# К построению эффективного решения задачи пересечения отрезков

Тарас Вознюк, Василий Терещенко
Факультет кибернетики
Киевский национальный университет имени Тараса Шевченко, Киев, Украина
taarraas@gmail.com, v_ter@ukr.net

## Abstract

Paper presents Balaban's algorithm modification. Most of calculations have been moved from children's nodes to parent, which gave additional performance.

***Ключевые слова:*** *пересечение отрезков, дерево алгоритма, лестница отрезков, детерминированный алгоритм.*

## 1. ВВЕДЕНИЕ

*Постановка проблемы.* В работе рассматривается один из подходов решения задачи детерминированного пересечения отрезков. Результаты решения этой задачи имеют большое теоретическое и практическое значение в информатике и ряде прикладных наук. В частности, необходимость решения задач на пересечение возникает в архитектурном проектировании, компьютерной графике (удаление невидимых линий и поверхностей, построение сложных 3D изображений), в задачах сегментации изображений и оптимального раскроя. В микроэлектронике возникает необходимость проверки пересечений разных компонентов интегральных схем, которые состоят из большого количества элементов. С теоретической точки зрения, разработка эффективных алгоритмов определения пересечений позволяет исследовать сложность и глубинную структуру геометрических задач, что в свою очередь, открывает путь к поиску оптимальных решений.

*Анализ последних исследований.* Методы поиска пересечений отрезков разделяются на детерминированные и недетерминированные. Тривиальный детерминированный алгоритм имеет временную сложность $O(n^2)$ и суть его заключается в проверке попарного пересечения отрезков. Сложнее, но эффективнее алгоритм Бентли-Отмена [2] с оценкой сложности $O((n+k)\log n+k)$, в основе которого лежит метод заметающей прямой. Алгоритм, предложенный Чазеле и Едельсбруннером [3], имеет лучшую оценку $O(n \log n + k)$, но в отличие от предыдущих методов требует квадратичной памяти. Оптимальный детерминированный алгоритм был предложен Балабаном [1] с временной оценкой сложности $O(n \log (n) + k)$ и $O(n)$ памяти. В работе [4] акцентировано внимание на отсутствие необходимости в использовании дополнительной памяти. В предлагаемой работе реализован более эффективный алгоритм, позволяющий повысить скорость работы алгоритма Балабана в 1.5 - 2.5 раза в зависимости от количества пересечений за счет выявления некоторых пересечений еще в родительских узлах графа алгоритма на основе метода "разделяй и властвуй".

*Цель работы.* Оптимизировать по времени алгоритм Балабана [1] поиска пересечения отрезков.

**Постановка задачи (*поиск пересечения отрезков*).** *Пусть заданное множество S состоящее с N отрезков на плоскости. Необходимо определить полное множество всех точек попарного пересечения отрезков S.*

## 2. АЛГОРИТМ ПОСТРОЕНИЯ РЕШЕНИЯ

### 2.1 Основные понятия и определения

Введем некоторые обозначения. Пусть *Int(S)* - множество всех точек пересечения отрезков *S*, а *|Int(S)|* - количество пересечений *K;* через $<b, e>$ обозначим вертикальную полосу, которая ограничена прямыми $x = b$ и $x = e$, а через *s* отрезок с концами абсцисс *l* и *r*. Рассмотрим взаимное расположение вертикальной полосы $<b, e>$ и отрезка *s*.

**Определение.** Будем говорить, что отрезок *s*, с концами абсцисс *l* и *r* :

- содержит(span) полосу $<b, e>$, если $l \le b \le e \le r$;
- внутренний для полосы $<b, e>$, если $b < l < r < e$;
- пересекает(strip) полосу $<b, e>$ в других случаях.

Введем отношение порядка на множестве отрезков $s_1 <_b s_2$, если оба отрезка пересекают вертикальную линию $x = b$ и точка пересечения этой прямой с отрезком $s_1$ лежит ниже точки пересечения с $s_2$.

**Определение.** *«Лестница» D* — это пара $<Q, <b, e>>$, в которой отрезки *Q* удовлетворяют следующим условиям :

- любой отрезок из *Q* содержит полосу $<b, e>$;
- нет пересечений отрезков внутри лестницы;
- *Q* упорядочена по отношению $<_b$.

Часть отрезков лестницы внутри полосы будем называть *ступеньками.*

**Определение.** Будем называть лестницу *D полностью соотносимой к множеству S*, если каждый отрезок из *S* не пересекает полосу $<b, e>$, или же пересекает одну из ступенек *D*.

**Определение.** Отрезки $s_1$ и $s_2$ будем называть пересекающимися в полосе $<b, e>$, если абсцисса точки их пересечения находится между *b* и *e*. Обозначим *Int(S, S')* множество пересечений множества отрезков *S* из *S'*.

**Предварительная обработка и структура данных.** Пусть задано множество отрезков $S=\{s_1, s_2, . s_n\}$. Упорядочим множество его конечных точек $P_i=(x_i, y_i)$ по возрастанию абсциссы, а при равенстве абсцисс - по ординате. Полученное таким образом множество обозначим через *L*. Пусть *s(i)* - номер отрезка, которому принадлежит точка $P_i$.

### 2. 2 Алгоритм

1. Введем вспомогательную функцию *Split*, которая разделяет входное множество отрезков *L*, пересекающих некоторую полосу $<b, e>$, на подмножества *Q* и *L '* так, что лестница $<Q,$

<b, e>> полностью соотносима к множеству отрезков $L'$, рис. 1.

```
procedure Split_{b,e}(L, Q, L');
{Let L = (s_1, ..., s_k), s_i <_b s_{i+1}}
  L' := ∅; Q := ∅;
  For j = 1, ..., k do
    If the segment s_j doesn't intersect
    the last segment of Q within ⟨b, e⟩ and
    spans this strip then
      add s_j to the end of Q
    else
      add s_j to the end of L';
    endif;
  endfor;
end procedure.
```

**Рисунок 1:** Разделение множества $L$ на подмножества $Q$ и $L$.

2**.** Пусть множество отрезков $L$ содержит полосу <*b, e*> и отсортировано по отношению $<_b$. Определим функцию, которая находит все пересечения множества отрезков $L$ в *полосе* <*b, e*> и формирует множество $R$ из этих же отрезков, но отсортированных по отношению $<_e$. Все отрезки в $L$ содержат *полосу* <*b, e*>. Разобьем множество $L$ на $L_1$, $L_2$, ..., $L_n$ с помощью функции *Split* следующим образом. Пусть $L_0 = L$, тогда функция *Split*($L_0$, $Q_0$, $L_1$) дает лестницу <$Q_0$, <*b, e*>>, которая полностью соотносима с множеством отрезков $L_1$. Аналогично разбивается $L_1$ на $Q_2$ и $L_2$. Процесс разбиения продолжается до тех пор, пока $L_i$ не станет равным пустому множеству, рис. 2. На следующем шаге найдем все пересечения для каждого $i$ = 1,…, $n$ между лестницей <$L_i$, <*b, e*>> и множеством $L'_i$ полностью соотносимых отрезков, которые ее пересекают. Так как оба множества являются упорядоченными по отношению $<_b$ , то данную операцию можно провести за линейное время от количества пересечений для каждого из отрезков $L_i$.

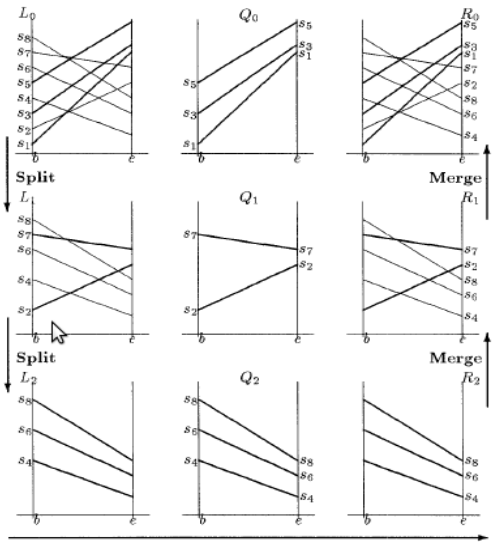



**Рисунок 2:** Пример работы алгоритма поиска всех пересечений отрезков в полосе.

Описанный алгоритм представлен на рис. 3. В нем используется процедура *Merge*, которая объединяет два множества $S$ и $S'$, отсортированные по $<_x$.

```
procedure SearchInStrip_{b,e}(L, R)
  Split(L, Q, L');
  If L' = ∅ then R := Q; exit; endif;
  Find Int_{b,e}(Q, L');
  SearchInStrip_{b,e}(L', R');
  R := Merge_e(Q, R');
end procedure.
```

**Рисунок 3:** Алгоритм поиска пересечений для случая, когда все отрезки содержат некоторую полосу.

3. Общий алгоритм нахождения всех пересечений отрезков с помощью алгоритма Балабана, рис.4.

```
procedure IntersectingPairs(S_0).
  Sort the 2N endpoints by abscissa and
    find p_i, s(i), i = 1, ..., 2N; S_r := S_0;
  TreeSearch(S_r, 1, 2N);
end procedure.

procedure TreeSearch(S_v, b, e).
1. If e − b = 1 then
   L_v:=sort S_v by <_b; SearchInStrip_{b,e}(L_v, R_v); exit;
   endif;
2. Split S_v into Q_v and S'_v so that staircase
     D_v := (Q_v, ⟨b, e⟩) be complete relative to S'_v;
3. Find Int(D_v, S'_v);
4. c := ⌊(b + e)/2⌋;
5. Place segments of S'_v
     crossing the strip ⟨b, c⟩ into S_{ls(v)} and
     the strip ⟨c, e⟩ into S_{rs(v)};
6. TreeSearch(S_{ls(v)}, b, c);
7. TreeSearch(S_{rs(v)}, c, e);
end procedure.
```

**Рисунок 4:** Основная часть алгоритма Балабана.

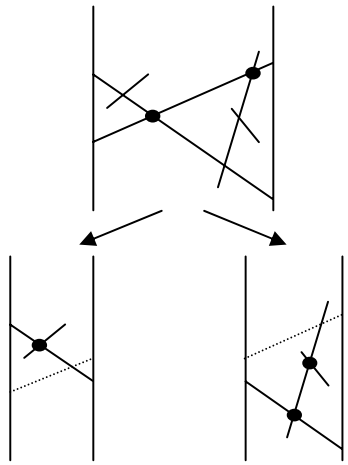
На первом шаге алгоритма выполняется поиск отрезков для случая, когда все отрезки содержат полосу. Для оптимизации проверяется только случай, когда полоса проходит через две соседние вершины упорядоченного списка вершин всех отрезков. На шаге 2 выполняется процедура *Split*, которая разбивает множества всех отрезков полосы на лестницу и соотносимое множество. На шаге 3 находим пересечения между полученными множествами с помощью описанного ранее оптимального алгоритма. На шагах 4 -5 множества всех отрезков разбивается на две части. Для этого разделяем полосу <*b, e*> на две части - <*b, c*> и <*c, e*>, где *c* медиана упорядоченного списка вершин отрезков полосы <*b, e*>. В полосу <*b, c*> войдут все внутренние и пересекающие ее отрезки. Последние шаги вызывают рекурсивно алгоритм для полос <*b, c*> и <*c, e*>. Порядок обхода здесь имеет значение, так как после выполнения шага 6 получаем упорядоченное по $<_c$ множество отрезков, необходимое для шага 7. Другими словами, сыновья дерева «разделяй и властвуй» связанны между собой. Эта особенность алгоритма не позволяет его распараллеливание, выполняя вычисления в сыновьях на разных процессорах, что является существенным недостатком алгоритма.

Из иллюстрации на рис.2 видно, что процедура *Split* возвращает в переменную $Q_v$ не все отрезки, содержащие в себе полосу <*b, e*>, и соответственно находит не все возможные пересечения между прямыми, которые содержат полосу, с пересекающими их прямыми. Эти отрезки обрабатываются в узлах сыновей, поскольку, если отрезок содержит целую полосу, то он содержит и ее части. Поэтому, для улучшения эффективности работы алгоритма предлагается ввести шаг, который при необходимости проведет дополнительное разбиение, вызвав рекурсивно функцию *TreeSearch* для этой же полосы:

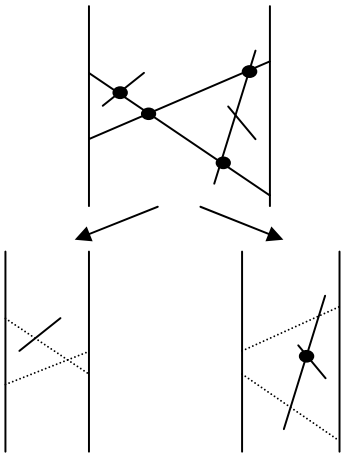
3'. **If** $|S'v| < |Int(D_v, S'_v)|$ **then**
TreeSearch($S'_v$, *b, e*); **exit**;
**endif**

## 3. ОЦЕНКА СЛОЖНОСТИ

Пусть *N* - это количество отрезков, *K* - количество пересечений. Процедура просматривает каждый из *N* отрезков. При этом расходуется *O(1)* времени на просмотр отрезка, так как вставка в конец массива выполняется за константное время. Откуда и получаем необходимую оценку.



*а*



*б*

**Рисунок 5**: Порядок нахождения пересечений: а- до оптимизации, б- после оптимизации.

Время запуска этой процедуры равно суммарному времени ее запусков. Подсчитаем время *i*-того запуска. Пусть $L_i$, $Q_i$, $L_i'$ — соответствующие множества, причем $L_0=L$, $L_{i+1}=L_i'$. Тогда *Split* и *Merge* выполняются за $O(|L_i|)$. Для каждого из отрезков время нахождения его пересечений пропорционально количеству пересечений, тогда время нахождения всех пересечений всех отрезков пропорционально количеству всех пересечений $|Int_{(b,e)}(Q_i, L_i')|$. В результате имеем общую сложность $O(|L_i|+|Int_{(b,e)}(Q_i, L_i')|)$

Процедура *IntersectingPairs* работает за время *O(Nlog (N) + K)* с использованием *O(N)* памяти.

Полное обоснование асимптотической сложности алгоритма Балабана приведено в [1].

Предложенный новый шаг 3' позволяет найти пересечения между отрезками, которые содержат полосу <*b, e*> для текущей вершины дерева алгоритма "разделяй и властвуй", но которые не были отнесены к множеству $Q_v$, с другими отрезками в родительском узле для полосы <*b, e*>. До этого шага 3' все такие отрезки разбивались на 2 части, и обрабатывались в узлах сыновей. В самом худшем случае такие отрезки будут обработаны в листьях, так как процедура *SearchInStrip* работает до тех пор, пока множество отрезков, которое передается ей на вход и содержащее полосу, не станет пустым. Однако, очевидно, что выполнять одну и туже работу многократно не эффективно. Лучше, если это можно сделать один раз. С другой стороны, возможен случай, когда количество найденных новых отрезков, содержащих полосу, будет небольшим. Тогда время их поиска будет больше времени, сэкономленного вышеприведенной оптимизацией. Поэтому было принято решение выполнять разделение множества до тех пор, пока количество найденных пересечений превышает количество отрезков, которые пересекают полосу.

На рис. 5 приведен пример сравнения работы классического алгоритма Балабана и модифицированного алгоритма. Пунктиром обозначены не обрабатываемые на данном шаге отрезки, а жирными точками — найденные на шаге пересечения.

В первом случае на первом шаге была удалена только одна прямая и найдено 2 точки пересечения, а в узлах сыновей обработано 7 отрезков. Тогда как во втором случае сразу было найдено 4 вершины. В левом поддереве алгоритма «разделяй и властвуй» остался только один отрезок, для которого не требуется дополнительных вычислений, а в правом – 2 отрезка, которые дают еще одно пересечение. Таким образом, обработано 3 отрезка в узлах сыновей с использованием оптимизации, что в 2 раза меньше, чем без нее.

## 4. РЕАЛИЗАЦИЯ

Реализация модифицированного алгоритма проводилась на языке программирования Java с использованием библиотеки *Swing* для создания интерфейса программы. Для проверки работы программы использовалось юнит-тестирование отдельных модулей, и непосредственно сверка результатов алгоритма Балабана с тривиальным алгоритмом, проверяющим попарно все пересечения за $O(N^2)$, на больших наборах случайных данных. Результат работы алгоритма представлено на рис. 6.

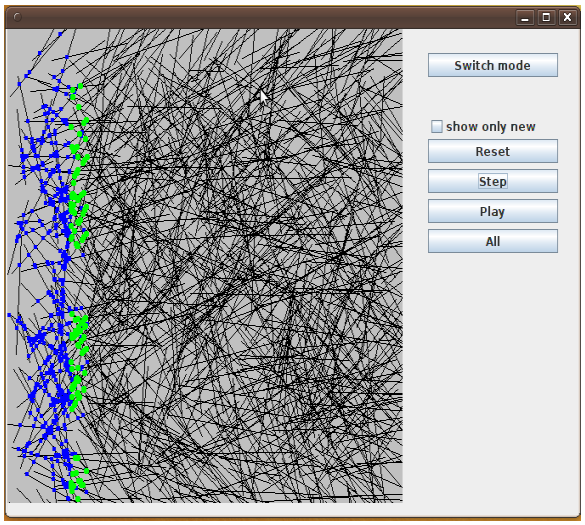



**Рисунок 6:** Пример работы программы алгоритма

## 5. ЗАКЛЮЧЕНИЕ

В работе проведен сравнительный анализ существующих алгоритмов поиска пересечения отрезков [6] и предложена эффективная модификация лучшего из них – алгоритма Балабана [1]. Из таблицы 1 видно, что уже при количестве отрезков равным 2000 и большому количеству пересечений целесообразно использовать алгоритм Балабана. Однако в результате громоздкости и высокой сложности реализации алгоритма в большинстве практических задач используется алгоритм заметающей прямой Бентли-Отмена [2].

| n | V | S | M | T | B | BM |
|---|---|---|---|---|---|---|
| 2000 | 4007 | 1,14 | 1,74 | 15,13 | 1,94 | 1,05 |
| 2000 | 4026 | 1,18 | 2,25 | 14,87 | 2,07 | 1,08 |
| 2000 | 4136 | 1,25 | 2,91 | 15,26 | 2,17 | 1,09 |
| 2000 | 4428 | 1,39 | 3,44 | 15,06 | 2,33 | 1,18 |
| 2000 | 5857 | 1,81 | 4,44 | 15,31 | 2,48 | 1,27 |
| 2000 | 10954 | 3,03 | 5,93 | 15,41 | 2,74 | 1,4 |
| 2000 | 29683 | 7,57 | 9,71 | 16,01 | 3,22 | 1,63 |
| 2000 | 91789 | 22,84 | 20,04 | 16,62 | 5,38 | 2,59 |
| 2000 | 267048 | 70,24 | 48,96 | 18,42 | 11,54 | 5,76 |

**Таблица 1:** Время выполнения алгоритмов поиска пересечений отрезков в секундах для одной и той же тестовой машины.

Здесь *n* - количество сегментов, *V* - количество граней на которые отрезки разбивают плоскость, *S* - алгоритм Бентли-Отмена, *M* - недетерминированный алгоритм Mulmuley [8], *B* - алгоритм Балабана, *T* - тривиальный алгоритм, *BM* - улучшенный алгоритм Балабана.

На рис. 7 показано сравнение эффективности работы модифицированного алгоритма и классического. Дальнейшее усовершенствование алгоритма должно заключаться в возможности запуска его на многопроцессорных системах.

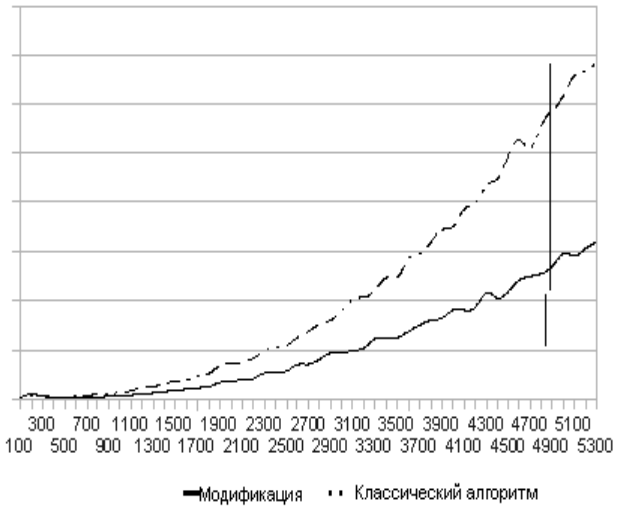



**Рисунок 7:** Зависимость времени выполнения модифицированного и классического алгоритмов от количества отрезков.

## 6. ЛИТЕРАТУРА

[1] I.J. Balaban. An optimal algorithm for finding segments intersections. *In Proceedings of the Eleventh Annual Symposium on Computational Geometry*, ACM Press, New York, 1995. - pp. 211–219.

[2] B. Chazelle. Intersecting is easier than sorting. *In Proceedings of the 16th Annual ACM Symposium on Theory of Computing,* 1984. - pp. 125– 134.

[3] B. Chazelle, H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *Journal of the ACM* , 39 (1), 1992. - pp. 1–54.

[4] Jan Vahrenhold. Line-segment intersection made in-place. *Computational Geometry* , 38 , 2007. – pp. 213–230.

[5] Goodman J.E., O'Rourke J. Handbook of discrete and computational geometry (2ed., CRC, 2004). - pp. 551-554.

[6] http://www.algorithmic-solutions.com/leda/

[7] Kurt Mehlhorn and Stefan Näher. LEDA—a platform for combinatorial and geometric computing. *Cambridge University Press,* Cambridge, UK, 2009.- pp.733-735.

[8] K. Mulmuley. A fast planar partition algorithm, I. *Journal of Symbolic Computation*, 10(3/4), 1990. -pp.:253-280.

## About the author

Vasyl Tereshchenko is an associate professor at National Taras Shevchenko University of Kyiv, Faculty of Cybernetics. His contact email is v_ter@ukr.net.

Taras Voznyuk is a student at National Taras Shevchenko University of Kyiv, Faculty of Cybernetics. His contact email is taarraas@gmail.com