برای حل مسئله موجودیت‌های Card, Column, GameState در نظر گرفته شده است.

هر کارت شاید مقدار و رنگ آن است. هر ستون شامل یک دسته از کارت‌ها است و در نهایت نیز این ستون‌ها در کنار هم یک GameState را تشکیل می‌دهند که با کمک این حالت‌ها، درخت یا گرافی تشکیل می‌دهیم که هر گره‌ی آن یک GameState است.

در روند حل مسئله تلاش می‌کنیم تا با گسترش این حالت‌ها در گراف، به حالت هدفمان برسیم.

(BFS

در این الگوریتم با کمک یک مجموعه Explored برای نگهداری گره‌هایی که تا کنون آن‌ها را بسط داده‌ایم و یک مجموعه Frontier برای نگهداری گره‌هایی که کاندید برای بسط دادن هستند، مسئله را جلو می‌بریم به این صورت که از ریشه شروع می‌کنیم و تمامی گره‌هایی که می‌توانیم با عوض کردن جای یک کارت بین ستون‌ها، به آن برسیم را پیدا می‌کنیم سپس بررسی می‌کنیم که آیا این گره در Explored وجود دارد یا خیر؛ اگر وجود داشت که آن را رها می‌کنیم اما اگر وجود نداشت، آن را به مجموعه Frontier اضافه می‌کنیم. مجموعه‌ی Frontier ما یک صف است یعنی هر گره‌ای که زودتر به آن وارد شده باشد، زودتر از آن خارج می‌شود پس در ادامه نیز در هر مرحله یک گره را از این مجموعه خارج می‌کنیم و همان کارهایی را که در مرحله قبل با ریشه گراف انجام دادیم را انجام می‌دهیم.

هر گاه گره‌ای از مجموعه Frontier خارج شد، بررسی می‌کنیم که آیا این گره برابر با حالت هدف ما که همان مرتب شدن کارت‌های هم‌رنگ در ستون مجزا است، می‌باشد یا خیر؛ در صورت برابری این گره به عنوان نتیجه مسئله برگردانده می‌شود و از طریق این گراف، مسیر رسیدن به این گره بدست می‌آید.

(IDS

در این الگوریتم که به نوعی از ویژگی‌های هر دو الگوریتم BFS, DFS استفاده می‌کند به این صورت که تا عمق خاصی الگوریتم DFS را پیاده می‌کنیم که البته به خاطر محدود بودن عمق آن، به آن DLS می‌گوییم و پس از پایان آن یک واحد، عمق را افزایش می‌دهیم و دوباره الگوریتم را تکرار می‌کنیم.

خود الگوریتم DLS هم به صورت بازگشتی پیاده‌سازی شده و شرط پایان آن رسیدن به حالتی است که محدودیت عمق آن برابر صفر شود یا اینکه به یک گره‌ی هدف برسد.

(A*

در این الگوریتم مشابه BFS عمل می‌کنیم و یک تفاوت آن وجود یک هیوریستیک در هر گره است که باعث می‌شود خارج کردن گره‌ها از مجموعه‌ی Frontier شبیه یک صف معمولی نباشد بلکه گره‌ای را از آن خارج می‌کنیم که مجموع هیوریستیک (هزینه تخمینی تا هدف) و عمق آن (هزینه‌ی ریشه تا خود این گره) کمتر از سایر گره‌های موجود در این مجموعه باشد.

تفاوت دیگر آن در محاسبه‌ی تابع هیوریستیک، پیش از اضافه کردن هر گره به گراف است. هیوریستیکی که برای این مسئله در نظر گرفته شده به این صورت عمل می‌کند که می‌رود تمامی ستون‌های موجود در گره را بررسی می‌کند و اگر خالی نباشد، آن کارتی را که زیر بقیه کارت‌های این ستون هستند را نگاه می‌کند، اگر این کارت برابر با حداکثر مقدار ممکن برای شماره کارت که همان n است نباشد، یک واحد به مقدار هیوریستیک اضافه می‌کند.

بررسی نتیجه استفاده از الگوریتم‌ها)

نتیجه‌هایی که در استفاده از این الگوریتم‌ها بدست آمد، در بیشتر موارد نشان می‌داد که این الگوریتم‌ها در عمق یکسانی به جواب می‌رسیدند اما در تعداد گره‌های تولید شده و بسط داده شده با هم متفاوت اند؛ به این صورت که الگوریتم A* با تعداد گره کمتری نسبت به بقیه به جواب می‌رسید و پس از آن نیز الگوریتم BFS با تعداد گره کمتری نسبت به IDS به جواب مسئله که همان مرتب بودن لیست کارت‌ها بود، می‌رسید.