haven seen SMS-EMOA performing better than NSGA-II. While NSGA-II performs well in the boundary points of the Pareto front, it seems to struggle with finding concave solutions. For the second model we can see how the SMS-EMOA results are a very close approximation to the Gurobi QP results.

The problem with the first model is that the empty subset is part of the Pareto front. Common sense tell us that testing only a few or no compounds is not effective. The cardinality constraint of the second model solves this problem. However, it is not always desirable to fix the number of compounds selected. The third model solves the problem of the first model without a constraint on the number of compounds selected. The diversity of a subset increases as cardinality of the subset increases, as does the expected return. Thus a small subset of compounds is undesirable. This however poses another problem. Because increasing the cardinality will always cause an increase in diversity, maximizing diversity is basically maximizing the cardinality. Thus the model has a preference for cheap compounds, filling the subset with as many compounds as possible while staying under budget.

We have seen a problem with the success probability formulation. In the original model the activity is normalized given average activity of the dataset. As the average activity increases the probability decreases. Because of this a given subset of the dataset with 1000 compounds has a smaller return than the same subset of the dataset with 5000 compounds. We proposed a new formulation where probability of success is proportional to the activity and the proportionality constant is based on the average activity of the dataset containing 1000 compounds. Our research has suggested that this new formulation is correct.

7 Future work

In this thesis we have focussed on finding the Pareto optimal set of portfolios. However, for in-vitro testing we will need to choose one of the portfolios of the optimal set. The selection of a portfolio from the optimal front was not within the scope of this thesis but poses a new challenge for future work. In their work Yevseyeva et al. used the Sharpe ratio which is often used in financial Portfolio Selection Problems. The Sharpe ratio is a ratio between expected return and variance. A similar ratio could be created for the model that maximizes diversity.

Now that we have shown that the models, solved using the SMS-EMOA algorithm, can be used for virtual screening it would be interesting to see how this can be integrated into the process of drug discovery. A possibility would be to use the SMS-EMOA on bigger datasets to do a pre-selection and then use the Gurobi QP solver to find the optimal solution from the pre-selection. An issue with this is computation time, as the runtime of the SMS-EMOA is proportional to the number of compounds in the dataset and it takes approximately 10 minutes to compute a selection from a dataset of 1000 compounds.

The more likely scenario is using this model as an optimizer on a subset of the database. This subset, of say 2500 compounds, should be selected from the database using some fast heuristic filter. On this subset we can apply the model that has a cardinality constraint to select 100 compounds for physical testing.

Acknowledgements

I would like to thank my supervisor Dr. Michael Emmerich for his guidance and support during my research. I would like to thank Dr. André Deutz for his constructive criticism on my thesis. I would like to thank Dr. Iryna Yevseyeva for providing the results from the Gurobi QP solver and for giving feedback on results obtained with the genetic algorithms. I would like to thank Bart Lenselink for his insights on the drug discovery process and for making the datasets available to me.

References

[1] Thomas Bäck. *Evolutionary algorithms in theory and practice.* Oxford Univ. Press, 1996.

[2] Viviane Grunert da Fonseca, Carlos M Fonseca, and Andreia O Hall. Inferential performance assessment of stochastic optimisers and the attainment function. In *Evolutionary Multi-Criterion Optimization*, pages 213–225. Springer, 2001.

[3] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

[4] Michael Emmerich, Nicola Beume, and Boris Naujoks. An EMO algorithm using the hypervolume measure as selection criterion. In *Evolutionary Multi-Criterion Optimization*, pages 62–76. Springer, 2005.

[5] Michael TM Emmerich and André H Deutz. Test problems based on Lamé superspheres. In *Evolutionary Multi-Criterion Optimization*, pages 922–936. Springer, 2007.

[6] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2014.

[7] Harry Markowitz. Portfolio selection. *The journal of finance*, 7(1):77–91, 1952.

[8] Thorsten Meinl, Claude Ostermann, and Michael R Berthold. Maximum-score diversity selection for early drug discovery. *Journal of chemical information and modeling*, 51(2):237–247, 2011.

[9] Andrew Solow, Stephen Polasky, and James Broadus. On the measurement of biological diversity. *Journal of Environmental Economics and Management*, 24(1):60–68, 1993.

[10] W Patrick Walters, Matthew T Stahl, and Mark A Murcko. Virtual screening–an overview. *Drug Discovery Today*, 3(4):160–178, 1998.

[11] Iryna Yevseyeva, Michael Emmerich, Bart Lenselink, and Ad IJzerman. Molecule portfolio selection. Technical report, LACDR/LIACS, Leiden University, April 2014.

A Main.py

```
from GAs import SMS_EMOA, NSGA
from ReadFiles import *
import time, random
random.seed()

def variables(sim,top):
    C = 1.0/(100.0*math.exp(6.76529709791))
    ID, PRICE, RET, Q = readFiles(sim,top,C,THETA=5,GAIN=1000)
    SIZE_DB       = len(ID)
    SIZE_SUBSET   = 100
    BUDGET        = SIZE_SUBSET*sum(PRICE)/SIZE_DB
    POP_SIZE      = 10
    CO_N          = 1
    CO_R          = 2.0/POP_SIZE
    MR_A          = 5.0/POP_SIZE
    MR_D          = 1.0/POP_SIZE
    MR_C          = 1.0/SIZE_SUBSET
    return locals()

OBJ=[("exp_return","max"),("variance","min")]

VAR=variables("similarityData.txt", "priceActivityData.txt")
a=time.time();
for i in range(10):
    print "run", i+1
    SMS_EMOA(VAR,OBJ,NUM_GEN=100000)
    print "runtime", time.time()-a; a=time.time();
    NSGA(VAR,OBJ,NUM_GEN=10000)
    print "runtime",time.time()-a; a=time.time();

VAR=variables("data/sim2500.txt", "data/top2500.txt")
a=time.time();
for i in range(10):
    print "run", i+1
    SMS_EMOA(VAR,OBJ,NUM_GEN=250000)
    print "runtime", time.time()-a; a=time.time();
    NSGA(VAR,OBJ,NUM_GEN=25000)
    print "runtime",time.time()-a; a=time.time();


VAR=variables("data/sim5000.txt", "data/top5000.txt")
a=time.time();
for i in range(10):
    print "run", i+1
    SMS_EMOA(VAR,OBJ,NUM_GEN=500000)
    print "runtime", time.time()-a; a=time.time();
    NSGA(VAR,OBJ,NUM_GEN=50000)
    print "runtime",time.time()-a; a=time.time();
```

B GAs.py

```python
from MoleculeSelection import *
import importlib,time

class NSGA:
    def __init__(self,VAR,OBJ,NUM_GEN=1000000,TIME=1000000):
        t=time.time();   gen=0
        P=Population(); P.initial(VAR); fast_nondominated_sort(P,OBJ);
        Q=Population(); Q.make_new_pop(P,VAR);
        R=Population();
        while time.time()-t < TIME and gen < NUM_GEN:
            R.solutions = P.solutions+Q.solutions
            F=fast_nondominated_sort(R,OBJ)
            P=Population()
            rank=1
            while len(P.solutions)+len(F[rank])<VAR['POP_SIZE']:
                P.solutions += F[rank]
                rank+=1
            F[rank] = self.crowding_distance_assignment(F[rank],VAR,OBJ
            P.solutions += F[rank]
            P.solutions = P.solutions[:VAR['POP_SIZE']]
            Q=Population(); Q.make_new_pop(P,VAR);
            gen+=1
        print gen,"NSGA"; P.print_solutions(VAR)

    def crowding_distance_assignment(self,L,VAR,OBJ):
        l = len(L);
        for i in L:
            i.distance = 0;
        for m,n in OBJ:
            L=sorted(L,key=lambda x:getattr(x,m));
            max(L,key=lambda x:getattr(x,m)).distance+=99999999;
            min(L,key=lambda x:getattr(x,m)).distance+=99999999;
            for i in range(1,l-1):
                L[i].distance += getattr(L[i+1],m)-getattr(L[i-1],m)
        L=sorted(L,key=lambda x: x.distance,reverse=True)
        for i in range(l):
            L[i].fitness+=i/(VAR['POP_SIZE']*2.0)
        return L


class SMS_EMOA:
    def __init__(self,VAR,OBJ,NUM_GEN=1000000,TIME=1000000):
        t=time.time();   gen=0;
        P=Population(); P.initial(VAR); fast_nondominated_sort(P,OBJ);
        q=Solution([]); q.make_new_solution(P,VAR)
        R=Population();
        while time.time()-t < TIME and gen < NUM_GEN:
            P.solutions = P.solutions+[q]
            F=fast_nondominated_sort(P,OBJ)
            r=self.hypervolume_assignment(F[max(F.keys())],VAR,OBJ)
            P.solutions.remove(r[-1])
            q=Solution([]); q.make_new_solution(P,VAR)
            gen+=1
        print gen,"SMS";P.print_solutions(VAR)
        self.P=P

    def hypervolume_assignment(self,L,VAR,OBJ):
        l = len(L);
        for i in L:
            i.volume = 1.0;
        for (m,n) in OBJ:
            L=sorted(L,key=lambda x:getattr(x,m));
            max(L,key=lambda x:getattr(x,m)).volume+=99999999;
            min(L,key=lambda x:getattr(x,m)).volume+=99999999;
            if n == "min":
                for i in range(1,l-1):
                    L[i].volume *= (getattr(L[i+1],m)-getattr(L[i],m))
            elif n == "max":
                for i in range(1,l-1):
                    L[i].volume *= (getattr(L[i],m)-getattr(L[i-1],m))
        L=sorted(L,key=lambda x: x.volume,reverse=True)
        for i in range(l):
            L[i].fitness+=i/(VAR['POP_SIZE']+1.0)
        return L


def fast_nondominated_sort(P,OBJ):
    F={}; S={}; n={}; F[1]=[]
    for p in P.solutions:
        S[p]=[]; n[p]=0
        for q in P.solutions:
            if dominates(p,q,OBJ):
                S[p]=S[p]+[q]
            elif dominates(q,p,OBJ):
                n[p]+=1
        if n[p]==0:
            F[1]=F[1]+[p]
            p.fitness = 1
    i=1
    while F[i]!=[]:
        F[i+1]=[]
        for p in F[i]:
            for q in S[p]:
                n[q]-=1
                if n[q]==0:
                    F[i+1]=F[i+1]+[q]
                    q.fitness = i+1
        i+=1
    F.pop(i)
    return F


def dominates(p,q,OBJ):
    if p.allowed==False :
        return False;
    if q.allowed==False:
        return True;
    dom = True;
    for m,n in OBJ:
        if n == "min":
            if getattr(p,m) > getattr(q,m):
                return False
        elif n == "max":
            if getattr(p,m) < getattr(q,m):
                return False
    return True;
```

C MoleculeSelection.py

```
import numpy, math, random

class Population:
    def __init__(self):
        self.solutions=[];

    def initial(self,VAR):
        ints=range(VAR['SIZE_DB']);
        for i in range(VAR['POP_SIZE']):
            selected=random.sample(ints,VAR['SIZE_SUBSET']);
            S=Solution(selected);
            S.calculate_objectives(VAR);
            self.solutions+=[S];

    def make_new_pop(self,P,VAR):
        parents = binary_tournament_selection(P,VAR);
        children = n_point_crossover(parents,VAR);
        self.solutions = mutation(children,VAR);
        for s in self.solutions:
                s.calculate_objectives(VAR);

    def print_solutions(self,VAR):
        P=sorted(self.solutions,key=lambda x:x.exp_return);
        print "variance","diversity","exp_return", "num_selected"
        for i in P:
            print i.variance, i.diversity, i.exp_return, len(i.selected


class Solution:
    def __init__(self,selected):
        self.exp_return=0;
        self.variance=0;
        self.diversity=0
        self.total_cost=0;
        self.selected=selected;
    def calculate_objectives(self,VAR):
        self.selected.sort()
        if len(set(self.selected))<len(self.selected):
            self.allowed=False; self.reason="Duplicates"; return
        c=[VAR['PRICE'][i] for i in self.selected]
        self.total_cost = sum(c);
        if self.total_cost>VAR['BUDGET']:
            self.allowed=False; self.reason="Budget"; return
        r=[VAR['RET'][i] for i in self.selected]
        self.exp_return = sum(r)
        M=numpy.array([[VAR['Q'][i][j] for j in self.selected] for i in
        self.variance=numpy.sum(M)
        if not M.size or numpy.linalg.det(M)<0.0001:
            self.allowed=False; self.reason="No␣Inverse"; return
        Minv=numpy.linalg.inv(M)
        self.diversity=(numpy.sum(Minv))
        self.allowed=True;

    def make_new_solution(self,P,VAR):
        parents=[];i=1
        parents = binary_tournament_selection(P,VAR);
        children = n_point_crossover(parents[0:2],VAR);
        self.selected = mutation(children,VAR)[0].selected;
        self.calculate_objectives(VAR);


def binary_tournament_selection(P,VAR):
    parents=[];
    for i in range(VAR['POP_SIZE']):
        x=random.sample(P.solutions,2);
        if x[0].fitness < x[1].fitness:
            parents.append(x[0]);
        else:
            parents.append(x[1]);
    return parents;

def n_point_crossover(parents,VAR):
    children=[];
    for i in range(0,len(parents)-1,2):
        p1=parents[i].selected[:]
        p2=parents[i+1].selected[:]
        if random.random()<VAR['CO_R']:
            if len(p1)>0:
                for i in range(VAR['CO_N']):
                    x=random.randrange(len(p1));
                    a=p2[:x]+p1[x:]
                    p1=p1[:x]+p2[x:]
                    p2=a
        children.append(Solution(p1))
        children.append(Solution(p2))
    return children;


def mutation(children,VAR):
    for s in children:
        l=s.selected
        if random.random()<VAR['MR_A'] and len(l)<VAR['SIZE_DB']:
            l.append(random.randrange(VAR['SIZE_DB']))
        if random.random()<VAR['MR_D'] and len(l)>0:
            l.pop(random.randrange(len(l)))
        for x in range(len(l)):
            if random.random()<VAR['MR_C']:
                l[x] = random.randrange(VAR['SIZE_DB']);
    return children;
```

D ReadFiles.py

D.1 With original probability calculation

```
import math

def readFiles(sim, top, C, THETA=5, GAIN=1000):
    f=open(sim,"rU");
    g=open(top,"rU");
    PRICE=[];
    Activity=[]
    ID=[];
    i=-1;
    for line in g:
        if i>-1:
            j=line.split()
            PRICE.append(int(j[2]));
            Activity.append(math.exp(float(j[1])));
            ID.append(j[0]);
        i+=1;
    Normalization=(len(Activity)/100)/sum(Activity);
    RET=[a*Normalization*GAIN for a in Activity]

    Q = f.readlines()[1:];
    for i in range (len(Q)):
        line=Q[i].split()[1:]
        Q[i]=[math.exp(-THETA*(1-float(x))) for x in line]
    f.close(); g.close();
    print "files␣read", len(ID)
    return ID, PRICE, RET, Q
```

D.2 With improved probability calculation

```
import math

def readFiles(sim, top, C, THETA=5, GAIN=1000):
    f=open(sim,"rU");
    g=open(top,"rU");
    PRICE=[];
    Activity=[]
    ID=[];
    i=-1;
    for line in g:
        if i>-1:
            j=line.split()
            PRICE.append(int(j[2]));
            Activity.append(math.exp(float(j[1])));
            ID.append(j[0]);
        i+=1;
    RET=[C*a*GAIN for a in Activity]

    Q = f.readlines()[1:];
    for i in range (len(Q)):
        line=Q[i].split()[1:]
        Q[i]=[math.exp(-THETA*(1-float(x))) for x in line]
    f.close(); g.close();
    print "files read", len(ID)
    return ID, PRICE, RET, Q
```